Merica

EVENTMAP

25-28 AUG 2015

MISTA 2015 PROCEEDINGS

25 -28 Aug 2015 | Prague, Czech Republic
ISSN: 2305-249X

MISTA 2015

Proceedings of the

7th Multidisciplinary International Conference on Scheduling: Theory and Applications

25th – 28th August 2015
Prague, Czech Republic

Edited by
Zdenek Hanzálek, Czech Technical University in Prague
Graham Kendall, University of Nottingham, UK/Malaysia
Barry McCollum, Queens University Belfast, UK
Premysl Šůcha, Czech Technical University in Prague

MISTA 2015 Conference Program Committee

MISTA 2015 International Advisory Committee

- Graham Kendall (chair)
- Abdelhakim Artiba, Facultes Universitares Catholiques de Mons (CREGI - FUCAM), Belguim
- James Bean, University of Michigan, USA
- Jacek Blazewicz, Institute of Computing Science, Poznan University of Technology, Poland
- Edmund Burke, The University of Nottingham, UK
- Xiaoqiang Cai, The Chinese University of Hong Kong, Hong Kong
- Ed Coffman, Columbia University, USA
- Moshe Dror, The University of Arizona, USA
- David Fogel, Natural Selection Inc., USA
- Michel Gendreau, University of Montreal, Canada
- Fred Glover, Leeds School of Business, University of Colorado, USA
- Bernard Grabot, Laboratoire Génie de Production - ENIT, Tarbes, France
- Toshihide Ibaraki, Kyoto University, Japan
- Claude Le Pape, ILOG, France
- Ibrahim Osman, American University of Beirut, Lebanon
- Michael Pinedo, New York University, USA
- Jean-Yves Potvin, Université de Montréal, Canada
- Michael Trick, Graduate School of Industrial Administration, Carnegie Mellon University, USA
- Stephen Smith, Carnegie Mellon University, USA
- Steef van de Velde, Erasmus University, Netherlands
- George White, University of Ottawa, Canada
- Gerhard Woeginger, University of Twente, Netherlands

Acknowledgements

This conference would not have been possible without the assistance of a great many people.

Any scientific conference is underpinned by the quality of the papers that it publishes. In turn, this is a function of the quality of the Program Committee. MISTA is fortunate enough to have many of the world's scheduling experts as members of the Program Committee. We are extremely grateful for the time and effort that they devote to making MISTA the success it is.

The editors would also like to thank the international advisory committee for their helpful advice and comments as we plan each conference and seek to develop the MISTA series year-on-year. Their advice is always insightful and made in the best interests of the conference.

We are grateful to the chairs of the special session on **Educational Timetabling** (Barry McCollum and Hana Rudová), **Hyperheuristics in Scheduling** (Nelishia Pillay and Rong Qu), **Cloud Based Resource Scheduling** (Per-Olov Östberg and Barry McCollum) and **Operations Research Models for Scheduling Problems with Preventive Maintenance** (Rachid Benmansour and Hamid Allaoui) for taking the time to collect together an excellent set of papers. We really appreciate the time and effort you gave to the conference.

We greatly appreciate the support that we have received from EventMap Ltd., which have supported the conference once again. We are also very grateful to the Mercia s.r.o, specifically in sponsoring the program booklet.

We would also like to thank the Journal of Scheduling which supports the MISTA conference series by allowing us to guest edit a special issue of the journal which is associated with the conference. This certainly adds to the conference and the post-conference opportunities.

We are also grateful for the continued support we have received from the University of Nottingham (both UK and Malaysia campuses).

Thanks must also go to a dedicated team from the Czech Technical University in Prague. Without the support from a local team, MISTA would not happen, and certainly not as efficiently as it does with the hard work that is vital to the success of MISTA.

MISTA 2015 is the venue where the results of the Nurse Rostering Competition will be announced. We'd like to acknowledge the hard work of the team that organised this competition; Patrick De Causmaecker, Andrea Schaerf, Stefaan Haspeslagh, Nguyen Thi Thanh Dang and Sara Ceschia.

As in previous years, the conference owes a great deal to Debbie Pitchfork. She has worked tirelessly since performing similar tasks for MISTA 2009, 2011 and 2013. If it were not for Debbie this conference would not have taken place. Thank you, from all who are involved in MISTA 2015, whether are part of the organisational team, or delegates.

Table of Contents

Plenary Presentations

Papers

Abstracts

Plenary Presentations

Hana Rudová

University course timetabling: From theory to practice

Abstract

University course timetabling introduces diverse complex problems which may be very different in correspondence with country, institution, or even a school. Complexity of the problems is related with the size represented by the number of courses and students and with the characteristics such as curricula structure, course structure, or involved optimization criteria. In practice, standard benchmark problems represented by enrollment-based and curriculum-based timetabling must be extended by elective courses, course sections and issues related with fairness of generated timetables. Compactness of timetables common in high school timetabling becomes very complex issue given the diversity of student timetables.

Theoretical advances with respect to above mentioned issues will be discussed. Practical application and experience with timetabling at two different institutions in Europe and United States will be summarized.

Martin Skutella

LP-based algorithms for scheduling unrelated parallel machines

Abstract

Since the early days of scheduling, algorithms and techniques from the closely related area of mathematical programming have played a pivotal role. In this talk, we focus on the use of linear programming (LP) relaxations in the design of approximation algorithms for NP-hard scheduling problems. The classical problem of scheduling jobs on unrelated parallel machines subject to release dates and with total weighted completion time objective and special cases of this problem serve as an example for which we discuss the strengths and weaknesses of different types of linear and convex programming relaxations. We also discuss recent LP-based approximation results for a stochastic variant of unrelated parallel machine scheduling (joint work with Maxim Sviridenko and Marc Uetz).

Johann Hurink

Scheduling for Decentralized Energy Management

Abstract

Our energy systems undergo a fundamental change. Where in the past the energy mainly was generated in large power plants using fossil fuels, in the future a large part of the generation will result from small plants in decentralized locations using uncontrollable renewable sources. This leads to a loss of control over a larger fraction of the generation. To be able to compensate for this loss in flexibility on the generation side, we have to create and use flexibility on the consumption side. This has led to the concept of ‘Smart Grids’ and decentralized energy management is seen as a key element for these Smart Grids.

In this talk we first give a sketch of possible concepts and methods for decentralized energy management. Furthermore, we argue that devices with inherent storage offer large portions of flexibility and discus a range of scheduling problems that come up for these devices. Finally, for a few of these problems we sketch possible solution approaches.

Papers

MISTA 2015

A Probabilistic Constrained Approach for Unrelated Parallel Machine Scheduling

Pablo Adasme · Janny Leung · Abdel Lisser

Abstract In this paper, we investigate a probabilistic constrained variant of the well known unrelated parallel machine scheduling problem. For this purpose, we assume that each vector of job processing times is an independent and multivariate normally distributed vector with known mean and covariance matrix. This assumption allows transforming the probabilistic constraints into deterministic equivalent second order conic constraints [9]. In particular, we consider the problem of makespan minimization when completing a subset of jobs subject to machine energy consumption and job assignment constraints. We compute feasible solutions by solving directly the equivalent deterministic mixed integer second order conic (MISOC) programming problem and also by means of piecewise mixed integer linear programming (MILP) formulations we obtain from the MISOC problem. Our numerical results indicate that one of the piecewise linear formulations allows finding better feasible solutions for instances with up to ten machines and fifty jobs in less average computational cost.

1 Introduction

The unrelated parallel machine scheduling problem has been studied since several decades ago [11,13]. In general, the problem consists of assigning a set of jobs to a set of different parallel machines such that each job is processed in an unique machine and the worst maximum completion time (makespan) of all machines is minimized. In this paper, we investigate a new probabilistic constrained variant of this well known

Pablo Adasme
Departamento de Ingeniería Eléctrica, Universidad de Santiago de Chile, Avenida Ecuador 3519 Santiago, Chile.
E-mail: pablo.adasme@usach.cl

Janny Leung
Department of Systems Engineering & Engineering Management, Chinese University of Hong Kong, Shatin, Hong Kong.
E-mail: janny@se.cuhk.edu.hk

Abdel Lisser
Laboratoire de Recherche en Informatique, Université Paris-Sud XI, Bâtiment 650, 91405 Orsay Cedex France.
E-mail: abdel.lisser@lri.fr

machine scheduling problem. For this purpose, we assume that each vector of job processing times is an independent and multivariate normally distributed vector with known mean and covariance matrix. This assumption allows transforming the probabilistic constraints into deterministic equivalent second order conic constraints [9]. In particular, we consider the problem of makespan minimization when completing a subset of jobs subject to machine energy consumption and job assignment constraints. We consider range machine energy constraints that help balancing the amount of work to be processed by the different machines in a given period of time (a day, a week, etc.). So far, we assume that the energy consumptions required by the machines to process the different jobs are deterministic input data for the problem. We compute feasible solutions by solving directly the equivalent deterministic mixed integer second order conic (MISOC) programming problem and also by means of piecewise mixed integer linear programming (MILP) formulations we derive from the MISOC problem. Piecewise linear approximations allows transforming nonlinear programming problems into pure MILP problems that can be efficiently handled by specialized solvers [1]. We refer the reader to [3,5,8,10,12] for a deeper comprehension on this subject. Stochastic programming (SP), on the other side, is an optimization technique which helps dealing with the uncertainty of the input parameters of a mathematical program [14,16]. In SP, the input parameters are modeled as random variables and thus, the theory of probabilities can be applied. The probability distributions governing the data are commonly known or can be estimated. Probabilistic or chance constrained programs are stochastic optimization problems where a subset of the whole constraints is satisfied for at least a prescribed threshold. For a deeper understanding on probabilistic constrained approaches, we refer the reader to [9,14,16] and references therein. As far as we know, joint probabilistic constrained approaches and piecewise linear formulations have not yet been investigated so far for parallel machine scheduling problems.

This paper is organized as follows. In section 2, we present the new probabilistic constrained parallel machine scheduling problem. Then, in section 3 we obtain a deterministic equivalent MISOC formulation and present two piecewise MILP formulations that we derive from the MISOC problem. Subsequently, in section 4 we present numerical results in order to compare the performance of the proposed models. Finally, in section 5 we conclude the paper.

2 Problem formulation

In order to state our stochastic version of the parallel machine scheduling problem, we consider a set of jobs $\mathcal{J} = \{1, .., n\}$ to be processed by a set of parallel machines $\mathcal{M} = \{1, .., m\}$. The processing time required by machine $i \in \mathcal{M}$ to process job $j \in \mathcal{J}$ is denoted by $p_{ij}(\xi)$ where ξ is a random variable normally and independently distributed. We denote the energy cost for processing job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ by c_{ij}. Finally, we denote by $\underline{C}_i$ and $\overline{C}_i$, the minimum and maximum energy values that machine $i \in \mathcal{M}$ is allowed to use for processing jobs. We consider the following probabilistic constrained optimization problem

$$P_0 : \min_{\{x, c_{max}\}} c_{max} \tag{1}$$

$$\text{s.t.: } \mathbf{P}\left\{\sum_{j=1}^{n} p_{ij}(\xi) x_{ij} \leq c_{max}\right\} \geq (1-\alpha), \quad \forall i \in \mathcal{M} \tag{2}$$

$$\underline{C}_i \leq \sum_{j=1}^{n} c_{ij} x_{ij} \leq \overline{C}_i, \quad \forall i \in \mathcal{M} \tag{3}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \mathcal{J} \tag{4}$$

$$x_{ij} \in \{0,1\}, \quad \forall i,j \tag{5}$$

where $x_{ij} = 1$ if job $j \in \mathcal{J}$ is assigned machine $i \in \mathcal{M}$ and $x_{ij} = 0$ otherwise. The nonnegative continuous variable c_{max} denotes the makespan, i.e. the maximum completion time of all machines. In P_0, the objective function represents the makespan whereas constraint (2) is a generic probabilistic constraint imposed on the total time required for machine $i \in \mathcal{M}$ to process all the jobs assigned to it. The parameter $\alpha \in [0, 0.5)$ represents the risk of not satisfying the probabilistic constraint for some occurrences of $p_{ij}(\xi)$. Constraint (3) is a fairness range energy capacity constraint that each machine must respect. The underlying idea of this constraint is to balance the amount of work to be processed by each machine. Constraint (4) ensures that each job must be assigned to at most one machine. Notice that when constraint (4) is less than one for a particular $\bar{j} \in \mathcal{J}$, it means that job $\bar{j}$ is postponed for a next processing period. Finally, constraint (5) is a domain constraint for the decision variables.

3 MISOC and piecewise MILP formulations

In this section, first we present a deterministic equivalent formulation for P_0. Then, we present two piecewise MILP formulations that we derive from the deterministic equivalent model.

3.1 Deterministic equivalent MISOC formulation

In order to obtain a deterministic equivalent formulation for P_0, we assume that each input vector $p_{i,\bullet}(\xi)$ is an independent multivariate random variable normally distributed with known mean $(\bar{p}_{i,\bullet})$. Also, let $\Sigma^i = (\Sigma^i_{lj})\ \forall l,j \in \mathcal{J}, i \in \mathcal{M}$ be the corresponding covariance matrix for vector $(\bar{p}_{i,\bullet})$. This allows writing the following deterministic equivalent model [9]

$$\begin{aligned}
P_1: \quad & \min_{\{x, c_{max}\}} c_{max} \\
\text{s.t.:} \quad & \sum_{j=1}^{n} \bar{p}_{ij} x_{ij} + F^{-1}(1-\alpha) \sqrt{\sum_{l=1}^{n} \left(\sum_{j=1}^{n} \Sigma^i_{lj} x_{ij} \right)^2} \leq c_{max}, \quad \forall i \in \mathcal{M} \qquad (6) \\
& \underline{C}_i \leq \sum_{j=1}^{n} c_{ij} x_{ij} \leq \overline{C}_i, \quad \forall i \in \mathcal{M} \\
& \sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \mathcal{J} \\
& x_{ij} \in \{0,1\}, \quad \forall i,j
\end{aligned}$$

where $F^{-1}(1-\alpha)$ denotes the inverse of $F(1-\alpha)$ which is the standard normal cumulative distribution function. Notice that P_1 is formulated as a MISOC programming problem. Hereafter, we denote its convex nonlinear programming relaxation by RP_1.

3.2 Piecewise MILP formulations

In order to obtain a first piecewise MILP formulation for P_1, we linearize constraint (6) as follows. We replace the argument in the root square term by the nonnegative continuous variable $y_i \geq 0, \forall i \in \mathcal{M}$. This allows writing constraint (6) by means of the following set of constraints

$$\sum_{j=1}^{n} \bar{p}_{ij} x_{ij} + F^{-1}(1-\alpha)\sqrt{y_i} \leq c_{max}, \quad \forall i \in \mathcal{M} \tag{7}$$

$$y_i = \sum_{l=1}^{n} \left(\sum_{j=1}^{n} \Sigma_{lj}^{i} x_{ij} \right)^2 \geq 0, \forall i \in \mathcal{M} \tag{8}$$

$$y_i \geq 0, \forall i \in \mathcal{M}$$

Next, we approximate each root square term $\sqrt{y_i}, \forall i \in \mathcal{M}$ with a set of line segments $\mathcal{S} = \{1, .., S\}$. Subsequently, we replace the term $\sqrt{y_i}$ in (7) by the expression $\sum_{s=1}^{S} (a(s) y_i \varphi_{si} + b(s)\varphi_{si})$ such that $\sum_{s=1}^{S} \varphi_{si} = 1, \forall i \in \mathcal{M}$. The parameters $a(s)$ and $b(s)$ are the slopes and constant terms of each line segment $s \in \mathcal{S}$. Next, we introduce the following bounding constraints for each $y_i, i \in \mathcal{M}$.

$$-(1-\varphi_{si})\mathcal{T} + Lo(s) \leq y_i \leq Up(s) + (1-\varphi_{si})\mathcal{T}, \quad \forall i \in \mathcal{M}, s \in \mathcal{S}$$

where $\mathcal{T}$ represents a large positive value. The parameters $Lo(s)$ and $Up(s)$ correspond to lower and upper bounds for each line segment $s \in \mathcal{S}$. This allows to control the value of variable y_i by using the binary variables φ_{si}. Hence, if $\varphi_{\tilde{s}i} = 1$, it means that y_i lies in the line segment $\tilde{s} \in \mathcal{S}$. Each y_i can only lie in an unique line segment which is controlled by the constraint $\sum_{s=1}^{S} \varphi_{si} = 1, \forall i \in \mathcal{M}$. Finally, by the use of standard linearization techniques [4,6], we can write the following equivalent piecewise MILP formulation

$$P_2 : \min_{\{x,\theta,\phi,\varphi,y,c_{max}\}} \quad c_{max}$$

$$\text{s.t.:} \quad \sum_{j=1}^{n} \bar{p}_{ij} x_{ij} + F^{-1}(1-\alpha) \sum_{s=1}^{S} (a(s)\phi_{is} + b(s)\varphi_{si}) \leq c_{max}, \quad \forall i \in \mathcal{M}$$

$$\sum_{s=1}^{S} \varphi_{si} = 1, \quad \forall i \in \mathcal{M}$$

$$-(1-\varphi_{si})\mathcal{T} + Lo(s) \leq y_i \leq Up(s) + (1-\varphi_{si})\mathcal{T}, \quad \forall i \in \mathcal{M}, s \in \mathcal{S}$$

$$\phi_{is} \leq y_i, \quad \forall i, s \tag{9}$$

$$\phi_{is} \leq \varphi_{si}\mathcal{T}, \quad \forall i, s \tag{10}$$

$$\phi_{is} \geq y_i + \varphi_{si}\mathcal{T} - \mathcal{T}, \quad \forall i, s \tag{11}$$

$$\phi_{is} \geq 0, \quad \forall i, s \tag{12}$$

$$\underline{C}_i \leq \sum_{j=1}^{n} c_{ij} x_{ij} \leq \overline{C}_i, \quad \forall i \in \mathcal{M}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \mathcal{J}$$

$$y_i - \sum_{l=1}^{n}\sum_{j=1}^{n} \left(\Sigma_{lj}^i\right)^2 x_{ij} + \sum_{l=1}^{n} \left(\sum_{j=1}^{n} \sum_{\substack{k=1 \\ (j\neq k)}}^{n} \Sigma_{lj}^i \Sigma_{lk}^i \theta_{jk}^i \right) = 0, \quad \forall i \in \mathcal{M} \qquad (13)$$

$$\theta_{jk}^i \leq x_{ij}, \quad \forall i,j,k \qquad (14)$$

$$\theta_{jk}^i \leq x_{ik}, \quad \forall i,j,k \qquad (15)$$

$$\theta_{jk}^i \geq x_{ij} + x_{ik} - 1, \quad \forall i,j,k \qquad (16)$$

$$\theta_{jk}^i \in \{0,1\}, \quad \forall i,j,k \qquad (17)$$

$$x_{ij} \in \{0,1\}, \forall i,j, \quad y_i \geq 0, \forall i$$

where the constraints (9)-(12) linearize the quadratic terms $y_i\varphi_{si}, \forall i \in \mathcal{M}, s \in \mathcal{S}$ and the constraints (14)-(17) linearize $x_{ij}x_{ik}, \forall i \in \mathcal{M}, j,k \in \mathcal{J}$, respectively. In particular, the quadratic terms $x_{ij}x_{ik}$ arise when linearizing equation (8) that we transform into constraint (13). Hereafter, we denote by LP_2 the LP relaxation of P_2.

An alternative convex piecewise MILP based formulation for P_1 can be constructed as follows. Consider the tabular data for each root square term $z_i = \sqrt{y_i}, \forall i \in \mathcal{M}$ in (7) for a given interval $y_i \in [0, U]$. In our case, we may use for instance the upper bound $U = max_{\{i\in\mathcal{M}\}} \left\{ \sum_{l=1}^{n} \left(\sum_{j=1}^{n} \Sigma_{lj}^i \right)^2 \right\}$. Now, let us denote this data by the points $(\bar{y}_k; \bar{z}_k)$, $k \in \mathcal{K} = \{1, \ldots, K\}$. By introducing continuous nonnegative variables $\lambda_{ik}, \forall i \in \mathcal{M}, k \in \mathcal{K}$, accordingly we can write the following equivalent MILP formulation

$$P_3 : \min_{\{x,y,z,\lambda,\theta,c_{max}\}} \quad c_{max}$$

$$\text{s.t.:} \quad \sum_{j=1}^{n} \bar{p}_{ij} x_{ij} + F^{-1}(1-\alpha) z_i \leq c_{max}, \quad \forall i \in \mathcal{M}$$

$$y_i = \sum_{k=1}^{K} \lambda_{ik} \bar{y}_k, \quad \forall i \in \mathcal{M} \qquad (18)$$

$$z_i = \sum_{k=1}^{K} \lambda_{ik} \bar{z}_k, \quad \forall i \in \mathcal{M} \qquad (19)$$

$$\sum_{k=1}^{K} \lambda_{ik} = 1, \quad \forall i \in \mathcal{M} \qquad (20)$$

$$\lambda_{ik} \geq 0 \text{ and SOS-2}, \quad \forall i \in \mathcal{M}, k \in \mathcal{K}$$

$$0 \leq y_i \leq U, \quad \forall i \in \mathcal{M}$$

$$\underline{C}_i \leq \sum_{j=1}^{n} c_{ij} x_{ij} \leq \overline{C}_i, \quad \forall i \in \mathcal{M}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \mathcal{J}$$

$$y_i - \sum_{l=1}^{n}\sum_{j=1}^{n}\left(\Sigma_{lj}^{i}\right)^2 x_{ij} + \sum_{l=1}^{n}\left(\sum_{j=1}^{n}\sum_{\substack{k=1\\(j\neq k)}}^{n}\Sigma_{lj}^{i}\Sigma_{lk}^{i}\theta_{jk}^{i}\right) = 0, \quad \forall i \in \mathcal{M}$$

$$\theta_{jk}^{i} \leq x_{ij}, \quad \forall i,j,k$$

$$\theta_{jk}^{i} \leq x_{ik}, \quad \forall i,j,k$$

$$\theta_{jk}^{i} \geq x_{ij} + x_{ik} - 1, \quad \forall i,j,k$$

$$\theta_{jk}^{i} \in \{0,1\}, \quad \forall i,j,k$$

$$x_{ij} \in \{0,1\}, \forall i,j$$

where the constraints (18) and (19) are the domain and function evaluation constraints for the root square terms in (7), respectively. Constraints in (20) are convexity constraints. Finally, we impose that $\lambda_{i,k} \geq 0$, $\forall i,k$ be of type SOS-2 which means that only two consecutive variables $\lambda_{i,k-1}, \lambda_{i,k}, \forall k = \{2,\ldots,K\}$ may be non-zero [7]. We denote by LP_3 the LP relaxation of P_3. Finally, we also consider the case where $\Sigma_{lj}^{i} = 0$ $\forall i \in \mathcal{M}, l,j \in \mathcal{J}$. In this case, P_1 reduces to the following MILP problem

$$P_4: \min_{\{x,c_{max}\}} c_{max}$$

$$\text{s.t.:} \sum_{j=1}^{n} \bar{p}_{ij}x_{ij} \leq c_{max}, \quad \forall i \in \mathcal{M}$$

$$\underline{C}_i \leq \sum_{j=1}^{n} c_{ij}x_{ij} \leq \overline{C}_i, \quad \forall i \in \mathcal{M}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \mathcal{J}$$

$$x_{ij} \in \{0,1\}, \quad \forall i,j$$

We use P_4 only as an alternative way to compute feasible solutions for P_1. Thus, in our numerical results, we evaluate the feasible solutions obtained with P_4 in the objective function of P_1 in order to give some insight with respect to the distance between these two objective functions.

4 Numerical results

In this section, we present numerical results for all the proposed models: P_1, RP_1, P_2, LP_2, P_3, LP_3 and P_4. We implement a Matlab program using SBB and CONOPT solvers [2,15] for solving P_1 and RP_1 respectively and CPLEX 12 [7] to solve the MILP and LP models. All these solvers are used with default options. So far, we generate the input data randomly and arbitrarily. In a larger version of this work, we will consider input data of more realistic applications as well. The processing times $\bar{p}_{i,j}$ and the energy costs $c_{i,j}$ are drawn from the intervals $[0,20]$ and $[0,50]$, respectively. Each entry in matrix $(\Sigma_{lk}^{i}), \forall i \in \mathcal{M}, l,j \in \mathcal{J}$ is drawn from the interval $[0,2]$. The risk parameter $\alpha = 0.1$. The input parameters $\underline{C}_i, \overline{C}_i, \forall i \in \mathcal{M}$ are computed as

$$\underline{C}_i = 0.4\frac{\sum_{j=1}^{n} c_{ij}}{m}$$

#	Inst. Dim.		MISOC model				Linear programs				
	m	n	P_1	RP_1	$Time\ P_1$	$Time\ RP_1$	P_2	LP_2	$Time\ P_2$	$Time\ LP_2$	$P_1(x^2)$
Small size instances											
1	2	6	13.32	8.78	0.67	0.09	13.32	5.97	0.22	0.11	13.32
2	2	10	21.82	19.19	0.76	0.09	21.82	11.36	2.34	0.17	21.82
3	2	20	32.80	25.28	0.95	0.09	32.80	10.40	39.69	0.94	32.80
4	4	10	11.20	8.43	0.97	0.09	11.20	3.91	2.40	0.25	11.20
5	4	20	22.72	14.68	83.37	0.09	22.72	6.58	384.73	1.81	22.72
6	4	30	29.94	26.08	2196.61	1.54	29.94	6.34	1894.14	8.31	29.94
Medium and large size instances											
7	2	30	51.05	49.33	1.64	0.09	51.05	16.02	31.37	0.48	51.05
8	2	40	72.35	70.40	20.14	0.09	72.35	21.91	547.92	1.55	72.35
9	2	50	79.06	75.17	41.11	0.09	79.06	14.49	573.91	2.20	79.06
10	4	40	$*$	35.73	$*$	7.33	40.15	10.51	2421.86	1.96	40.15
11	4	50	39.99	39.26	98.02	0.16	39.99	9.46	2473.60	7.60	39.99
12	8	20	11.92	5.56	26.65	0.11	11.96	2.92	3.56	0.51	11.92
13	8	30	14.64	10.52	214.11	0.23	14.69	4.59	98.89	1.54	14.64
14	8	40	$*$	14.36	$*$	6.15	21.69	2.58	696.09	3.88	21.68
15	8	50	$*$	17.03	$*$	3.63	23.27	3.42	3600	14.02	23.27
16	10	30	$*$	10.10	$*$	2.06	14.56	2.30	1546.51	9.44	14.55
17	10	40	$*$	10.92	$*$	2.87	14.69	3.12	1741.43	9.47	14.67
18	10	50	$*$	14.43	$*$	5.32	21.77	1.31	3296.85	13.37	21.77
min.			11.20	5.56	0.67	0.09	11.20	1.31	0.22	0.11	11.20
max.			79.06	75.17	2196.61	7.33	79.06	21.91	3600	14.02	79.06
ave.			$*$	25.29	$*$	1.67	29.84	7.62	1075.30	4.31	29.83

$*$: No solution found due to a SBB shortage of memory in at most 1 hour.

Table 1 Feasible solutions obtained for the MISOC and linear programs.

and

$$\overline{C}_i = 0.7\frac{\sum_{j=1}^{n} c_{ij}}{m}$$

respectively. For the root square term in (7) that we use in our piecewise linear formulations, we consider the interval $[0, U]$ divided in subintervals of length one, on small size instances, for each line segment where U is computed as mentioned in subsection 3.2. While for medium and large size instances of the problem, we use the interval $[0, U]$ divided in subintervals of length 10 for each line segment. Finally, we set the parameter $\mathcal{T} = 1e^{10}$ in P_2. The numerical experiments have been carried out on an Intel(R) 64 bits core(TM) with 3.4 Ghz and 8 GoBytes of RAM. In each row of Tables 1 and 2, we present numerical results for the same instances. In particular, rows 1-6 present numerical results for small size instances of the problem while in rows 7-18, we present numerical results for medium and large size instances of the problem. In Table 1, column 1 shows the instance number. Columns 2-3 show the instances dimensions. In columns 4-7, we present the optimal solution values of P_1 and RP_1, and their CPU time in seconds, respectively. Similarly, in columns 8-11, we present the optimal solution values of P_2 and LP_2, and their CPU time in seconds, respectively.

Finally, in column 12 we present the objective function value of P_1 obtained with the optimal solution of P_2 or with the best solution found with P_2 in one hour. As for the medium and large size instances, we set the maximum time available to solve P_1, P_2, P_3 and P_4 be at most one hour. In Table 2, column 1 shows the instance number which is the same as for Table 1. Columns 2-5 show the optimal solutions of P_3 and LP_3, and their CPU time in seconds, respectively. Next, in column 6 we show the objective function value of P_1 obtained with the optimal solution of P_3 or with the best solution found with P_3 in one hour. Finally, columns 7-9 present the optimal

#	Linear programs							
	P_3	LP_3	$Time\ P_3$	$Time\ LP_3$	$P_1(x^3)$	P_4	$Time\ P_4$	$P_1(x^4)$
Small size instances								
1	13.32	6.66	0.11	0.09	13.32	9.06	0.09	13.32
2	21.82	12.98	0.25	0.14	21.82	13.08	0.09	21.82
3	32.80	13.01	6.18	1.04	32.80	14.01	0.09	32.80
4	11.20	4.29	0.44	0.14	11.20	5.53	0.09	11.20
5	22.72	7.28	678.37	1.45	22.72	10.79	0.09	23.39
6	29.94	8.90	918.37	19.95	29.94	8.75	0.09	30.92
Medium and large size instances								
7	51.05	21.47	8.30	0.31	51.05	16.56	0.11	54.22
8	72.35	31.16	161.97	0.79	72.35	23.56	0.13	89.57
9	79.06	24.26	301.44	3.21	79.06	14.69	0.13	88.95
10	†	12.68	3600	2.18	†	12.51	0.14	44.72
11	†	12.83	3600	4.73	†	10.11	0.11	56.32
12	11.90	3.04	1.01	0.27	11.92	5.28	0.11	11.92
13	14.62	4.90	13.99	0.81	14.64	6.23	0.09	18.79
14	†	3.57	3600	2.81	†	3.84	0.09	30.03
15	†	4.06	3600	8.42	†	4.28	0.09	24.50
16	14.50	2.69	37.36	0.97	14.52	4.63	0.09	20.86
17	14.66	3.33	77.44	3.46	14.67	4.27	0.09	22.35
18	†	1.69	3600	11.34	†	1.83	0.13	31.88
min.	11.20	1.69	0.11	0.09	11.20	1.83	0.09	11.20
max.	79.06	31.16	3600	19.95	79.06	23.56	0.14	89.57
ave.	†	9.93	1122.51	3.45	†	9.39	0.10	34.87

†: No solution found with CPLEX in 1 hour.

Table 2 Feasible solutions obtained for the linear models.

solution of P_4, its CPU time in seconds and the objective function value of P_1 obtained with the optimal solution of P_4.

From Tables 1 and 2, first we see that the objective values of P_1, $P_1(x^2)$ and $P_1(x^3)$ are exactly the same for the small size instances of the problem. This means that P_1, P_2 and P_3 allow finding the optimal solution of the problem. Notice that solving directly P_1 allows finding the optimal solution of the problem whereas the piecewise linear formulations can only guaranty a feasible solution of the problem, possibly the optimal solution when the line segments considered grows to infinity.

Regarding the medium and large size instances of the problem, we observe that not all the instances can be solved to optimality with P_1 and P_3 in one hour of CPU time. Also, we see that the average CPU time is lower for P_2 than for P_3 and that P_2 can find optimal solutions in less than one hour for most of the instances which is not possible to achieve with P_1 and P_3. In general, we observe that P_4 can solve all the instances to optimality in both Tables 1 and 2 and in significantly less CPU time when compared to P_1, P_2 and P_3 respectively. However, the feasible solutions obtained with P_4 are not near optimal for P_1. This can be verified by computing the average $Gap = \frac{P_1(x^4)-P_1(x^2)}{P_1(x^2)} * 100$ which is 16.88%. Finally, we observe that the average CPU time for LP_2 is higher than for RP_1 and LP_3. In general, from our preliminary numerical results presented in Tables 1 and 2, we mainly observe that P_2 is more effective than P_1 and P_3 as it allows solving more instances (e.g., instances 1-14, 16-18) to optimality in less average computational cost.

5 Conclusions

In this paper, we proposed a probabilistic constrained variant of the well known unrelated parallel machine scheduling problem. For this purpose, we assumed that each vector of job processing times is an independent and multivariate normally distributed vector with known mean and covariance matrix. This assumption allows transforming the probabilistic constraints into deterministic second order conic constraints [9]. In particular, we considered the problem of makespan minimization when completing a subset of jobs subject to machine energy consumption and job assignment constraints. We computed feasible solutions by solving directly the equivalent deterministic mixed integer second order conic programming problem and also by means of piecewise mixed integer linear programming formulations. Our numerical results showed that one of the piecewise linear formulations allows finding better feasible solutions for instances with up to ten machines and fifty jobs in less average computational cost.

References

1. R. Bixby, Solving Real-World Linear Programs: A Decade and More of Progress, Operations Research, 50(1), 1-13, (2002)
2. CONOPT is a solver for large-scale nonlinear optimization (NLP) developed and maintained by ARKI Consulting & Development A/S in Bagsvaerd, Denmark. `http://www.conopt.com/Algorithm.htm`
3. G. Dantzig, Linear programming and extensions, Princeton University Press, (1963)
4. R. Fortet, Applications de l'algebre de boole en recherche operationelle, Revue Francaise de Recherche Operationelle, 4, 17-26, (1960)
5. B. Geiler, A. Martin, A. Morsi, and L. Schewe, IMA Volume on MINLP, chapter Using piecewise linear functions for solving MINLPs, Springer, (2010)
6. A. Gupte, S. Ahmed, M. Cheon, and S. Dey, Solving mixed integer bilinear problems using MILP formulations, Siam Journal on Optimization, 23(2), 721-744, (2013)
7. IBM ILOG CPLEX Optimization Studio Information Center. Webpage: `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4/index.jsp`
8. A. Keha, I. de Farias, and G. Nemhauser, Models for representing piecewise linear cost functions, Operations Research Letters, 32(1), 44-48, (2004)
9. M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, Linear Algebra and its Applications, 284, pages: 193-228, (1998)
10. H. Markowitz and A. Manne, On the solution of discrete programming-problems, Ecometrica, 25, 84-110, (1957)
11. R. McNaughton, Scheduling with deadlines and loss functions, Management science, 6, pages: 112, (1959)
12. G. Nemhauser and J. Vielma, Modeling disjunctive constraints with a logarithmic number of binary variables and constraints, Lecture Notes in Computer Science, 5035, 199-213, (2008)
13. C. Potts and V. Strusevich, Fifty years of scheduling: a survey of milestones, Journal of the Operational Research Society 60, 41-68, (2009)
14. A. Prékopa, On probabilistic constrained programming, Proceedings of the Princeton Symposium on Mathematical Programming, Princeton University Press, Princeton, 1970.
15. SBB is a new GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. `http://www.gams.com/presentations/present_sbb.pdf`
16. A. Shapiro, D. Dentcheva, and A. Ruszczynski, Lectures on stochastic programming: Modeling and theory, 436. SIAM Philadelphia, Series on Optimization, Vol. 9 of MPS/SIAM, Philadelphia (2009)

MISTA 2015

Exact and heuristic methods for trading-off makespan and stability in stochastic project scheduling

Simon Mountakis · Tomas Klos · Cees Witteveen · Bob Huisman

Abstract This paper addresses a problem of practical value in project scheduling: trading expected makespan for stability, under stochastic activity duration uncertainty. We present the formal statement of a problem that we name Proactive Stochastic RCPSP (PS-RCPSP). Assuming activity durations follow known probability distributions, PS-RCPSP asks to find a so-called earliest-start (es) policy and a proactive schedule that together minimize the weighted sum of expected project makespan and expected instability (deviation of the realized from the proactive schedule). Extending an existing MILP model for the well-known deterministic Resource-Constrained Project Scheduling Problem (RCPSP), we present a MILP model for PS-RCPSP, which allows us to find optimal (es-policy, proactive schedule) pairs. To deal with instances of practical size, we propose a Linear Programming (LP)-based and a Mixed-Integer LP (MILP)-based heuristic. Our LP-based heuristic optimizes the proactive schedule while keeping the es-policy part of the solution fixed. Our MILP-based heuristic aims to optimize the structure of the policy together with the proactive schedule. In contrast to existing state-of-art approaches such as CCP [21] and STC [31], our heuristics rely on optimizing the proactive schedule together with the scheduling policy. Experiments show that the LP-based heuristic is efficient and compares favorably with the state-of-art (i.e. achieves smaller expected makespan for a certain level of expected instability) when the aim is to achieve near-zero instability at the cost of higher makespan. The MILP-based heuristic seems more effective (albeit not as efficient) when the aim is to achieve low expected makespan at the cost of moderate or high instability.

Simon Mountakis
Delft University of Technology
E-mail: k.s.mountakis@tudelft.nl

Tomas Klos
Delft University of Technology
E-mail: t.b.klos@tudelft.nl

Cees Witteveen
Delft University of Technology
E-mail: c.witteveen@tudelft.nl

Bob Huisman
NedTrain
E-mail: b.huisman@nedtrain.nl

1 Introduction

Project scheduling literature mostly concentrates on scheduling subject to temporal and resource constraints. The schedule sought for is an assignment of start-times to activities, facilitating the efficient use of limited resources in order to minimize a lateness measure such as the project makespan. Finding a schedule usually invovles solving the Resource-Constrained Project Scheduling Problem (RCPSP) (see [1, 14] for comprehensive surveys). This problem has been shown to be NP-Hard [7] and finds several industrial applications (e.g. [8, 6]). Associated literature includes numerous exact and (meta-)heuristic algorithms, able to find good schedules for large instances and diverse lateness measures (e.g. [28, 20, 18, 10]). Solving an RCPSP serves the purpose of preparing a feasible schedule, assuming a static deterministic project execution environment. In practice, however, this assumption is rarely valid. Activity durations used for preparing the schedule are mostly rough estimates, since most projects are subject to delays during execution and the final realized schedule is the result of subjecting the original schedule to modifications which make it consistent with the project constraints in the face of delays. Ad-hoc modifications lead to realized activity start-times that might differ from planned start-times, compromising project predictability and timeliness.

This paper addresses the issue of hedging against project uncertainty by preparing a schedule in combination with an execution strategy for coping with delays. In line with other works in *stochastic project scheduling* (see [17] for a comprehensive review) we assume activity durations are given as stochastic variables with known distributions and propose a new *proactive-reactive* scheduling method. First, we define the Proactive Stochastic (PS) RCPSP as an extension of RCPSP. The solution to PS-RCPSP is a *proactive schedule* and an *earliest-start (es-) policy* that together minimize the weighted sum of the realized schedule's expected makespan and its expected deviation from the proactive schedule. PS-RCPSP is, in fact, a generalization of the Stochastic RCPSP (S-RCPSP), which asks to find a *stochastic scheduling policy* instead of a schedule and various *classes* of policies have been proposed in the literature [25, 26, 17]. In general, a policy defines a mapping between activity duration realizations to realized schedules. S-RCPSP asks to find a policy that minimizes the expected project makespan, with only few exact and heuristic approaches (mainly meta-heuristics) proposed over the last decade [29, 4, 5, 3].

Not preparing the project execution based on a schedule that can more or less be trusted (but rather, letting the realized schedule unfold during execution) has been recognized as a shortcoming of S-RCPSP [16]. This shortcoming motivates us and a number of other authors to propose a proactive-reactive approach, with [31, 11, 21] yielding the most promising computational results in existing literature. Different approaches pursue different optimization objectives; however, the common aim is to optimize some *tradeoff* between expected makespan and expected deviation from the proactive schedule. In line with other authors we represent activity duration distributions with a sample and propose a Linear Programming (LP)-based heuristic for PS-RCPSP, a Mixed-Integer LP (MILP) model enabling us to obtain exact solutions, and a MILP-based heuristic which asimilates *iterative flattening* [27]. The (MI)LP models for PS-RCPSP proposed in this paper are the result of adjusting the models proposed by

Artigues et al. in [2] and [23]. We refer to exact solutions assuming stochastic duration distributions can be represented exactly by a sufficiently large sample.[1]

The contributions of this paper extend from section 3 and onwards. In order to base the paper on a consistent and self-contained framework of notation, section 2 summarizes existing concepts from deterministic, reactive, and proactive-reactive project scheduling. Section 3 introduces in a formal manner the problem studied here, that we name Proactive Stochastic RCPSP. Section 4 presents one main contribution of this paper: a LP-based heuristic for solving this problem. Section 5 presents another main contribution: a MILP model for this problem which enables us to obtain exact PS-RCPSP solutions. To our knowledge, no other exact solution methods have been proposed for poblems of similar type. Section 6 presents yet another contribution, a MILP-based heuristic for PS-RCPSP. Section 7 presents an experimental study in which we find that the LP-based heuristic performs favorably in comparison to the state-of-art, especially when the aim is to achieve near-zero instability. We also find the MILP-based heuristic to be more effective (albeit less efficient) when one is willing to accept medium levels of instability in order to minimize expected makespan. Section 8 concludes the paper.

2 Preliminaries

The purpose of this section is to introduce the research area of proactive-reactive (stochastic) project scheduling, which is where the contributions of this paper belong to. To establish autonomy and to facilitate discussion in further sections, we use convenient notation (which sometimes departs from standard notation) and begin with summarizing existing concepts from deterministic and (purely) reactive project scheduling. For a comprehensive survey of deterministic, reactive and proactive-reactive project scheduling, the reader may refer to [17].

2.1 Deterministic project scheduling

A project is usually represented as a directed acyclic graph $G(N, E)$, with nodes $N = \{1, \ldots, n\}$ corresponding to the set of n project activities. Each directed arc (i, j) in $E \subseteq \{(i, j) \in N^2\}$ defines a direct temporal constraint between activities i and j, meaning that j may not start unless activity i has finished. In effect, E defines a binary, irreflexive and transitive relation: if there is a path from activity i to activity j in $G(N, E)$ then j cannot start unless i has finished. Let us $T(E) \supseteq E$ denote the transitive closure of E, defined as

$$T(E) := \{(i, j) \in N^2 : \exists \text{ a path from } i \text{ to } j \text{ in } G(N, E)\}$$

We shall name a *temporally independent set* each subset of activities $X \subseteq N$ which are mutually independent with respect to temporal constraints. That is, if X is a temporally independent set, then $X^2 \cap T(E) = \emptyset$. Obviously, if only temporal constraints are taken into account, the activities of a temporally independent set may overlap in time in a schedule.

[1] This notion of exactness is in line with Stork [29] who represent stochastic duration distributions with a sample when proposing exact search methods for the S-RCPSP.

We assume as input a set $R := \{1, \ldots, m\}$ of m resources which must be shared among activities. Each resource $r \in R$ is associated with known capacity $b_r \in \mathbb{N}_0$. Furthermore, each activity $i \in N$ requires a known amount $q_{ir} \leq b_r$ of resource r while it executes. Vector $\boldsymbol{b} \in \mathbb{N}_0^m$ and matrix $\boldsymbol{q} \in \mathbb{N}_0^{n \times m}$ define the problem's resource constraints. Every independent set X for which $\sum_{i \in X} q_{ir} > b_r$ for some $r \in R$ is called a *forbidden* set. Even though it is allowed by the temporal constraints E, all activities in X may not overlap at some timepoint t because resource r will be used beyond its capacity, which is not possible.

Let $H \subseteq N^2$ denote a set of temporal constraints. Below we give the definition of a function Φ which returns the set of all forbidden sets w.r.t. temporal constraints H and the problem's resource constraints $(\boldsymbol{q}, \boldsymbol{b})$.

$$\Phi(H) := \{X \subseteq N : X^2 \cap T(H) = \emptyset, \sum_{i \in X} q_{ir} > b_r \text{ for some } r \in R\} \tag{1}$$

In addition to the parameters mentioned so far, we assume as input a vector $\boldsymbol{d} \in \mathbb{N}_0^n$ such that d_i defines the duration of activity i. Overall, a tuple $(N, R, E, \boldsymbol{d}, \boldsymbol{q}, \boldsymbol{b})$ specifies an instance of the RCPSP. A schedule $\boldsymbol{s} \in \mathbb{N}_0^n$ such that s_i defines the start time of activity i, is a feasible solution when it satisfies the temporal and resource constraints, meaning that

$$s_j \geq s_i + d_i \qquad \forall (i,j) \in E \tag{2}$$

$$a(\boldsymbol{s}, t) \notin \Phi(E) \qquad \forall t \geq 0 \tag{3}$$

Here, $a(\boldsymbol{s}, t) := \{i \in N : t \in [s_i, s_i + d_i)\}$ is the set of activities executing at timepoint t according to $\boldsymbol{s}$ and Φ as defined earlier. Thus, (3) ensures there is no timepoint t at which the activities of a forbidden set overlap concurrently.

Project source-sink convention. RCPSP asks to find a feasible schedule of minimum makespan $C_{\max}(\boldsymbol{s}) := \max\{s_i + d_i : i \in N\}$. Most RCPSP-related works assume that the last activity, n, is a dummy activity with zero duration (i.e. $d_n = 0$) and that it must wait for the completion of every other activity (i.e. $(i, n) \in T(E)$ for every $i \in N - \{n\}$). This dummy activity is often known as the project "sink" and it holds that $C_{\max}(\boldsymbol{s}) = s_n$. Another convention of most RCPSP-related works is that the first activity, 1, often known as the project "source" must be waited on by every other activity (i.e. $(1, j) \in T(E)$ for every $j \in N - \{1\}$).

We shall hereafter assume activities 1 and n correspond to the project source and sink, respectively. The RCPSP can now be formally stated as:

$$\boldsymbol{s}^* := \arg\min\{s_n : (2), (3), \boldsymbol{s} \geq 0\} \tag{4}$$

2.2 Reactive project scheduling

In the research area of stochastic project scheduling, the activity durations vector $\boldsymbol{d}$ is replaced with a stochastic vector $\boldsymbol{D}$ such that D_i is the stochastic variable representing the uncertain duration of activity i, with a known probability distribution $\mathbb{P}[D_i = t]$. In line with recent works on S-RCPSP, we shall denote (elements of) stochastic vectors with a capital symbol.

S-RCPSP is a purely reactive extension of RCPSP. The solution sought for is no longer a schedule, but a reactive scheduling policy. A policy is a combinatorial object π which parameterizes the mapping from stochatic vector $\boldsymbol{D}$ to a corresponding realized schedule $\boldsymbol{S}(\pi, \boldsymbol{D})$. Note that $\boldsymbol{S}$ denotes a function which returns a vector of activity start times (of length n). Furthermore, if a realization $\boldsymbol{d}$ of $\boldsymbol{D}$ is passed as an argument, then $\boldsymbol{S}(\pi, \boldsymbol{d})$ denotes a deterministic vector. if $\boldsymbol{D}$ is passed as an argument, $\boldsymbol{S}(\pi, \boldsymbol{D})$ denotes a stochatic vector.

Different classes of policies have been proposed in the literature [25,26,29,3]. One condition that all policy classes are expected to satisfy is that function $\boldsymbol{S}$ complies with the *non-anticipativity constraint*: the decision to start activity i at time $[\boldsymbol{S}(\pi, \boldsymbol{D})]_i$ cannot rely on information from the feature: the value of $[\boldsymbol{S}(\pi, \boldsymbol{D})]_i$ must be determined by time $t \leq [\boldsymbol{S}(\pi, \boldsymbol{D})]_i$. Other features such as the structure of π and the definition of function $\boldsymbol{S}$ depend on the class under study.

List-based policies. Two classes of policies which are prominent in the literature are *resource-based* (rb) policies and *activity-based* (ab) policies, also known collectively as *list-based policies*. A list-based policy is a priority vector $\boldsymbol{l} \in \mathbb{R}^n$ assigning priority l_i to activity i. Realized schedule $\boldsymbol{S}(\boldsymbol{l}, \boldsymbol{D})$ is computed by a variant of the well-known parallel schedule-generation-scheme (SGS) complying with the non-anticipativity constraint [5]; with the SGS definition being slightly different between rb-policies and ab-policies. As far as list-based policies are concerned, S-RCPSP asks to find a vector $\boldsymbol{l} \in \mathbb{R}^n$ that minimizes $\mathbb{E}[[\boldsymbol{S}(\boldsymbol{l}, \boldsymbol{D})]_n]$. Stork [29] proposes exact branch-and-bound algorithms for both rb and ab-policies. Ballestín [4] proposes an efficient genetic algorithm for ab-policies, providing the first computational experience on larger S-RCPSP instances. Ballestín and Leus [5] manage to obtain better results with a Greedy Randomized Adaptive Search Procedure (GRASP), again for the class of ab-policies. The best performance (w.r.t. expected makespan minimization) has so far been obtained with the more recent work of Ashtiani et al. [3] who propose a GRASP for a new class, namely *pre-processing* (pp) policies–a hybrid between rb-policies and es-policies.

Earliest-start policies. An es-policy is a set of temporal constraints $\mathcal{E} \subseteq N^2$ chosen such that

$$T(\mathcal{E}) \supseteq E, \tag{5}$$

$$\Phi(\mathcal{E}) = \emptyset, \tag{6}$$

$$G(N, \mathcal{E}) \text{ is acyclic} \tag{7}$$

Condition (5) ensures that a schedule $\boldsymbol{s}$ satisfying $s_j \geq s_i + d_i$ for each $(i, j) \in \mathcal{E}$ (here $\boldsymbol{d}$ can be any arbitrary choice of activity durations) is feasible with respect to the problem's precedence constraints E. Condition (6) ensures that $\boldsymbol{s}$ satisfying $\mathcal{E}$ implies that it also satisfies resource constraints prescribed by availabilities $\boldsymbol{b}$ and requirements $\boldsymbol{q}$ (as described earlier). Condition (7) ensures that the set of schedules satisfying $\mathcal{E}$ (for any arbitrary choice of activity durations $\boldsymbol{d}$) is non-empty.

When a project is executed according to an es-policy $\mathcal{E}$, the schedule that is realized, $\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})$, is what is often known as the *earliest-start* schedule specified by $\mathcal{E}$. The earliest-start schedule of $\mathcal{E}$ can be defined as:

$$[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_j := \max\{[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_i + D_i : (i, j) \in \mathcal{E}\} \tag{8}$$

To put it simply, an activity j starts immediatelly when all its predecessors in $G(N, \mathcal{E})$ have finished. This time quantity (the latest finish time of j's predecessors) is often known as the length of the *critical path* from project source 1 to activity j. As far as es-policies are concerned, the S-RCPSP asks to find some $\mathcal{E}^*$ which minimizes $[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_n$ (the length of the critical path to the project sink) by expectation:

$$\mathcal{E}^* := \arg\min\{\mathbb{E}[[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_n] : (5-7), \mathcal{E} \in N^2\} \tag{9}$$

Constraints $(5-7)$ enforce the feasibility of any realized schedule with both the precedence and the resource constraints of the problem.

Stork [29] proposed an exact branch-and-bound search for problem (9). His algorithm considers each *minimal* forbidden set X (subset-minimal forbidden set) in some order and branches on each of $|\{(i,j) \in X^2\}|$ arcs which can be included in $\mathcal{E}$ in order to eliminate X from $\Phi(\mathcal{E})$. Without obtaining new computational results, in [22] Leus gives a formal treatment of es-policies as resource-flow networks (flow networks which can represent feasible RCPSP schedules) and proposes a refined version of the branch & bound algorithm of Stork. Exploiting the relation between resource-flows and es-policies, Artigues et al. [23] propose a robust optimization model for es-policies, for when a stochastic characterization of uncertainty is not available.

2.3 Proactive-reactive project scheduling

Reactive project scheduling allows one to pick activity start times dynamically during the project, under conditions of uncertainty. A main drawback of this approach (e.g. [16, 17, 9]) is that prior to (and during) project execution there is no schedule prescribing activity start times that can more or less be trusted. Such a "proactive" schedule can serve important organizational purposes; in fact, the deviation of the realized schedule from this proactive schedule is expected to induce organizational costs.

Attempts to overcome this drawback gave rise to the research area of proactive-reactive project scheduling, which is the research area that this paper belongs to. The main idea behind the proactive-reactive approach is to execute the project by using a proactive schedule together with a scheduling policy. Under uncertainty, some activities may not start at their proactive start times, because activities they have to wait for are not yet finished and/or resources they require are not yet released. In such cases, the scheduling policy determines which activities to start at their prescribed start times and which to postpone. It should be noted that most works assume *railway-mode scheduling*, meaning that an activity may not start earlier than its proactive start time, which strengthens the "stability" of the project execution. Clearly, the realized schedule is a function of the policy and the proactive schedule. Achieving low instability (deviation of the realized from the proactive schedule) requires "spreading" proactive activity start times far appart, in effect increasing the expected makespan. The general aim is to optimize some tradeoff between expected makespan and expected instability.

Van de Vonder et al. [32, 31, 30] propose and evaluate experimentally various proactive-reactive heuristics. The proposed heuristics assume as input an instance of S-RCPSP along with an initial schedule. The best performing heuristic is the so-called Starting Time Criticality (STC) heuristic. An es-policy is extracted from the structure of this initial schedule and used to iteratively transform the initial schedule into a proactive schedule by inserting time-buffers betwen activities. Deblaere et al. [11] propose an approach which integrates the proactive step (forming a proactive schedule)

and the reactive step (forming the adjoining policy). Their approach is only possible to compare with ours and others that assume railway-mode scheduling, by choosing sufficiently high penalties for earliness (w.r.t. the proactive schedule).[2] More recently, Vilches and Demeulemeester [21] propose a Chance-Constrained Programming model (CCP) for the RCPSP which asks to find a minimum makespan schedule subject to probabilistic temporal and resource constraints. They propose a Mixed Integer LP model, the solution to which is a proactive schedule that will most likely remain feasible under stochastic duration variability, without presumption on the policy that will be used during project execution.

3 Proactive Stochastic RCPSP

In deterministic and reactive project scheduling, the main problem under study (RCPSP and S-RCPSP, respectively) is stated clearly. A clearly stated problem model cannot be found in proactive-reactive project scheduling literature, perhaps because this research area is still in a burn-in phase.[3]Existing literature seems to pursue the general aim of optimizing some tradeoff between expected makespan and expected instability (deviation from the proactive schedule). This section presents the formal statement of a proactive-reactive scheduling problem for which (heuristic and exact) solution methods are proposed in subsequent sections.

The problem presented here, the Proactive Stochastic RCPSP (PS-RCPSP), asks to find a tuple $(\mathcal{E}, \boldsymbol{t})$ where $\mathcal{E}$ is an es-policy and $\boldsymbol{t}$ is a proactive schedule, minimizing the weighted sum of two performance criteria:

1. expected value of project makespan,
2. expected value of tardiness with respect to proactive release-times.

The first criterion measures lack of robustness and is relevant for obvious reasons. The second criterion measures instability and captures the expected deviation of the realized schedule from the proactive schedule. Note that $\boldsymbol{t}$ prescribes activity release-times (an activity i may not start earlier than t_i). Intuitively, instability represents the extent to which the proactive start-times can be trusted, when used for organizational purposes before and during project execution.

An instance of this problem is encoded by a tuple $(n, m, \boldsymbol{q}, \boldsymbol{b}, E, \boldsymbol{D}, \alpha)$. For clarity, we summarize the meaning of problem parameters. Positive integer n is the number of activities and m is the number of resources. Parameters $\boldsymbol{q} \in \mathbb{N}_0^{m \times n}$ and $\boldsymbol{b} \in \mathbb{N}_0^m$ define resource requirements and availabilities respectively. Set $E \subseteq \{1, \ldots, n\}^2$ defines pairwise precedence constraints. Stochastic vector $\boldsymbol{D}$ is of length n with each element D_i a stochatic variable (with given distribution $\mathbb{P}[D_i = t]$) which describes the uncertain duration of activity i. Finally, parameter $\alpha \in [0, 1]$ determines the desired tradeoff between robustness (i.e. minimization of expected makespan) and stability (i.e. minimization of expected instability). More emphasis can be put on either minimizining makespan (by choosing α closer to one) or minimizing instability (by choosing α closer to zero).

[2] We are grateful to one of our anonymous reviewers for this remark.

[3] Some works refer to [16] but this is a formal treatment of a proactive-reactive *machine* scheduling problem.

Formally, the problem can be stated as follows:

$$\min \quad f(\mathcal{E},\boldsymbol{t}) := \alpha \cdot \mathbb{E}\left[[\boldsymbol{S}((\mathcal{E},\boldsymbol{t}),\boldsymbol{D})]_n\right] + (1-\alpha)\cdot \mathbb{E}\left[\sum_{i=1}^{n}([\boldsymbol{S}((\mathcal{E},\boldsymbol{t}),\boldsymbol{D})]_i - t_i)\right] \quad (10)$$

$$\text{s.t.} \quad \Phi(G(N,\mathcal{E})) = \emptyset \quad (11)$$

$$T(\mathcal{E}) \supseteq E \quad (12)$$

$$G(N,\mathcal{E}) \text{ is acyclic} \quad (13)$$

$$\mathcal{E} \in \{1,\ldots,n\}^2, \boldsymbol{t} \geq 0 \quad (14)$$

Conditions (12,13) ensure there is a non-empty set of schedules satisfying the problem's precedence constraints as prescribed in E. Condition (11) ensures that each such schedule also satisfies the problem's resource constraints prescribed by $\boldsymbol{q}$ and $\boldsymbol{b}$.

4 Heuristic LP-based approach

This section presents a polynomial-time heuristic for PS-RCPSP which consists of two steps:

1. using mean activity durations, the deterministic RCPSP $(n, m, \boldsymbol{q}, \boldsymbol{b}, E, \mathbb{E}[\boldsymbol{D}])$ is solved to obtain a good schedule $\boldsymbol{s}$ and a feasible es-policy $\mathcal{E}$ (i.e. satisfying (11),(12) and (13)) is derived from the structure of $\boldsymbol{s}$ in polynomial time (this procedure is described in [2]),
2. by solving a linear program presented below, we find a proactive schedule $\boldsymbol{t}$ that is optimally combined with $\mathcal{E}$ (which is kept fixed) so as to minimize an approximation of (10).

After $\mathcal{E}$ has been obtained in the first step, finding $\boldsymbol{t}$ which minimizes (an approximation of) the PS-RCPSP objective is achieved by solving the LP model presented below.

$$\min \quad \left[\alpha\left(\frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma} s_n^\gamma\right) + (1-\alpha)\left(\frac{1}{|\Gamma|}\sum_{i=1}^{n}\sum_{\gamma\in\Gamma}(s_i^\gamma - t_i)\right)\right] \quad (15)$$

$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma \qquad \forall (i,j)\in\mathcal{E}, \gamma\in\Gamma \quad (16)$$

$$s_i^\gamma \geq t_i \qquad \forall i = 1,\ldots,n \quad (17)$$

$$\boldsymbol{t} \geq 0 \quad (18)$$

Here, (15) approximates the objective (10) based on $\Gamma \subseteq \mathbb{R}^n$: an adequately-sized sample of stochastic vector $\boldsymbol{D}$. The realization of activity durations under sample scenario $\gamma \in \Gamma$ is represented by vector $\boldsymbol{d}^\gamma = (d_1^\gamma,\ldots,d_n^\gamma)$. The corresponding realized schedule is *earliest-start*$((\mathcal{E},\boldsymbol{t}),\boldsymbol{d}^\gamma) = (s_1^\gamma,\ldots,s_n^\gamma)$, as computed by the model constraints. The solution is a proactive schedule $\boldsymbol{t} = (t_1,\ldots,t_n)$ that optimizes the tradeoff between expected makespan and instability for the given es-policy $\mathcal{E}$. This LP model has $n(|\Gamma|+1)$ linear variables (n variables t_i and $n|\Gamma|$ variables s_i^γ).

4.1 Related work

Van de Vonder et al. [31] propose several heuristics, of which the most competitive is the Starting Time Criticality (STC) heuristic and we shall therefore restrict our attention to it. Our LP-based heuristic bears similarities with STC. In fact, the first step of our heuristic is identical to that of STC: an es-policy $\mathcal{E}$ is extracted by the structure of an initial schedule $\boldsymbol{s}$. The second step of STC involves transforming the "unstable" schedule $\boldsymbol{s}$ into a "stable" schedule $\boldsymbol{t}$ with an iterative procedure, while keeping $\mathcal{E}$ fixed. In each iteration a one-unit time buffer is added at the start of that activity that "needs it the most" (as determined by a proposed "starting time criticality" measure) until adding more buffer time would not further reduce the instability of $\boldsymbol{t}$, which is measured by

$$\mathbb{E}\left[\sum_{i=1}^{n} w_i([\boldsymbol{S}((\mathcal{E},\boldsymbol{t}),\boldsymbol{D})]_i - t_i)\right] \tag{19}$$

Here, w_i is a cost associated with the instability of activity i. Furthermore, t_n is kept fixed to a project deadline and therefore w_n represents the marginal cost of deviating from this project deadline. Note that by replacing α in (10) with individual weights w_i and choosing a fixed project deadline, it is straightforward to adapt our approach to the instability objective considered by van de Vonder et al. However, we felt that the choice of (10) as an objective is advantageous, as it underlines the tradeoff between expected makespan and instability more clearly and simplifies discussion by not involving a weight per individual activity and not requiring the choice of a project deadline.

Note that $\boldsymbol{t}$ is not guaranteed to be (precedence and resource) feasible with respect to mean activity durations (as required in the work of van de Vonder [31]). Enforcing $\boldsymbol{t}$ to hold this property in our approach can be accomplished by including the following constraint in the LP model:

$$t_j \geq t_i + \mathbb{E}[D_i] \qquad \forall (i,j) \in \mathcal{E}$$

However, this property only adds to the organizational value of $\boldsymbol{t}$ when mean values are reasonable estimates of activity durations.

Let us note that both our heuristic and STC have polynomial worst-case complexity (in the number of activities). However, in contrast with STC, our approach guarantees that $\boldsymbol{t}$ is chosen optimally when $\mathcal{E}$ is kept fixed and assuming the distribution of $\boldsymbol{D}$ is approximated with a sample. Therefore, if efficiency considerations enable us to choose a large-enough sample Γ (which is mostly the case due to the efficiency of existing LP solvers), our heuristic is expected to perform at least as well as STC. Finally, note that our heuristic is simpler to implement, requiring only the description of the presented LP model.

Leus et al. [24] assume as input a proactive schedule $\boldsymbol{t}$ (e.g. one that has been produced by STC). They propose a branch-and-bound search which returns the es-policy $\mathcal{E}$ which fits $\boldsymbol{t}$ optimally in minimizing an expression of expected instability similar to (19).

5 Exact MILP-based approach

PS-RCPSP (section 3) asks to find an es-policy and a proactive schedule $(\mathcal{E},\boldsymbol{t})$ that together minimize the weighted sum of expected makespan and instability. Section 4

presented a heuristic approach according to which $\mathcal{E}$ is kept fixed while $\boldsymbol{t}$ is optimally paired with the policy by solving a LP. This section presents a Mixed Integer LP (MILP) model with which PS-RCPSP can be solved to optimality. However, it should be pointed-out that a solution is trully exact only if we assume stochastic duration distributions can be accurately described by the chosen sample Γ; for general probability distributions we obtain a lower bound. In fact, the problem of computing the exact expected makespan of a given es-policy (and assuming duration distributions with discrete support) has been shown by Hagstrom in [13] to be intractable. However, our notion of exactness is in line with the computational study of Stork [29] where "optimal" scheduling policies are computed by using a fixed sample of duration distributions.

This model includes binary variables representing the structure of $\mathcal{E}$ and linear variables representing $\boldsymbol{t}$. To our knowledge, no other exact approaches have been proposed in the literature for problems of similar type (i.e. asking for a scheduling policy and proactive schedule that together optimize some tradeoff between expected makespan and instability). To arrive at this PS-RCPSP model, we merge the LP model presented in the previous section with a MILP model that has been proposed by Artigues et al. [2] and which allows to solve the deterministic RCPSP by treating it as a flow-network problem. The model presented here is not entirely new, since a similar technique (repeating precedence constraints for each scenario of the chosen sample) has been proposed in [23] for minimizing the maximum regret of an es-policy.

5.1 The RCPSP model of Artigues et al.

Artigues et al. [2] represent a solution to the RCPSP as a so-called *resource-flow* $\boldsymbol{f} \in \mathbb{R}_0^{n\times n\times m}$; an assignment to variables f_{ijr} associated with each pair of activities $(i,j) \in N^2$ and each resource $r \in R$. A resource-flow describes the "passing" of resource units inbetween activities. More precisely, $\boldsymbol{f}$ is an indirect representation of every schedule $\boldsymbol{s}$ in which f_{ijr} units of resource r are released by activity i at its completion $s_i + d_i$ and then "picked up" by activity j at its start s_j, without another activity using these units between $s_i + d_i$ and s_j.

A resource-flow is feasible when it satisfies

$$\sum_{j\in N-\{i\}} f_{jir} = q_{ir} \quad \forall i \in N - \{1\} \tag{20}$$

$$\sum_{j\in N-\{i\}} f_{ijr} = q_{ir} \quad \forall i \in N - \{n\} \tag{21}$$

Eq. (20) asks that each activity i (except for the sink) receives as many resource units as it requires the moment it starts. Eq. (21) asks that each activity i (except for the source) releases as many resource units as it has used the moment it finishes.

The flow network $G(N, \phi(\boldsymbol{f}))$ associated with $\boldsymbol{f}$ is defined as $\phi(\boldsymbol{f}) := \{(i,j) \in N^2 : f_{ijr} > 0 \text{ for some } r \in R\}$; i.e. there is an arc from each activity to every other activity it passes at least one resource unit to. As shown by Leus [22,23], feasible resource-flows and es-policies are interrelated: $\mathcal{E} = E \cup \phi(\boldsymbol{f})$ is a feasible es-policy if $\boldsymbol{f}$ is a feasible resource-flow (and $G(N,\mathcal{E})$ is acyclic). Therefore, every schedule which satisfies $G(N,\mathcal{E})$ is feasible. The following MILP model proposed by Artigues et al. enables one to find a feasible resource-flow $\boldsymbol{f}$ which minimizes the cost (described by function g) of a schedule $\boldsymbol{s}$ which satisfies the temporal constraints of $G(N, E \cup \phi(\boldsymbol{f}))$.

$$\min \quad s_n \tag{22}$$
$$\text{s.t.} \quad s_j \geq s_i + d_i - M(1 - z_{ij}) \qquad \forall (i,j) \in N \tag{23}$$
$$z_{ij} = 1 \qquad \forall (i,j) \in E \tag{24}$$
$$f_{ijr} \leq M z_{ij} \qquad \forall (i,j) \in N^2, r \in R \tag{25}$$
$$(20), (21) \tag{26}$$
$$f_{ijr} \geq 0, z_{ij} \in \{0,1\} \qquad \forall (i,j) \in N^2, r \in R \tag{27}$$

Here M is a large constant. Due to (26) $\boldsymbol{f}$ is a feasible resource-flow. Due to (25), if $f_{ijr} > 0$ for one or more $r \in R$ then $z_{ij} = 1$, meaning that variables z_{ij} describe the flow-network $\phi(\boldsymbol{f})$ of the resource-flow (i.e. $\phi(\boldsymbol{f}) = \{(i,j) \in N^2 : z_{ij} = 1\}$). Due to (23) and (24), $\boldsymbol{s}$ describes a schedule which satisfies the temporal constraints in $G(N, E \cup \phi(\boldsymbol{f}))$. Since $\boldsymbol{f}$ is a feasible resource-flow, $\boldsymbol{s}$ is a feasible schedule.

5.2 Extension for S-RCPSP

This section presents a trivial extension to the RCPSP model of Artigues et al. which enables us to find optimal es-policies for the S-RCPSP. Considering a sample $\Gamma \subset \mathbb{R}^n$ of stochastic activity durations vector $\boldsymbol{D}$ allows us to present the following MILP model.

$$\min \quad \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} s_n^\gamma \tag{28}$$
$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma - M(1 - z_{ij}) \qquad \forall (i,j) \in N^2, \gamma \in \Gamma \tag{29}$$
$$(24 - 27) \tag{30}$$

Our extension is rather straightfoward. Each variable s_i is included here as variable s_i^γ for each sample scenario $\gamma \in \Gamma$. Precedence constraints (23) from before are now replicated for each scenario $\gamma \in \Gamma$ in condition (29). Objective (22) is now replaced with objective (28), which estimates the makespan expectation $\mathbb{E}[S(E \cup \phi(\boldsymbol{f}))]$ based on sample Γ.

5.3 Extension for PS-RCPSP

Here we extend the previous model by including a variable t_i for each $i \in N$, which determines the activity's proactive starting time. The resulting PS-RCPSP MILP model is presented below.

$$\min \quad \left[\alpha \left(\frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} s_n^\gamma \right) + (1 - \alpha) \left(\frac{1}{|\Gamma|} \sum_{i=1}^{n} \sum_{\gamma \in \Gamma} (s_i^\gamma - t_i) \right) \right] \tag{31}$$
$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma - M(1 - z_{ij}) \qquad \forall (i,j) \in N^2, \gamma \in \Gamma \tag{32}$$
$$(24 - 27) \tag{33}$$
$$s_i^\gamma \geq t_i \qquad i \in N, \gamma \in \Gamma \tag{34}$$
$$\boldsymbol{t} \geq 0 \tag{35}$$

The objective now becomes identical to that of the LP-based heuristic, measuring the weighted sum of expected makespan and expected instability. Condition (34) ensures that an activity may not start earlier than its proactive start time.

To summarize, by solving this model we obtain a PS-RCPSP solution $(\mathcal{E}, \boldsymbol{t})$ where $\mathcal{E} = \{(i,j) \in N^2 : z_{ij} = 1\}$ is a feasible es-policy and $\boldsymbol{t}$ defines a proactive schedule.

Proposition 1 *Define $\mathcal{E} := \{(i,j) : z_{ij} = 1\}$ the arcs of the flow-network $G(N, E \cup \phi(\boldsymbol{f}))$ associated with resource-flow $\boldsymbol{f}$. Let $\boldsymbol{f}, \boldsymbol{z}, \boldsymbol{s}, \boldsymbol{t}$ be an optimal solution. For each scenario $\gamma \in \Gamma$, $\boldsymbol{s}^\gamma$ defines a schedule where each activity i starts as soon as allowed by its proactive release-time t_i and es-policy $\mathcal{E}$.*

Proof For each scenario $\gamma \in \Gamma$, let $\bar{x}^\gamma$ denote the earliest allowed start time for activity i, allowed by the combination of $\mathcal{E}$ and proactive schedule $\boldsymbol{t}$. We want to prove that in an optimal solution, $\boldsymbol{s}^\gamma = \bar{x}^\gamma$ for all $\gamma \in \Gamma$.

Assume that $s_i^\gamma = \bar{x}_i^\gamma + \delta$ for some $\gamma \in \Gamma, i \in N$, with $\delta > 0$. Since (31) increases monotonically with s_i^γ, the objective can be improved by setting $s_i^\gamma = \bar{x}_i^\gamma$, without violating any constraints. Therefore, in every optimal solution we have $s_i^\gamma = \bar{x}_i^\gamma$ for all $\gamma \in \Gamma$ and $i \in N$, meaning that each $\boldsymbol{s}^\gamma$ defines the earliest start times schedule allowed by the combination of by es-policy $\mathcal{E}$ and proactive schedule $\boldsymbol{t}$ under scenario γ. □

By proposition 1 it follows that an optimal solution to the MILP model presented above is, in fact, an optimal solution for the PS-RCPSP.

6 Heuristic MILP-based approach

Even for small instances (e.g. with 30 activities and 4 resources), solving the proposed model might take an inordinate amount of time. We propose an algorithm (Algorithm 1), the main idea of which was inspired by the *iterative flattening* heuristic of Oddi et al. [27]. The heuristic of Oddi et al. was developed for the deterministic RCPSP with minimum/maximum time-lag precedence constraints [15]. Every feasible schedule for the problem they study is compactly represented as a network of temporal constraints (known as a Simple Temporal Network [12]). It is the similarity with an es-policy (which is a network of zero-lag temporal constraints) that has inspired the development of the heuristic presented here.

The proposed heuristic involves solving a sequence of sufficiently small subproblems with non-increasing optimal objective values. Each iteration involves solving a partially solved instance to optimality. Thus, worst-case complexity is exponential in the number of activities. In practice, however, "good" solutions can be obtained with relative efficiency.

Algorithm 1 assumes as input an instance of the PS-RCPSP. According to aforementioned notation, the instance is represented as $(N, R, E, \boldsymbol{D}, \boldsymbol{q}, \boldsymbol{b}, \alpha)$. An initial solution is obtained by solving a deterministic RCPSP (lines 1-3) which can be done efficiently with one of the various existing heuristics. This solution will serve as a starting point for the first iteration, which is described as follows. A partial solution is formed by removing a random subset of highly critical arcs from the current solution (line 6). The resulting subproblem is solved to optimality (by use of the proposed model) and a complete solution is obtained (line 7). If this new solution is better, it becomes the starting point of the next iteration. The algorithm may terminate when, e.g., a chosen number of iterations have been performed, or the objective has failed to improve a certain number of times.

Algorithm 1 Iterative flattening for PS-RCPSP

1: $\boldsymbol{s} \leftarrow$ schedule for RCPSP $(N, R, E, \mathbb{E}[\boldsymbol{D}], \boldsymbol{q}, \boldsymbol{b})$
2: $\mathcal{E}^* \leftarrow E \cup \phi(\boldsymbol{f^s})$ with $\boldsymbol{f^s}$ extracted from $\boldsymbol{s}$
3: $\boldsymbol{t}^* \leftarrow (0, \ldots, 0)$
4: **while** termination criteria not met **do**
5: $\quad\mathcal{H} \leftarrow$ random subset of $\mathcal{E}^* - T(E)$ chosen by criticality probability
6: $\quad(\mathcal{E}, \boldsymbol{t}) \leftarrow$ optimal solution for PS-RCPSP $(N, R, \mathcal{E}^* - \mathcal{H}, \boldsymbol{D}, \boldsymbol{q}, \boldsymbol{b}, \alpha)$
7: $\quad$**if** $(\mathcal{E}, \boldsymbol{t})$ has a lower objective than $(\mathcal{E}^*, \boldsymbol{t}^*)$ **then**
8: $\quad\quad(\mathcal{E}^*, \boldsymbol{t}^*) \leftarrow (\mathcal{E}, \boldsymbol{t})$
9: $\quad$**end if**
10: **end while**
11: return $(\mathcal{E}^*, \boldsymbol{t}^*)$

Further efficiency improvements. Note that the optimal solution $(\mathcal{E}, \boldsymbol{t})$ of the subproblem solved in each iteration cannot be worse than the best solution seen so far, $(\mathcal{E}^*, \boldsymbol{t}^*)$. To improve performance one may use $(\mathcal{E}^*, \boldsymbol{t}^*)$ as an initial solution when solving the model (line 6). Efficiency can be further improved by reducing the number of binary variables z_{ij} in the model. This can be accomplished by observing that z_{ij} for each $(i, j) \in T(\mathcal{E}^* - \mathcal{H})$ can be fixed to one and z_{ij} for each $(j, i) \in T(\mathcal{E}^* - \mathcal{H})$ can be fixed to zero.

7 Experiments

In [21], Vilches and Demeulemeester compare their method (CCP) with that by Van de Vonder et al. (STC) [31]. To the best of our knowledge, STC and CCP constitute the state-of-art as far as trading expected makespan for instability in stochastic project scheduling is concerned. In this section we extend this comparison by using the same experimental set-up and including results for our LP-based heuristic (Section 4) and the MILP-based heuristic (Section 6). As the results show, our approaches compare favorably with STC and CCP.

The set-up used in [21] was based on the `J30` deterministic RCPSP bench-set of the well-known PSPLIB [19], which comprises 480 deterministic RCPSP instances, each with $n = 30$ activities. Based on this bench-set, three stochastic RCPSP bench-sets were derived, namely `J30-low`, `J30-med`, and `J30-high`, corresponding to conditions of low, medium, and high project uncertainty, with activity durations following a discretized beta distribution. Specifically, each activity i with duration d_i in the deterministic RCPCP instance now has stochastic duration $D_i = [X_i d_i 0.5(l + h)]$ with $\mathbb{E}[D_i] \simeq d_i$ where

- X_i follows a beta distribution with shape parameters $\alpha = 2, \beta = 5$;
- $l = 0.75$ and $h = 1.625$ in the low variability bench-set;
- $l = 0.5$ and $h = 2.25$ in the medium variability bench-set;
- $l = 0.25$ and $h = 2.875$ in the high variability bench-set;
- operator $[\cdot]$ represents rounding to the closest integer.

Each of the evaluated methods (including STC and CCP) has a certain "tradeoff parameter" which determines whether more emphasis is put on minimizing expected makespan or minimizing expected instability. For our LP-based and MILP-based heuristic this tradeoff parameter is the weight α in expression (10). CCP and STC have corresponding parameters with a similar effect. By varying the choice of the

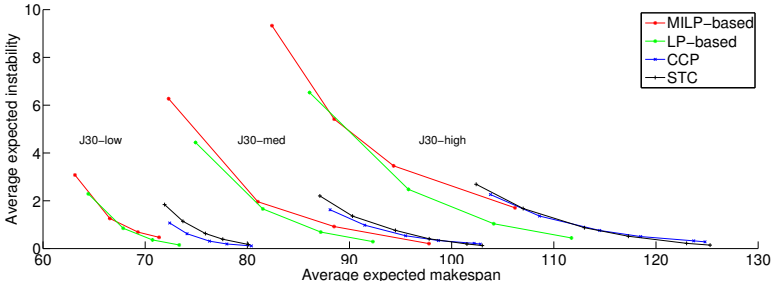


Fig. 1: Trading expected makespan for stability.

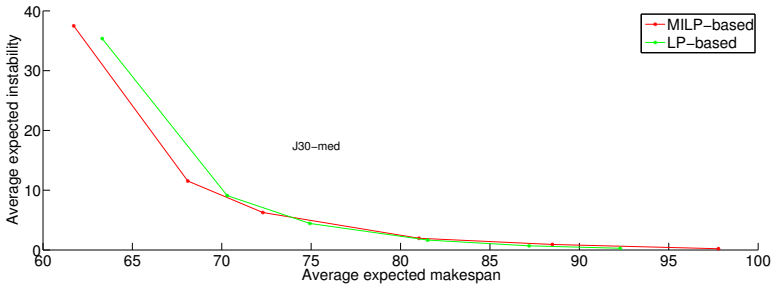


Fig. 2: Trading expected makespan for stability for higher α.

corresponding tradeoff parameter(s), a set of tradeoff data-points is obtained for each method, on each of the three bench-sets.

In Figure 1, the data-points for each method are displayed as a tradeoff curve, on each of the three bench-sets, resulting in three "clusters" of tradeoff curves. A tradeoff curve captures the average performance of the method on that bench-set. Specifically, each data-point is two-dimensional and records the average expected makespan and average expected instability for a certain choice of the tradeoff parameter(s), where the average is taken over all 480 instances of the bench-set. Data-points for CCP and STC are borrowed from the work of Vilches and Demeulemeester [21]. Data-points for our heuristics are obtained by setting $\alpha = 0.05$, 0.1, 0.2, and 0.4. Higher alpha values correspond to data-points closer to the upper left corner, with higher instability and lower makespan.

Figure 2 focuses on the medium variability case for higher α values, including additional data-points for $\alpha = 0.6$ and $\alpha = 0.9$. This allows us to compare the MILP-based and LP-based heuristics when more emphasis is put on minimizing expected makespan.

The expected makespan and expected instability of the solution provided by each of the methods on a particular instance is computed with a sample $\mathit{\Gamma}^{large} \subset \mathbb{R}^n$ comprising $|\mathit{\Gamma}^{large}| = 10^3$ realization of durations vector $\boldsymbol{D}$. Note that the data-points of Vilches and Demeulemeester were computed with a different sample of size 10^3. We assume that 10^3 is a sufficiently large sample size to facilitate comparability with our results.

Configuration of heuristics. The sample Γ^{milp} used by our MILP-based heuristic during optimization (see line 6 of Algorithm 1) is of size $|\Gamma^{milp}| = 30$. Our MILP-based heuristic is configured to perform three (3) iterations and the number of highly critical arcs removed in each iteration (see line 5 of Algorithm 1) is $|\mathcal{H}| = 20$. Note that the criticality of the arcs is computed based on sample Γ^{large} (this is done efficiently, in time quadratic in n and linear in $|\Gamma^{large}|$). The solver we use is CPLEX version 12.6. Furthermore, we set a time-limit for the solver to 50 seconds (since each iteration starts from a feasible solution, the solver will always return with a solution within the time-limit). The polynomial-time complexity of our LP-based heuristic (no binary variables in the model) allows us to use a large sample during optimization. In fact, we use sample Γ^{large}. To find a deterministic schedule (as required in step 1 of this LP-based heuristic) we used a priority rule procedure recently proposed in [10]. Vilches and Demeulemeester use a sample of size 10^2 during optimization, for both STC and CCP. Furthermore, they limit the time spent in solving their CCP model on an instance to a maximum of 10 seconds.

Observations. Figure 1 suggests that regardless of the mode of variability (low, medium, or high), when the purpose is to achieve near-zero instability, the LP-based heuristic yields the best results. This can be attributed to the efficiency of solving a LP model, which enables us to use a large sample (of size 10^3 in this case) during optimization.

Figure 2 suggests that even though the sample used during optimization is much smaller for the MILP-based heuristic (of size 30), it is more effective than the LP-based heuristic for higher α values (i.e. when minimizing instability is more important than minimizing makespan). Both the LP-based and the MILP-based heuristics start from the same es-policy (see step 1 in section 4 and line 2 of Algorithm 1, respectively). However, the MILP-based heuristic restructures the policy and this enables it to perform better at minimizing expected makespan.

Restructuring the policy comes at the cost of efficiency. With three iterations allowed per instance, this yields an average of 50 seconds per instance for the MILP-based heuristic. The LP-based heuristic is considerably more efficient, with an average of 1.5 seconds per instance. Vilches and Demeulemeester report that STC spends on average 0.2 seconds per instance, while their CCP approach spends on average 10 seconds per instance.

8 Conclusions and future work

This paper proposes the PS-RCPSP problem model which, assuming stochastic activity durations, asks to find a so-called earliest-start (es) policy and a proactive schedule that together minimize the weighted sum of expected project makespan and expected instability. Extending an existing MILP model for the RCPSP, a MILP model for PS-RCPSP is presented, which allows us to find optimal (es-policy, proactive schedule) pairs. Solving this problem to optimality might require an impractical amount of time, even for instances with few activities (e.g. 30). Therefore, we propose a LP-based and a MILP-based heuristic for the PS-RCPSP. Our LP-based heuristic optimizes the proactive schedule by keeping the es-policy part of the solution fixed. Our MILP-based heuristic optimizes the structure of the policy together with the proactive schedule. The LP-based heuristic, which is rather efficient, seems to be more effective compared to the state-of-art (i.e. achieves smaller expected makespan for a certain level of expected

instability) especially when the aim is to achieve close to zero instability. The MILP-based heuristic is rather effective when the aim is to achieve low expected makespan at the cost of moderate or high instability. In contrast to existing state-of-art approaches such as CCP [21] and STC [31], our heuristics rely on the idea of optimizing the proactive schedule together with the scheduling policy. This difference might in part explain observed performance differences.

Future work involves a thorough experimental analysis of the proposed heuristics, not for the purpose of comparing them to the state-of-art, but for a deeper understanding of their behavior and its dependence on problem characteristics. Furthermore, most existing stochastic project scheduling works are evaluated on instances where the deterministic RCPSP counterpart instance (formed by mean activity durations) serves as a good approximation of the stochastic instance. This is exploited by our heuristics and other heuristics such as STC. However, in certain practical domains (e.g. maintenance scheduling), the duration of some activities is known a-priori with accuracy, while the duration of other activities follows a distribution with very high variance. In maintenance scheduling, for example, "inspection" activities have known durations but "repair" activities might be (un)necessary with certain probabilities. We would like to investigate performance on such instances which cannot be approximated well by their determinitic counterpart.

Acknowledgements We would like to thank our anonymous reviewers for their constructive remarks. This research belongs to the *Job Scheduling Problem* part of the *Rolling Stock Life Cycle Logistics* applied research and development programme, funded by *NS/NedTrain*.

References

1. Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.
2. Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003.
3. Behzad Ashtiani, Roel Leus, and Mir-Bahador Aryanezhad. New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, 14(2):157–171, 2011.
4. Francisco Ballestín. When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166, 2007.
5. Francisco Ballestin and Roel Leus. Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, 18(4):459–474, 2009.
6. J-H Bartels and Jürgen Zimmermann. Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, 193(3):805–819, 2009.
7. Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.
8. Felix Bomsdorf and Ulrich Derigs. A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. *Or Spectrum*, 30(4):751–772, 2008.
9. Kristof Braeckmans, Erik Demeulemeester, Willy Herroelen, and Roel Leus. Proactive resource allocation heuristics for robust project scheduling. *DTEW Research Report 0567*, pages 1–22, 2005.
10. Frits de Nijs and Tomas Klos. A novel priority rule heuristic: Learning from justification. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.
11. Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316, 2011.

12. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991.
13. Jane N Hagstrom. Computational complexity of pert problems. *Networks*, 18(2):139–147, 1988.
14. Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.
15. Willy Herroelen, Erik Demeulemeester, and Bert De Reyck. A note on the paper resource-constrained project scheduling: Notation, classification, models and methods by brucker et al. *European Journal of Operational Research*, 128(3):679–688, 2001.
16. Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550–565, 2004.
17. Willy Herroelen and Roel Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.
18. Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37, 2006.
19. Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.
20. Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.
21. Patricio Lamas Vilches and Erik Demeulemeester. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Available at SSRN 2464056*, 2014.
22. Roel Leus. Resource allocation by means of project networks: dominance results. *Networks*, 58(1):50–58, 2011.
23. Roel Leus, Christian Artigues, and Fabrice Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. In *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, pages 101–105. IEEE, 2011.
24. Roel Leus and Willy Herroelen. Stability and resource allocation in project planning. *IIE transactions*, 36(7):667–682, 2004.
25. Rolf H Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems i – general strategies. *Zeitschrift für Operations Research*, 28(7):193–260, 1984.
26. Rolf H Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems ii – set strategies. *Zeitschrift für Operations Research*, 29(3):65–104, 1985.
27. Angelo Oddi and Riccardo Rasconi. Iterative flattening search on rcpsp/max problems: Recent developments. In *Recent Advances in Constraints*, pages 99–115. Springer, 2009.
28. Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving rcpsp/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289, 2013.
29. Frederik Stork. Branch-and-bound algorithms for stochastic resource-constrained project scheduling. *Technical rep*, pages 702–2000, 2000.
30. Stijn Van de Vonder, Francisco Ballestin, Erik Demeulemeester, and Willy Herroelen. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, 52(1):11–28, 2007.
31. Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.
32. Stijn Van de Vonder, Erik Demeulemeester, Willy Herroelen*, and Roel Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2):215–236, 2006.

MISTA 2015

Optimal schedule of a statistical process control with a nonlinear expected loss

Valery Y. Glizer · Vladimir Turetsky

Abstract A problem of constructing an optimal state-feedback schedule for a statistical process control with a variable sampling time-interval is considered. The aim of the schedule is to minimize the expected loss, caused by delay in detecting a process change. The case where this loss depends nonlinearly on the sampling time-interval is treated. Two approaches to the design of the optimal schedule are proposed. The first approach based on converting the original optimization problem to an equivalent optimal control problem and applying to the latter the Pontryagin's Maximum Principle, which leads to an exact analytical solution. The second approach is based on a discretization of the original problem and using proper mathematical programming tools to the discrete problem, which provides an approximate numerical solution. The schedules, obtained by these two approaches, are compared to each other in numerical examples. Moreover, in such examples, the analytical schedule is compared to the suboptimal composite schedule of a statistical process control, known in the literature.

1 Introduction

The Statistical Process Control (SPC) (see e.g. [1]) means monitoring a process state by using samples of its key characteristic index in some time-intervals. It is widely used in industry, medicine, environment etc, and its objective is to minimize losses caused by delay in the detection of undesirable accidents, subject to acceptable inspection expenses. For many years, the traditional SPC practice in monitoring a process was to take samples of the process characteristic index with a Fixed Sampling Time-Interval (FSTI). The idea of using variable/adaptive SPC sampling time-intervals to achieve process stability is known in literature for about 25 recent years. For the first time it

Valery Y. Glizer
Department of Applied Mathematics, Ort Braude College of Engineering, 51 Snunit Str., P.O.B. 78, Karmiel 2161002, Israel
E-mail: valery48@braude.ac.il

Vladimir Turetsky
Department of Applied Mathematics, Ort Braude College of Engineering, 51 Snunit Str., P.O.B. 78, Karmiel 2161002, Israel
E-mail: turetsky1@braude.ac.il

was presented in the work [2]. Then, this idea was developed in a number of works (see e.g. [3–13]). A detailed review on the topic can be found in [12].

Due to [2], the reaction time of SPC to process change is considered as a main criterion of optimality. The alternative criterion, proposed in [14], and further developed in [12, 13], is the expected loss caused by delay in detecting process change. This criterion is more general and at the same time more usable, from the engineering viewpoint. The relation between the expected loss and the reaction time (the delay in detection of a process change) is not necessarily linear. There are various processes where such a relation is non-linear. For instance, the following processes can be mentioned: (1) fires propagation [15], (2) oil spills spreading [16], (3) cholesterol plaque growth [17], (4) epidemics propagation [18], (5) fatigue crack growth in ship hull structures [19], and some others.

Genichi Taguchi proposed the quadratic dependence of the expected loss on some critical performance parameter whose desired value should be as low as possible within the existing constraints [14], yielding the so called "the smaller – the best" loss function. In modern industry, medicine, natural environment defence, etc, the process control becomes an indispensable part of the process itself. Therefore, the detection delay, being a critical performance parameter of the process control, becomes a critical performance parameter of the process itself. Thus, the Taguchi model yields a quadratic dependence of the expected loss on the detection delay. It should be noted that the delay time in SPC is a random variable, which distribution depends on the process change extent. Therefore, expected (in the statistical sense) loss becomes the actual optimization criterion.

In this paper, the SPC schedule design problem is formulated as a calculus of variations problem, in which the expected loss should be minimized by a proper choice of a sampling time-interval. In this problem, two types of constraints are imposed. The first type of constraints is an isoperimetric (weak) constraint, meaning that the average variable sampling time-interval is equal to a properly prechosen constant nominal sampling time-interval. The second type of constraints is two geometric (hard) constraints, which determine the lower and upper bounds of the sampling time-interval. The latter is not addressed by the classical calculus of variations theory. This makes the considered extremal problem to be non-standard. In the previous work of the authors (see [13]), this problem was solved by its approximate decomposition into two simpler subproblems. Based on this decomposition, two approximate solutions, analytical and numerical, were derived. In the present paper, we solve the entire (without a decomposition) extremal problem. Two methods of its solution are proposed. The first method transforms equivalently the original extremal problem to an optimal control problem. Further, this optimal control problem is solved by using the Pontryagin's Maximum Principle yielding an exact analytical solution of the original problem. This solution constitutes the optimal (subject to the minimum of the expected loss) schedule of the SPC. In the second method, the original (continuous) extremal problem is replaced approximately by a discrete finite-dimensional extremal problem. The latter is solved by using corresponding mathematical programming tools, yielding an approximate numerical solution of the original extremal problem. This solution constitute a suboptimal schedule of the SPC.

It is important to note, that the SPC schedule, proposed in this paper, differs considerably from the well known train schedule, bus schedule, flight schedule, etc. Namely, the latter schedules are created in advance for some period, and they do not depend as a rule on a current state. In contrast with these schedules, the SPC schedule

does depend on a current state, and it is not designed in advance for a period of the process control. By using the terminology of control engineering, the classical train, bus, flight, etc schedules can be called open-loop schedules, while the SPC schedule, proposed in this paper, can be called a state-feedback schedule.

2 Problem statement

2.1 Process monitoring

Similarly to [12,13], we deal with the case, where the monitoring a performance parameter x of a controlled process is carried out according to a sample mean $\bar{x} \sim N(\mu, \sigma/\sqrt{n})$. The sample size n is assumed fixed. Thus, the standard score in an in-control state is

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \sim N(0,1). \tag{1}$$

The upper and lower limits of a standard Shewhart control chart for z are $z_{\min} = -3$ and $z_{\max} = 3$, respectively [1]. Therefore, the false alarm probability α (type I error), i.e. the probability of the event $z \notin [-3,3]$, is

$$\alpha = 1 - \frac{1}{\sqrt{2\pi}} \int_{-3}^{3} \exp(-z^2/2)dz = 1 - [\Phi(3) - \Phi(-3)] = 0.0027, \tag{2}$$

where $\Phi(z) = \left(1/\sqrt{2\pi}\right) \int_{-\infty}^{z} \exp(-\zeta^2/2)d\zeta$.

If the performance parameter mean value shifts by Δ, i.e. a new mean value is $\mu' = \mu + \Delta$, and the value of σ remains unchanged, then the distribution of z becomes

$$z \sim N(\delta, 1), \qquad \delta = \frac{\Delta}{\sigma/\sqrt{n}} \tag{3}$$

where δ is the so-called *signal-to-noise ratio*. The probability of discovering the shift (receiving the signal) by a single sample is the probability of the event $z \notin [-3,3]$ subject to (3):

$$1 - \beta = 1 - \frac{1}{\sqrt{2\pi}} \int_{-3}^{3} \exp(-(z-\delta)^2/2)dz = 1 - [\Phi(3-\delta) - \Phi(-3-\delta)], \tag{4}$$

where β is the probability of a type II error (not discovering the shift), depending on a normalized shift δ.

2.2 Constraints on variable sampling time-interval

Consider the adaptive statistical process control with a variable sampling time-interval $u(z)$, where z is a current value of the standard score, given by (1). Since the value of the sampling time-interval should depend only on the absolute value of the standard score z, the function $u(z)$ is even ($u(-z) = u(z)$). Therefore, in what follows, we consider the function $u(z)$ on the interval $[0,3]$. Also, for the sake of simplicity, we assume that $\delta \geq 0$. The case $\delta \leq 0$ is treated similarly.

Further, it is assumed that the expected sampling time-interval in the case of unshifted z, ($\delta = 0$), is equal to a prescribed nominal value T which, due to [12,13], yields the isoperimetric (integral) constraint on $u(z)$

$$\int_0^3 \exp(-z^2/2)[u(z) - T]dz = 0. \tag{5}$$

Also, it is assumed that the function $u(z)$ is bounded as:

$$0 < u_{\min} \le u(z) \le u_{\max}, \quad z \in [0, 3], \tag{6}$$

where $u_{\min} < T$ and $u_{\max} > T$.

In what follows, for the sake of convenience, we set $T = 1$, yielding

$$u_{\min} < 1, \quad u_{\max} > 1, \tag{7}$$

and

$$\int_0^3 \exp(-z^2/2)u(z)dz = a, \quad a \triangleq \int_0^3 \exp(-z^2/2)dz. \tag{8}$$

2.3 Quadratic expected loss model

If the process shift remains constant, the time t_d, required for discovering the shift (so-called *time to signal*), is the sum of a random amount N_d of random independent and identically distributed sampling time-intervals u_i, conditionally independent of N_d:

$$t_d = \sum_{i=1}^{N_d} u_i. \tag{9}$$

The cost functional, minimized by a properly chosen sampling time-interval $u(z)$, is the mathematical expectation $\mathbf{E}(L)$ of the loss L, caused by the shift detection delay (expected loss). The assumption that the loss L is proportional to t_d^2, ($L = kt_d^2$, $k > 0$ is a constant), yields

$$\mathbf{E}(L) = k\mathbf{E}(t_d^2). \tag{10}$$

The problem of constructing an optimal SPC schedule consists in a searching the sampling time-interval $u(z)$, which minimizes the cost functional $\mathbf{E}(t_d^2)$, subject to the constraints (6)-(8).

By virtue of [12,13], we have

$$\mathbf{E}(t_d^2) = A\left[\int_0^3 \psi(z,\delta)u^2(z)dz + B\left(\int_0^3 \psi(z,\delta)u(z)dz\right)^2\right], \tag{11}$$

where

$$A \triangleq \frac{\exp(-\delta^2/2)}{(1-\beta)\beta\sqrt{2\pi}} > 0, \quad B \triangleq \frac{2\exp(-\delta^2/2)}{(1-\beta)\sqrt{2\pi}} > 0, \tag{12}$$

$$\psi(z,\delta) \triangleq 2\exp\left(-z^2/2\right)\cosh(\delta z) > 0, \quad z \in [0, 3]. \tag{13}$$

Thus, due to (11)-(12), the mentioned above problem of constructing an optimal SPC schedule becomes as follows:

$$J\big(u(z)\big) \triangleq \int_0^3 \psi(z,\delta)u^2(z)dz + B\left(\int_0^3 \psi(z,\delta)u(z)dz\right)^2 \to \min_{u(z)}, \quad \text{s.t. } (6)-(8). \tag{14}$$

The objective of the paper is to solve the problem (14), thus constructing the optimal SPC schedule.

3 Analytical solution of (14)

The problem (14) is a nonstandard variational calculus problem with two types of constraints, an isoperimetric constraint (see (8)) and a geometric constraint (see (6)), imposed on the minimizing function. Note that the classical calculus of variations theory does not study the extremal problems with geometric constraints (see, e.g., [20]). If we omit the geometric constraints in (14), then the Euler–Lagrange equation, associated with the resulting (reduced) problem, becomes algebraic, because the integrands in the functional $J\big(u(z)\big)$ and in the isoperimetric constraint (8) of the reduced problem are independent of the derivative of the minimizing function $u(z)$. This equation admits a solution, not necessarily satisfying the geometric constraints (6) which is unacceptable.

Here, we propose another approach to solving this problem. This approach consists in an equivalent transformation of the original problem into an optimal control problem. The latter is analyzed by application of the Pontryagin's Maximum Principle (PMP) [21].

3.1 Transformation of (14)

Let us introduce the following auxiliary functions of $z \in [0,3]$:

$$w_1(z) = \int_0^z \psi(\zeta,\delta)u^2(\zeta)d\zeta, \tag{15}$$

$$w_2(z) = \int_0^z \psi(\zeta,\delta)u(\zeta)d\zeta, \tag{16}$$

$$w_3(z) = \int_0^z \exp\big(-\zeta^2/2\big)u(\zeta)d\zeta. \tag{17}$$

These functions satisfy the differential equations

$$\frac{dw_1}{dz} = \psi(z,\delta)u^2(z), \tag{18}$$

$$\frac{dw_2}{dz} = \psi(z,\delta)u(z), \tag{19}$$

$$\frac{dw_3}{dz} = \exp\big(-z^2/2\big)u(z), \tag{20}$$

and the initial conditions

$$w_1(0) = 0, \quad w_2(0) = 0, \quad w_3(0) = 0. \tag{21}$$

Moreover, due to the integral constraint (8), $w_3(z)$ satisfies the terminal condition

$$w_3(3) = a. \tag{22}$$

By using (15)-(16), the cost functional $J\big(u(z)\big)$ becomes

$$J\big(u(z)\big) = w_1(3) + B\big(w_2(3)\big)^2. \tag{23}$$

Thus, we have transformed the problem (14) into the following equivalent optimal control problem: to find the control function $u(z)$, transferring the system (18) – (20) from the initial position (21) to the terminal position (22) and minimizing the cost functional (23), subject to the geometric constraint (6)-(7). This optimal control problem is non-linear with respect to $u(z)$, and in what follows, it is called the Non-linear Optimal Control Problem (NOCP).

Remark 1 Due to [22] (see, Section 9.2, Theorem 3), the NOCP has a solution (optimal control).

3.2 Solution of the NOCP by using the PMP

The Variational Hamiltonian of the NOCP has the form

$$H = H(w_1, w_2, w_3, u, \lambda_1, \lambda_2, \lambda_3, z) = \lambda_1 \psi(\delta, z) u^2 + \lambda_2 \psi(\delta, z) u + \lambda_3 \exp\big(-z^2/2\big) u, \tag{24}$$

where $\lambda_i = \lambda_i(z)$, $(i = 1, 2, 3)$ are the costate variables.

These costate variables satisfy the differential equations

$$\frac{d\lambda_1}{dz} = -\frac{\partial H}{\partial w_1} = 0, \quad z \in [0, 3], \tag{25}$$

$$\frac{d\lambda_2}{dz} = -\frac{\partial H}{\partial w_2} = 0, \quad z \in [0, 3], \tag{26}$$

$$\frac{d\lambda_3}{dz} = -\frac{\partial H}{\partial w_3} = 0, \quad z \in [0, 3], \tag{27}$$

and the transversality conditions for λ_1 and λ_2

$$\lambda_1(3) = -\frac{\partial J}{\partial w_1(3)} = -1, \tag{28}$$

$$\lambda_2(3) = -\frac{\partial J}{\partial w_2(3)} = -2B\gamma, \quad \gamma \stackrel{\triangle}{=} w_2(3). \tag{29}$$

Due to the PMP, an optimal control $u^*(z)$ of the NOCP necessarily satisfies the following condition

$$u^*(z) = \arg \max_{u_{\min} \le u(z) \le u_{\max}} H\big(w_1(z), w_2(z), w_2(z), u(z), \lambda_1(z), \lambda_2(z), \lambda_3(z), z\big). \tag{30}$$

Thus, any control $u(z)$, satisfying the equations (24), (30), (18) –(22) and (25) – (29) is an optimal control candidate in the NOCP. We start with the obtaining such a control by solving the equations (25) – (29). These equations yield the solution

$$\lambda_1(z) = -1, \quad \lambda_2(z) = -2B\gamma, \quad \lambda_3(z) = C = \text{const}, \quad z \in [0, 3]. \tag{31}$$

By substituting (31) into (24) and using (13), the Variational Hamiltonian becomes

$$H = \exp\big(-z^2/2\big)G(u,z,\gamma,C), \tag{32}$$

where the function $G(u,z,\gamma,C)$ has the form

$$G(u,z,\gamma,C) = \big(C - 4B\gamma\cosh(\delta z)\big)u - 2\cosh(\delta z)u^2. \tag{33}$$

Due to (30), (32) – (33) and Remark 1, the optimal control of the NOCP has the form

$$u^*(z,\gamma,C) = \begin{cases} u_{\min}, & \bar{u}(z,\gamma,C) \le u_{\min}, \\ \bar{u}(z,\gamma,C), & u_{\min} < \bar{u}(z,\gamma,C) \le u_{\max}, \\ u_{\max}, & \bar{u}(z,\gamma,C) > u_{\max}, \end{cases} \tag{34}$$

where

$$\bar{u}(z,\gamma,C) = \frac{C}{4\cosh(\delta z)} - B\gamma \tag{35}$$

is the unique solution of the following equation with respect to u:

$$\frac{\partial G(u,z,\gamma,C)}{\partial u} = 0. \tag{36}$$

Remark 2 The derivative of $\bar{u}(z,\gamma,C)$ with respect to z has the form

$$\frac{d\bar{u}(z,\gamma,C)}{dz} = -\frac{C\delta\sinh(\delta z)}{4\cosh^2(\delta z)}. \tag{37}$$

Hence, for any γ and any $C > 0$, $\delta > 0$, the function $\bar{u}(z,\gamma,C)$ is monotonically decreasing on the interval $z \in [0,3]$. Therefore, for such γ, C, δ, the control $u^*(z,\gamma,C)$ is a non-increasing function of $z \in [0,3]$.

In order to use the equation (34), one has to know the constants γ and C. These constants should be chosen in such a way that the resulting control (34) will transfer the system (18) – (20) from the initial position (21) to the terminal position (22) and $w_2(3) = \gamma$. Due to (18) – (22), this means that the values γ and C should satisfy the set of two algebraic equations

$$\Lambda_1(\gamma,C) \triangleq \int_0^3 \exp(-z^2/2)u^*(z,\gamma,C)dz - a = 0, \tag{38}$$

$$\Lambda_2(\gamma,C) \triangleq \int_0^3 \psi(z,\delta)u^*(z,\gamma,C)dz - \gamma = 0. \tag{39}$$

Remark 3 By virtue of Remark 1, the set (38)-(39) has a solution. If this set has more than one solution, we choose the solution $\big(\gamma = \gamma^*, C = C^*\big)$, which provides the minimal value of the cost functional (23) in comparison with the other solutions.

Lemma 1 *There are no solutions of the set (38)-(39) in the half-planes*

$$C \le 4B\gamma + 4u_{\min} \triangleq C_{\min}(\gamma), \quad C \ge 4B\gamma\cosh(3\delta) + 4u_{\max}\cosh(3\delta) \triangleq C_{\max}(\gamma), \tag{40}$$

$$\gamma \le u_{\min}\int_0^3 \psi(z,\delta)dz \triangleq \gamma_{\min}, \quad \gamma \ge u_{\max}\int_0^3 \psi(z,\delta)dz \triangleq \gamma_{\max} \tag{41}$$

of the plane (γ, C).

Lemma 2 *Let $\delta > 0$. Let (γ, C) be a solution of the set (38)-(39). Then, the component γ of this solution satisfies the inequality*

$$2a < \gamma < 2a\cosh(3\delta). \tag{42}$$

Let us define

$$\Gamma_{\min} \stackrel{\triangle}{=} \max\left\{u_{\min}\int_0^3 \psi(z,\delta)dz, 2a\right\}, \tag{43}$$

$$\Gamma_{\max} \stackrel{\triangle}{=} \min\left\{u_{\max}\int_0^3 \psi(z,\delta)dz, 2a\cosh(3\delta)\right\}. \tag{44}$$

Due to (7),(8),(13),(43)-(44), as well as the integral mean value theorem and the inequality $1 \le \cosh(\delta z) \le \cosh(3\delta)$, $0 \le z \le 3$, we have the following inequality for $\delta > 0$: $0 < \Gamma_{\min} < \Gamma_{\max}$. Similarly, for any $\gamma \ge 0$, we obtain the inequality $0 < C_{\min}(\gamma) < C_{\max}(\gamma)$.

Consider the domain

$$\Omega \stackrel{\triangle}{=} \Big\{(\gamma, C) : \gamma \in \big(\Gamma_{\min}, \Gamma_{\max}\big), C \in \big(C_{\min}(\gamma), C_{\max}(\gamma)\big)\Big\}. \tag{45}$$

Based on Lemmas 1,2, and the equations (43)-(44),(45), one directly has the following two theorems.

Theorem 1 *Let $\delta > 0$. Let (γ, C) be a solution of the set (38)-(39). Then, $(\gamma, C) \in \Omega$.*

Theorem 2 *Let $\delta = 0$. Then, the set (38)-(39) has the unique solution $\big(\gamma = 2a, C = 8Ba + 4\big)$.*

The following two lemmas can be used in constructing methods of solution of the set (38)-(39).

Lemma 3 *For any $\delta > 0$ and $\gamma \in \big(\Gamma_{\min}, \Gamma_{\max}\big)$, the equation (38) has the unique solution $C = \widetilde{C}(\gamma)$, and*

$$\widetilde{C}(\gamma) \in \big(C_{\min}(\gamma), C_{\max}(\gamma)\big). \tag{46}$$

Lemma 4 *For any $\delta > 0$ and $\gamma \in \big(\Gamma_{\min}, \Gamma_{\max}\big)$, the equation (39) has the unique solution $C = \bar{C}(\gamma)$, and*

$$\bar{C}(\gamma) \in \big(C_{\min}(\gamma), C_{\max}(\gamma)\big). \tag{47}$$

Remark 4 Based on Lemmas 3 and 4, the γ-component of solution of the set (38)-(39) can be obtained by solving with respect to γ either the equation $\Lambda_2\big(\gamma, \widetilde{C}(\gamma)\big) = 0$, or the equation $\Lambda_1\big(\gamma, \bar{C}(\gamma)\big) = 0$, or the equation $\widetilde{C}(\gamma) = \bar{C}(\gamma)$.

4 Approximate numerical solution of (14)

Let us divide the interval $[0,3]$ into N equal subintervals by the collocation points

$$z_i = i\Delta z, \quad i = 0,1,\ldots,N, \quad \Delta z = 3/N. \tag{48}$$

Then, based on (48), let us approximate the integrals in the cost functional $J\big(u(z)\big)$ and in the isoperimetric constraint (8) by using the left rectangles formula [23].

Thus, the cost functional is approximated as

$$J\big(u(z)\big) \approx \tilde{J}^N(U) \stackrel{\triangle}{=} \Delta z \sum_{i=0}^{N-1} \psi(z_i,\delta)U_i^2 + B\left(\Delta z \sum_{i=0}^{N-1} \psi(z_i,\delta)U_i\right)^2, \tag{49}$$

where the vector $U \in E^N$ is

$$U = (U_0,U_1,\ldots,U_{N-1})^T = (u(z_0),u(z_1),\ldots,u(z_{N-1}))^T. \tag{50}$$

The constraint (8) is approximated as

$$\Delta z \sum_{i=0}^{N-1} \exp(-z_i^2/2)U_i = \Delta z \sum_{i=0}^{N-1} \exp(-z_i^2/2), \tag{51}$$

where the right-hand side expression approximates the value a.

Thus, dividing (49) and (51) by Δz and taking into account the geometric constraint (6) yield the following finite-dimensional Quadratic Programming Problem (QPP):

$$J^N(U) \stackrel{\triangle}{=} \sum_{i=0}^{N-1} \psi(z_i,\delta)U_i^2 + B\Delta z\left(\sum_{i=0}^{N-1} \psi(z_i,\delta)U_i\right)^2 \to \min_U \tag{52}$$

subject to

$$\sum_{i=0}^{N-1} \exp(-z_i^2/2)U_i = a_N, \qquad a_N \stackrel{\triangle}{=} \sum_{i=0}^{N-1} \exp(-z_i^2/2), \tag{53}$$

$$u_{\min} \le U_i \le u_{\max}, \qquad i = 0,1,\ldots,N-1. \tag{54}$$

The QPP (52)-(54) can be solved by using standard optimization tools, for example, the MATLAB function "quadprog". It is reasonable to expect that for large enough N, the components $U_i = U_i^*$, $(i = 0,1,...,N-1)$ of solution of QPP will be close to the corresponding values $u^*(z_i,\gamma^*,C^*)$, $(i = 0,1,...,N-1)$ of the optimal control in the NOCP solved in Section 3.2. In such a case, the optimal value of the cost functional (52) multiplied by Δz will be close to the optimal value of the cost functional (23).

5 Numerical evaluation of optimal and suboptimal schedules

For the numerical evaluation, the following two sets of parameters are chosen:

$$\text{(A)} \quad u_{\min} = 0.5, \quad u_{\max} = 3.5;$$

$$\text{(B)} \quad u_{\min} = 0.1, \quad u_{\max} = 2.5.$$

Based on these sets of parameters, the suboptimal composite SPC schedule was evaluated numerically in the work [13]. Here, we use the same sets of parameters to evaluate the SPC schedules (optimal and approximate), designed in the previous sections, as well as to compare the optimal schedule to the composite suboptimal one.

The construction of the composite suboptimal schedule is based on the behavior of the value B as a function of $\delta \in [0, 3]$. This function monotonically decreases from the large enough value (about 300) for $\delta = 0$ to the small enough value (close to 0) for $\delta = 3$. Due to this behavior of B, in the work [13] it was shown the existence of the value $\delta = \delta^*$ such that, for $\delta \in [0, \delta^*)$, the second addend of the cost functional in (14) dominates, while for $\delta \in (\delta^*, 3]$, the first addend dominates. Using this observation, in [13] the composite suboptimal SPC schedule was designed in the form

$$u_c(z) = \begin{cases} u_l^*(z), \ \delta \in [0, \delta^*), \\ \\ u_{nl}^*(z), \ \delta \in [\delta^*, 3], \end{cases} \qquad z \in [0, 3], \tag{55}$$

where $u_l^*(z)$ and $u_{nl}^*(z)$ are the solutions of the problems

$$J_1(u) \stackrel{\triangle}{=} \int_0^3 \psi(z, \delta) u(z) dz \to \min_u \quad \text{s.t. (6)} - \text{(8)} \tag{56}$$

and

$$J_2(u) \stackrel{\triangle}{=} \int_0^3 \psi(z, \delta) u^2(z) dz \to \min_u \quad \text{s.t. (6)} - \text{(8)}, \tag{57}$$

respectively. In order to determine δ^*, in [13] the following condition was proposed: $J(u_l^*)|_{\delta=\delta^*} = J(u_{nl}^*)_{\delta=\delta^*}$. In such a case, $J(u_l^*) < J(u_{nl}^*)$ for all $\delta \in [0, \delta^*)$, and $J(u_l^*) > J(u_{nl}^*)$ for all $\delta \in (\delta^*, 3]$.

In Fig. 1, the optimal schedule $u^*(z, \gamma, C)$, given by the analytical expression (34), is compared with the solution of the approximating problem (52) – (54) for the set (A) with $\delta = 2.9$ (Fig. 1a) and for the set (B) with $\delta = 2.5$ (Fig. 1b). It is seen that the approximation, obtained for $N = 100$, and the optimal schedule match well.

For obtaining the schedule $u^*(z, \gamma, C)$, the set (38) – (39) was solved numerically. The value of γ was calculated by applying the bisection algorithm to the equation $\Lambda_1(\gamma, \bar{C}(\gamma)) = 0$ for $\gamma \in (\Gamma_{\min}, \Gamma_{\max})$, where $\bar{C}(\gamma)$ is the solution with respect to C of the equation (39) mentioned in Lemma 4. The function $\bar{C}(\gamma)$ also was derived by the bisection method for $(\gamma, C) \in \Omega$. The numerical solution $(\gamma, C = \bar{C}(\gamma))$ of the set (38) – (39) and the absolute values of the functions $\Lambda_1(\gamma, C)$, $\Lambda_2(\gamma, C)$ are presented in Table 1. It is seen that the solution is obtained with the accuracy better than 10^{-4}.

In Fig. 2, the optimal schedule $u^*(z, \gamma, C)$ is compared to the composite schedule $u_c(z)$ for the set (A) and two values of δ: $\delta = 1 < \delta^* = 2.47$ where $u_c(z) = u_l^*(z)$ (see Fig. 2a), and $\delta = 2.6 > \delta^* = 2.47$ where $u_c(z) = u_{nl}^*(z)$ (see Fig. 2b). In Fig. 3, such a comparison with the same values of δ is presented for the set (B) where $\delta^* = 2.58$.

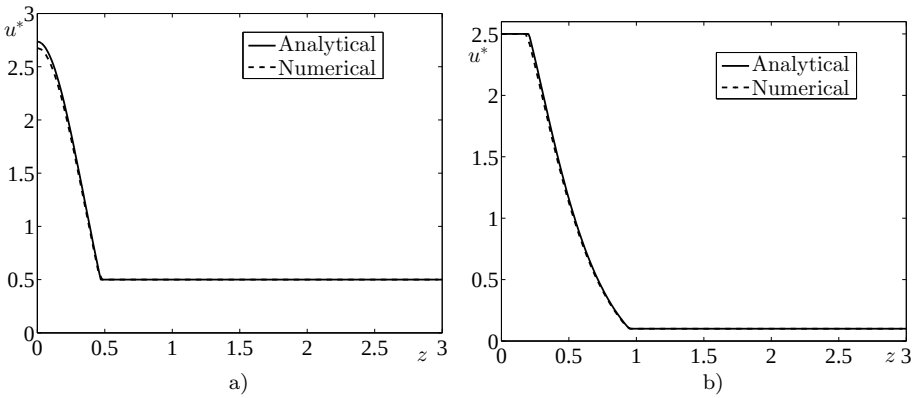


Fig. 1 Optimal (analytical) vs. approximate (numerical) schedule

Table 1 Numerical solution of the set (38) – (39)

$u_{\min}$	$u_{\max}$	δ	γ	C	$\lvert\Lambda_1(\gamma, C)\rvert$	$\lvert\Lambda_2(\gamma, C)\rvert$
0.5	3.5	2.5	21.13	20.58	$7.05 \cdot 10^{-5}$	$7.59 \cdot 10^{-5}$
0.1	2.5	2.9	12.62	13.39	$2.90 \cdot 10^{-5}$	$6.97 \cdot 10^{-5}$

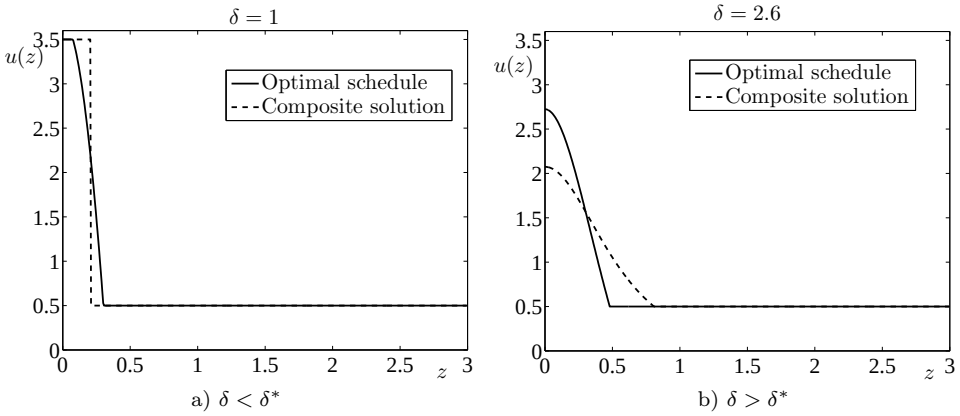


Fig. 2 Optimal schedule vs. composite schedule: set (A)

It is seen that in all cases the optimal and composite schedules, presented as functions of z, differ considerably. In Fig. 4, the ratio $J(u^*)/J(u_c)$ is depicted as a function of δ for two parameter sets. It is seen that the optimal schedule produces smaller values of the cost functional than the composite schedule. The advantage is not dramatic, but the calculating $u^*(z, \gamma, C)$ needs an essentially less computational effort than the calculating $u_c(z)$.

6 Conclusions

In this paper, the problem of constructing an optimal state-feedback schedule for the statistical process control was considered. The expected loss, quadratically dependent

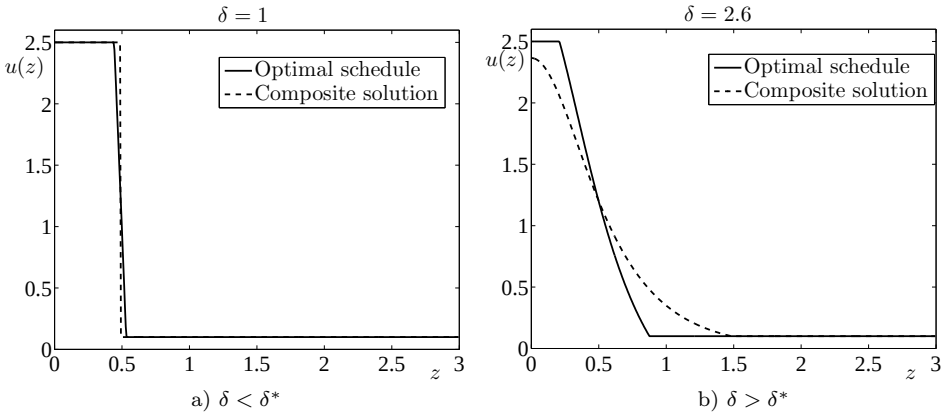


Fig. 3 Optimal schedule vs. composite schedule: set (B)

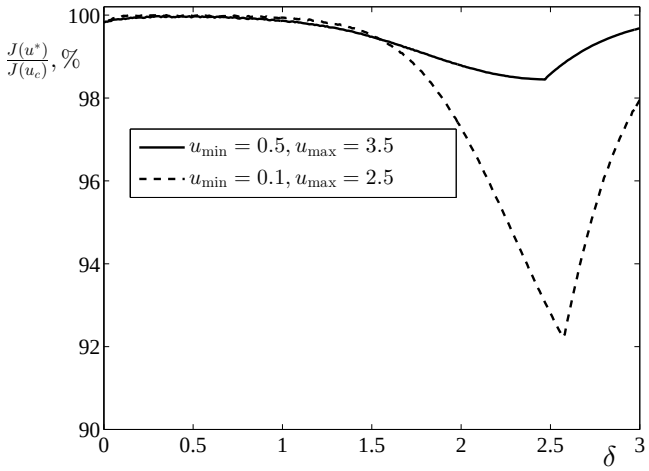


Fig. 4 Optimal schedule vs. composite schedule: cost functional value

on the sampling time-interval, was chosen as the criterion of the optimization. Two methods of solution of this problem were proposed. The first method transforms the original optimization problem to an equivalent optimal control problem. Then, the latter was solved analytically by using the Pontryagin's Maximium Principle, which yields the optimal schedule for the considered statistical process control. The second method uses a discretization of the original optimization problem. This leads to a finite-dimensional quadratic optimization problem, approximating the original one. This new optimization problem is solved by using the MATLAB function "quadprog", providing the suboptimal schedule for the statistical process control.

The optimal and suboptimal schedules were evaluated by numerical examples. This evaluation has shown a good match of the optimal analytical schedule and the suboptimal numerical schedule. The optimal schedule also was compared to the suboptimal composite schedule, designed in a previous work of the authors. This comparison has shown that the optimal schedule produces a smaller expected loss than the subopti-

mal composite schedule does, and the former requires for its design less computational effort than the latter needs.

References

1. D. C. Montgomery Introduction to statistical quality control, 734 pp. John Wiley and Sons Inc., New York, NY (2005)
2. M. R. Reynolds, R. W. Amin, J. C. Arnold and J. Nachlas, X charts with variable sampling intervals, Technometrics, Vol. 30, pp. 181–192 (1988)
3. R. W. Amin and R. Hemasinha, The switching behavior of X charts with variable sampling intervals, Communication in Statistics - Theory and Methods, Vol. 22, pp. 2081–2102 (1993)
4. R. W. Amin and R. W. Miller, A robustness study of X charts with variable sampling intervals, Journal of Quality Technology, Vol. 25, pp. 36–44 (1993)
5. A. F. B. Costa, X control chart with variable sample size, Journal of Quality Technology, Vol. 26, pp. 155–163 (1994)
6. S. S. Prabhu, D. C. Montgomery and G. C. Runger, A combined adaptive sample size and sampling interval X control scheme, Journal of Quality Technology, Vol. 26, pp. 164–176 (1994)
7. M. R. Reynolds, Evaluating properties of variable sampling interval control charts, Sequentional Analysis, Vol. 14, pp. 59–97 (1995)
8. A. F. B. Costa, X charts with variable sample sizes and sampling intervals, Journal of Quality Technology, Vol. 29, pp. 197–204 (1997)
9. A. F. B. Costa, Joint X and R control charts with variable parameters, IIE Transactions, Vol. 30, pp. 505–514 (1998)
10. A. F. B. Costa, X charts with variable parameters, Journal of Quality Technology, Vol. 31, pp. 408–416 (1999)
11. A. F. B. Costa and M. S. De Magalhães, An adaptive chart for monitoring the process mean and variance, Quality and Reliability Engineering International, Vol. 23, pp. 821–831 (2007)
12. E. Bashkansky and V. Y. Glizer, Novel approach to adaptive statistical process control optimization with variable sampling interval and minimum expected loss, International Journal of Quality Engineering and Technology, Vol. 3, pp. 91–107 (2012)
13. V. Y. Glizer, V. Turetsky and E. Bashkansky, Statistical process control optimization with variable sampling interval and nonlinear expected loss, Journal of Industrial and Management Optimization, Vol. 11, pp. 105–133 (2015)
14. G. Taguchi, S. Chowdhury and Y. Wu, Taguchi's Quality engineering handbook, 1662 pp. John Wiley and Sons Inc., Hoboken, NJ (2007)
15. V. Babrauskas, Heat release rates, in SFPE Handbook of Fire Protection Engineering, (Ed. P.J. DiNenno), National Fire Protection Association, pp. 1–59 (2008)
16. P. Sebastião and C. G. Soares, Modeling the fate of oil spills at sea, Spill Science and Technology Bulletin, Vol. 2, pp. 121–131 (1995)
17. M. A. K. Bulelzai and J. L. A. Dubbeldam, Long time evolution of atherosclerotic plaques, Journal of Theoretical Biology, Vol. 297, pp. 1–10 (2012)
18. T. E. Carpenter, J. M. O'Brien, A. Hagerman and B. McCarl, Epidemic and economic impacts of delayed detection of foot-and-mouth disease: A case study of a simulated outbreak in California, Journal of Veterinary Diagnostic Investigation, Vol. 23, pp. 26–33 (2011)
19. S. Kim and D. M. Frangopol, Optimum inspection planning for minimizing fatigue damage detection delay of ship hull structures, International Journal of Fatigue, Vol. 33, pp. 448–459 (2011)
20. I. M. Gelfand and S. V. Fomin, Calculus of variations, 240 pp. Prentice-Hall, Englewood Cliffs, NJ (1963)
21. L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze and E. F. Mishchenko, The mathematical theory of optimal processes, 360 pp. Interscience, New York, NY (1962)
22. A. D. Ioffe and V. M. Tihomirov, Theory of extremal problems, 460 pp. North-Holland Pub. Co., Amsterdam, Netherlands (1979)
23. P. J. Davis and P. Rabinowitz, Methods of numerical integration, 624 pp. Dover Publications, Inc., Mineola, NY (2007)

MISTA 2015

Scheduling competition in the airline industry and the issue of duplicate bookings

Ryosuke Ishii · Kuninori Nakagawa

Abstract In this study, we consider the relationship between a passenger's behaviour and competition over flight schedules and prices between two airlines. In particular, by focusing on how an early option to purchase tickets affects duopoly competition in this industry, we consider duplicate bookings when passengers' travel plans are uncertain. In our model, airlines can set their ticket prices twice: before and after passengers know their exact travel plans. We find that in a subgame perfect equilibrium, flights operate on an efficient schedule from a passenger perspective (i.e. a passenger's expected loss of utility is minimized).

1 Introduction

In this study, we analyse the competition over flight schedules and prices between two airlines, with a focus on airfare discounts for very early reservations. Ishii and Nakagawa (2015)[12] analyse how discounts for early booking affect the scheduling competition between two airlines.

The discounting of fares in the airline industry for both international and domestic flights fosters active price competition. Although passengers can receive attractive discounts, they must book such tickets at least some weeks prior to departure. In this article, we extend our model, focusing especially on duplicate bookings. We examine how passengers decide whether to purchase a discounted ticket in advance; in particular, we discuss the problem of duplicate bookings (i.e. two or more flights are booked for the same traveller).

Duplicate bookings are prohibited by airlines, simply because they want to stop the wasteful occupation of seats. However, airlines usually sell their discounted tickets when passengers still have no idea of their travel plans. Furthermore, the number of seats to which this type of discounted fare is applied is limited. Thus, passengers who

Ryosuke Ishii
Teikyo University, Otsuka 359, Hachioji, Tokyo 192-0395, Japan.
E-mail: ryosuke.ishii@main.teikyo-u.ac.jp

Kuninori Nakagawa
Shizuoka University, Ohya 836, Suruga-ku, Shizuoka 422-8529, Japan.
E-mail: nakagawa.kuninori@shizuoka.ac.jp

want to take advantage of discounted airfares must reserve a seat a month before their flights or even earlier, providing them with an incentive to make several duplicate bookings. We analyse this duplicate booking problem. We find that flight scheduling by two airlines in a subgame perfect equilibrium is socially optimal if a duplicate booking is not prohibited.

In particular, we analyse the advance selling of airline tickets by duopolists, focusing on the relationship between scheduling competition and a passenger's uncertainty about his or her own travel plans. We examine a model in which two airlines 'locate' their flight schedules on a timeline and set their ticket prices twice: before and after a passenger knows precisely his or her travel plans (i.e. ex ante and ex post, respectively). We thus consider the extent to which the tradeoff between low prices and a schedule better suited for the final itinerary is affected by passengers' decision making under uncertainty.

This competition over the scheduling of flight departure times and prices fits the model of spatial competition à la Hotelling (1929)[11]. Hotelling's spatial approach is useful for analysing this type of competition because in an airline market, the cost to a passenger of taking a certain flight is the ticket price and the cost to a passenger of adapting his or her travel plans is the flight departure time. However, a spatial approach is too complicated to provide clear implications for practitioners. Hence, Brueckner and Flores-Fillol (2007) [3] instead suggest that passengers are concerned about overall flight frequency rather than the departure times of individual flights (see also Brueckner, 2010, [4]).

Spatial models of product differentiation imply that firms face two opposing incentives: (1) counter differentiate in order to take customers from competitors and (2) differentiate in order to reduce price competition. For example, Greenhut et al. (1987) [9] discuss airline scheduling in this context and suggest that more competition leads to less differentiation. Theoretical studies of Hotelling's spatial competition find that the dominance of these opposing incentives depends on the assumptions made (d'Aspremont et al., 1979, [1]). According to empirical studies, both results are supported. Borenstein and Netz (1999)[2], for instance, find that after the deregulation of the airline industry in the United States in 1986, reductions in exogenous scheduling constraints increased differentiation, while Salvanes et al. (2005)[16] show that after deregulation in Norway, the clustering of flights increased on duopoly routes compared with monopoly routes.

In addition to examining the scheduling of flight departure times, we analyse the discounts offered for early reservation. In the literature on advance selling, advance purchase discounts are discussed in terms of price discrimination by a monopoly firm. Gale and Holmes (1993)[8], Dana (1998)[7], Nocke and Peitz (2007)[14], Möller and Watanabe (2010)[13], and Nocke et al. (2011)[15], among others, focus on the role of demand uncertainty and capacity constraints in the price discrimination policies of a monopolist in order to understand the allocation of resources in a ticket market.

This situation is related to Coase's (1972)[6] conjecture. According to the literature on this conjecture, both ex-ante and ex-post markets will open only if the time inconsistency problem is solved such that 1) airlines commit to not having sales ex post or 2) passengers are so irrational that they do not consider ex post (see Bulow, 1982, [5], Stokey, 1981, [17], Gul et al., 1986, [10]). However, it is difficult to examine explicitly how these problems affect competition over the scheduling of flight departure times and prices. In the present study, we put forth the model that, without assumptions 1) and 2) above, both markets open in a subgame perfect equilibrium. Moreover, in a subgame perfect equilibrium, we show that every flight operates on the most efficient interval

(i.e. where a passenger's expected loss of utility is minimized). Finally, we discuss the issue of duplicate bookings.

2 Model

We present our model based on that of Ishii and Nakagawa (2015)[12]. There are two airlines $i \in 1, 2$ and a passenger who plans to travel. The passenger does not know his or her most preferred boarding time t (here, we refer to this as the ideal point), but knows that t follows a uniform distribution over the interval $[0, 1]$. This closed interval also represents the airport's hours of operation. In Japan, for example, airport runways are usually closed at night to prevent aircraft noise from disturbing local residents (e.g. Osaka International Airport operates from 7:00 to 21:00).

This game proceeds as follows. First, the airlines simultaneously decide their flight schedules: airline $i(i = 1, 2)$'s departure time is $z_i(0 \leq z_1 < z_2 \leq 1)$. Second, they set the ex-ante prices p_i^b of their early discounted tickets. Third, the passenger has an opportunity to purchase the tickets. Fourth, the passenger learns his or her ideal points. Fifth, the airlines set the ex-post prices p_i^a of their normal tickets. Sixth, the passenger has a second opportunity to purchase tickets. Lastly, the passenger chooses a ticket to use.

Let (D^b, D^a) denote demand for the tickets that the passenger purchases before and after his or her ideal points are known. For example, $(D^b, D^a) = (0, 1)$ means that the passenger purchases no ticket ex ante and one ticket ex post. We assume that the passenger lexicographically prefers $(D^b, D^a) = (1, \cdot)$ to $(D^b, D^a) = (2, \cdot)$ when they are indifferent in terms of utility among these purchasing behaviours. We call this assumption 'inhibited duplicate bookings'. Because, in our model, the passenger finally consumes at most either one of these tickets, $(D^b, D^a) = (2, \cdot)$ means the presence of a duplicate booking when an advance purchase is made. Usually, when a duplicate booking exists, the passenger will be forced to cancel all the reservations made with the airline. This lexicographic preference means that the passenger avoids this cancelation. Here $(D^b, D^a) = (1, 1)$ is not a duplicate booking since the passenger will reserve a new ticket after he or she cancels the old one.

Let $\bar{u}(= const)$ be the passenger's utility when he or she uses his or her ideal tickets $z = t$. We define the passenger's utility u as $\bar{u} - (t - z_i)^2 - (\text{money paid})$, if airline i's ticket is used. $E[(t - z_i)^2]$ is the passenger's expected utility loss. We assume that $\bar{u}$ is large enough that the passenger purchases at least one ticket. Now, we can rewrite the passenger's utility maximization problem as a cost minimization problem. Here, his or her cost is the total cost of the tickets and the expected utility loss. Let π_i denote the profit of airline i. $\pi_i = p_i$ if purchased and 0 otherwise.

We solve this game by backward induction. We will show that with respect to all the passenger's behaviours in subgame perfect equilibrium outcomes, the airlines' flight schedules are symmetric and they set identical ex-ante prices. Both the passenger and the airlines are indifferent with respect to the passenger's choice between 'purchase' and 'not purchase' in the ex-ante period. Furthermore, the airlines' flight schedules in all equilibria are socially optimal in the sense that the passenger's expected utility loss $E[\min(t - z_i)^2]$ is minimized.

3 Price Game

3.1 Competition Ex Post

If the passenger does not purchase a ticket in advance, both airlines purely can compete on prices and Bertrand competition in ex-post markets results. If the passenger buys one ticket in advance whose departure time better fits his or her ideal travel plan, he or she will not purchase any more tickets. On the contrary, if the passenger has a ticket that is further from the realized ideal point, he or she will purchase the ticket whose departure time better matches his or her ideal. Therefore, the airline whose flight schedule matches the passenger's ideal departure time more closely will win the passenger's business. Furthermore, we find that the passenger purchases at most one ticket ex post.

Hereinafter, we analyse the case in which the passenger decides whether to purchase one or zero tickets after knowing his or her ideal departure time. First, we consider that the passenger does not purchase a ticket in advance. In this case, the passenger purchases one ticket after knowing his or her ideal points. Both airlines can purely compete on prices and Bertrand competition results. Airline i, whose departure time is closer to the passenger's ideal point than its opponent $j(j \neq i)$, sets its price as $p_i = p_j + (t - z_j)^2 - (t - z_i)^2$, while the other airline j sets $p_j = 0$. The passenger purchases airline i's ticket.

Second, we consider that the passenger has already purchased airline j's ex-ante ticket and examine whether he or she purchases an additional ticket from airline i after knowing his or her ideal departure time. If the realized ideal point t is close to airline j's flight schedule compared with airline i's schedule, there is no room for airline i to sell its ticket to the passenger. If this is not the case, then the passenger prefers airline i's departure time to that of airline j and purchases airline i's ticket at a competitive price. The passenger is willing to purchase airline i's ticket when the amount of utility loss and repayment is less than or equal to the utility loss when he or she uses airline j's ticket, i.e. $\bar{u} - p_j^b - p_i^a - (t - z_i)^2 \geq \bar{u} - p_j^b - (t - z_j)^2$. The optimal pricing for airline i is $p_i^a = (t - z_j)^2 - (t - z_i)^2$. The passenger purchases airline i's ticket.

Suppose that the realized t is closer to airline 2's flight schedule than airline 1. In any case of $D^b = 0, 1$, the passenger purchases and uses airline 2's ticket, whose ex-post price is $p_2^a = (t-z_1)^2-(t-z_2)^2$. By solving t in this equation, we obtain $t = \frac{z_1+z_2}{2} + \frac{p_2^a}{2(z_2-z_1)}$. We find from this that $t = \frac{z_1+z_2}{2}$ if and only if $p_2^a = 0$, which is the net price of airline 2's ticket without considering the transportation cost $(t - z_1)^2 - (t - z_2)^2$, which is determined according to the realized t. If $t = \frac{z_1+z_2}{2}$ is realized, then the effect of location is perfectly offset. Without location competition, this ex-post competition is the simple Bertrand type, which results in $p_1^a = 0, p_2^a = 0$. If $D^b = 0$, and airline 1 chooses $p_1^a > 0$, the passenger will purchase an airline 2's ticket for whatever p_1^a. Thus, airline 1's profit would be zero. Airline 1 has an incentive to undercut its ex-post price in order to make the passenger purchase airline 1's ticket, and thus improves its profit from zero to nonzero. Thus, airline 1 must choose $p_1^a = 0$ in the equilibrium. By contrast, given $p_1^a = 0$ and the realized t, airline 2 still has the margin of transportation cost $(t-z_1)^2-(t-z_2)^2$ by which to secure a positive profit. By using a similar argument, the same result can be obtained for airline 2's problem.

Therefore, we obtain the equilibrium price pair of the two airlines, as follows: $p_i^{a*} = (t - z_j)^2 - (t - z_i)^2, p_j^{a*} = 0$. Here, i and j denote airlines with schedules near to and

far from the realized ideal point of the passenger, respectively. In this equilibrium, the passenger purchases airline i's ticket.

3.2 Competition Ex Ante

In this section, we consider the ex-ante equilibrium prices p_1^b and p_2^b, and the passenger's behaviour D^b. Here, we obtain two kinds of equilibrium demand outcomes, $D^b = 0, 1$, while the equilibrium profits of both airlines are uniquely determined regardless of the passenger's behaviour.

First, we work with the condition that the passenger is indifferent between $D^b = 2$ and $D^b = 1$. Now, the passenger's expected utilities if $D^b = 0, 1, 2$ are

$$\bar{u} - \left(\int_0^{\frac{z_1+z_2}{2}} (t - z_2)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t - z_1)^2 dt \right), \tag{1}$$

$$\bar{u} - \left(\int_0^1 (t - z_i)^2 dt + p_i^b \right), \tag{2}$$

$$\bar{u} - \left(\int_0^{\frac{z_1+z_2}{2}} (t - z_1)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t - z_2)^2 dt + p_1^b + p_2^b \right), \tag{3}$$

respectively. Suppose that both airlines charge prices p_1^b, p_2^b, which is the price pair such that (2) = (3) holds for any i:

$$\int_0^1 (t - z_i)^2 dt + p_i^b = \int_0^{\frac{z_1+z_2}{2}} (t - z_1)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t - z_2)^2 dt + p_1^b + p_2^b,$$

where the passenger is only indifferent between $D^b = 2$ and $D^b = 1$. We rearrange these expressions with p_1^b, p_2^b to obtain

$$(p_1^*, p_2^*) = \left((z_2 - z_1)(\frac{z_1 + z_2}{2})^2, (z_2 - z_1)(1 - \frac{z_1 + z_2}{2})^2 \right). \tag{4}$$

The airline's profits are $\pi_1^* = (z_2 - z_1)(\frac{z_1+z_2}{2})^2$ and $\pi_2^* = (z_2 - z_1)(1 - \frac{z_1+z_2}{2})^2$, respectively if the passenger purchases either/both tickets at this price.

Next, we consider $D^b = 0$. Here, suppose that both airlines charge the ex-ante price pair shown in (4), and that the passenger does not purchase a ticket. From the discussion on the ex-post period in the previous section, the airlines' expected profits are

$$\pi_1^* = \int_0^{\frac{z_1+z_2}{2}} \left((t - z_2)^2 - (t - z_1)^2 \right) dt = (z_2 - z_1)(\frac{z_1 + z_2}{2})^2,$$

$$\pi_2^* = \int_{\frac{z_1+z_2}{2}}^1 \left((t - z_1)^2 - (t - z_2)^2 \right) dt = (z_2 - z_1)(1 - \frac{z_1 + z_2}{2})^2.$$

Thus, we find that neither airline has an incentive to deviate from the price pair in (4) in any cases of $D^b = 0, 1, 2$.

Next, we find that if these ex-ante prices are sufficiently high such that (1) $\geq$ (2) and (1) $\geq$ (3) hold:

$$\int_0^{\frac{z_1+z_2}{2}} (t-z_2)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t-z_1)^2 dt \leq \int_0^1 (t-z_i)^2 dt + p_i^b, \tag{5}$$

$$\int_0^{\frac{z_1+z_2}{2}} (t-z_2)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t-z_1)^2 dt \leq \int_0^{\frac{z_1+z_2}{2}} (t-z_1)^2 dt + \int_{\frac{z_1+z_2}{2}}^1 (t-z_2)^2 dt + p_1^b + p_2^b, \tag{6}$$

then the passenger has no incentive to deviate to purchase one or two tickets. Note that if (5) holds for each $i = 1, 2$, (6) also holds. Thus, we find that ex-ante ticket prices with $D^b = 0$ are $p_1^* \geq (z_2 - z_1)(\frac{z_1+z_2}{2})^2$ and $p_2^* \geq (z_2 - z_1)(1 - \frac{z_1+z_2}{2})^2$. Hence, the price pair shown in (4) is the airlines' best response to the passenger's behaviour.

Finally, we also find that $D^b = 0, 1, 2$ are the passenger's best responses to the price pair shown in (4). Thus, we conclude that this price pair is the best response of the airlines and that $D^b = 0, 1, 2$ are the passenger's best responses to this price pair.

We obtain the equilibrium price pair of the two airlines as follows: $(p_1^*, p_2^*) = \left((z_2 - z_1)(\frac{z_1+z_2}{2})^2, (z_2 - z_1)(1 - \frac{z_1+z_2}{2})^2\right)$. However, owing to the issue of inhibited duplicate bookings, the passenger does not purchase two tickets at the same time in advance. Thus the equilibrium purchasing behaviours of the passenger at the ex-ante price subgame are purchasing one ticket, $D^{b*} = 1$, or not purchasing any ticket, $D^{b*} = 0$.

4 Location Game

In this section, we solve the first stage of the game, which we call the 'location game'.

Proposition 1 *In a subgame perfect equilibrium, the following holds:*

$$z_1^* = \frac{1}{4}, z_2^* = \frac{3}{4}, \textit{and } \pi_1^* = \pi_2^* = \frac{1}{8}.$$

Proof In a subgame perfect equilibrium, the airlines' profits are $\pi_1^* = (z_2 - z_1)(\frac{z_1+z_2}{2})^2$ and $\pi_2^* = (z_2 - z_1)(1 - \frac{z_1+z_2}{2})^2$, respectively. The first-order conditions with respect to z_1 and z_2 provide the desired result. These values satisfy the second-order condition.

5 Concluding Remarks

In this study, we considered the relationship between a passenger's behaviour and competition over flight schedules and prices between two airlines operating in a duopoly. The analysis presented herein confirms that a socially optimal flight schedule results from the passenger's behaviour in response to these competitive conditions. Proposition 1 shows that in a subgame perfect equilibrium, the flight schedule is the socially optimal location, where the flights are at the first and third quartiles of the $[0, 1]$ interval.

The presented model could be extended by focusing on the number of airlines and flights operated by each airline. In terms of the former, researchers might consider an extension to the $n \geq 3$ airlines case. In terms of the latter, we attempted to calculate the case in which each airline operates two flights in our two-airline model; however,

the problem is complicated and cannot be solved analytically. Therefore, these two extensions remain as future work.

Lastly, we discuss 'inhibited duplicate bookings' in our model. Our analysis herein deals with the passenger's lexicographic preference in order to capture the idea that he or she decides against a duplicate booking. We find that this assumption is mild enough to keep the equilibrium in which the passenger purchases an early and thus discounted ticket. Supposing that the passenger can buy at most one ticket, for example, destabilizes such early purchasing behaviour. Intuitively, if the passenger purchases an early ticket, the airline that sells no tickets has an incentive to undercut its early price in order to make the passenger purchase, meaning that the earlier competition is intensified compared with the case in our model. At this point, the other airline that sells an advance ticket becomes worse off when offering discounts and thereby has an incentive not to sell in advance, which intensifies the price competition ex ante. As a result, an airline locates at the centre and sells its early ticket at a low price in a subgame perfect equilibrium. For more details, see Ishii and Nakagawa (2015)[12]. By contrast, our model has an equilibrium outcome with advance purchasing. In the first place, purchasing more than one ticket is not forbidden by law. In a sense, the advance purchasing outcome that assumes the lexicographic preference corresponds to real-life cases where people sometimes cancel and repurchase tickets at the last minute.

Acknowledgements We are deeply grateful to Haruo Imai for his insightful comments and suggestions. We also acknowledge the valuable comments and suggestions of the anonymous referees. All errors are the authors' own.

References

1. d'Aspremont, C., Gabszewicz, J.J. and Thisse, J.F., On Hotelling's "stability in competition" Econometrica, Vol. 47(5), pp. 1145-1151, (1979).
2. Borenstein, S. and Netz, J., Why do all the flights leave at 8 am?: Competition and departure-time differentiation in airline markets, International Journal of Industrial Organization, Vol. 17(5), pp. 611-640, (1999).
3. Brueckner, J.K. and Flores-Fillol, R., Airline schedule competition, Review of Industrial Organization, Vol. 30(3), pp. 161-177, (2007).
4. Brueckner, J.K., Schedule competition revisited, Journal of Transport Economics and Policy, Vol. 44(3), pp. 261-285, (2010).
5. Bulow, J.I., Durable-Goods Monopolists, Journal of Political Economy, Vol. 90, pp. 314-332, (1982).
6. Coase, R., Durability and Monopoly, Journal of Law and Economics, Vol. 15(1), pp. 143-49, (1972).
7. Dana Jr., J.D., Advance-purchase discounts and price discrimination in competitive markets, Journal of Political Economy, Vol. 106(2), pp. 395-422, (1998).
8. Gale, I.L. and Holmes, T.J., Advance-purchase discounts and monopoly allocation of capacity, American Economic Review, Vol. 83(1), pp. 135-146, (1993).
9. Greenhut, M.L., Norman, G., Hung, C.-S., The economics of imperfect competition: A spatial approach, Cambridge University Press, Cambridge, (1987).
10. Gul, F., Sonnenschein, H. and R. Wilson, Foundations of dynamic monopoly and the Coase conjecture, Journal of Economic Theory, Vol. 39(1), pp. 155-190, (1986).
11. Hotelling, H., Stability in competition, Economic Journal, Vol. 39, pp. 41-57, (1929).
12. Ishii, R. and Nakagawa, K., Early competition on Discount Tickets, Journal of Transport Economics and Policy, Vol. 49(2), pp. 219-235, (2015).
13. Möller, M. and Watanabe, M., Advance purchase discounts versus clearance sales, Economic Journal, Vol. 120, pp. 1125-1148, (2010).
14. Nocke, V. and Peitz, M., A theory of clearance sales, Economic Journal, Vol. 117, pp. 964-990, (2007).

15. Nocke, V., Peitz, M. and Rosar, F., Advance-purchase discounts as a price discrimination device, Journal of Economic Theory, Vol. 146(1), pp. 141-162, (2011).
16. Salvanes, K.G., Steen, F. and Sørgard, L., Hotelling in the air?: Flight departures in Norway, Regional Science and Urban Economics, Vol. 35(2), pp. 193-213, (2005).
17. Stokey, N.L., Rational expectations and durable goods pricing, Bell Journal of Economics, Vol. 12, pp. 112-128, (1981).

MISTA 2015

Continuous Time Model for Scheduling Operations in Cascaded Continuous Processing Units with Multiple Due Dates

Subir Bhattacharya • Sumit Kumar Bose

Abstract Continuous processing units, where input streams flow in at one end and the output streams flow out simultaneously at the other end, are quite common in petroleum and pharmaceutical industries. In a typical scenario each product needs to be processed by a specific sequence of units, and there can be several products requiring the same sequence of units. Each product may need to be shipped multiple times, in specified quantities, during the planning horizon. However, each unit can process only one product line at a time. Since the input streams for all the product lines flow into the block of units simultaneously, scheduling operations in these units calls for a balance among spillage penalties, changeover costs and shipment failure penalties. This paper uses the concepts of State Task Network and event points to develop a continuous time model with a goal to minimize the total cost. The model has been tested in a scenario having three units and three product lines as encountered in a refinery situation.

1 Introduction

In chemical processing industries the input materials, often in the form of fluid streams, need to be processed through a sequence of processing units before being converted to marketable finished products. If it is a continuous processing industry, input streams for a product line are fed in continuously at one end of a continuous processing unit, a fractionating column for example, and the output streams flow out simultaneously from the other end. Some of these processing units may be shared by more than one product line. These shared units work in 'blocked-out' fashion where, at any point in time, a unit can process only one product line, and the intermediate streams coming from upstream units along the other product lines have to wait on the input side of the unit in dedicated fixed capacity storage tanks. If the available free space in the storage tank of a waiting input stream is not enough, the incoming

---
Subir Bhattacharya
Indian Institute of Management Calcutta
E-mail: subir@iimcal.ac.in

Sumit Kumar Bose
Indian Institute of Management Calcutta
E-mail: bose_sumit@rediffmail.com

stream has to be 'spilled', i.e., converted to lower valued products, thus incurring an opportunity cost. Spillage can be reduced by quick changeovers, but that has its own associated cost and time. The problem of scheduling operations in these blocked-out units is accentuated by the presence of pre-specified shipment schedule for finished products. A product may be shipped several times by specified quantities during the scheduling horizon. A shipment failure penalty is incurred if the specified amount of a product is not made available on the due date. Thus, an optimal schedule for the units calls for a trade-off among spillage penalties, shipment failure penalties, and changeover costs. Such a balance would require each unit to process each product line in a number of stretches of possibly different durations, interleaved with stretches of other product lines. This is a typical short term scheduling problem in continuous processing units, and is different from the short term scheduling problem typically encountered in batch processing plants.

The short-term discrete time scheduling problem for batch processing plants and its variations has been widely studied in literature. Comprehensive reviews of the work done in this area can be found in [7], [11], [18], and [24]. A number of continuous time formulations for batch process scheduling have also been suggested using the concept of event points. The locations of the event points on the time axis are not known *a priori* and are variables to be determined during the optimization process. Use of both unit specific event points ([13], [14]), and global event points ([29]) have been reported for the batch process scheduling problem. Mendez et al. ([25]) discusses the MILP continuous-time models for the batching and scheduling problem for multiproduct batch plants with multiple product orders and different due-dates. Schilling and Pantelides ([27]) propose a branch and bound algorithm for the problem formulated as MILP based on a continuous representation of time. Liu and Karimi ([21]) propose several novel continuous time models for multi-stage batch plants with identical units. Unit specific continuous time representations for short term scheduling of batch processes have been considered in [15] and [28]. Marchetti and Cerdá ([23]) provide a mathematical model for multi-stage batch plants with multiple intermediate due dates. A formulation for short term scheduling of batch processes based on a novel continuous time representation wherein the start times and the end times for any of the tasks do not necessarily have to align with the event points has been proposed by Giménez et al ([8], [9]). Hazaras et. al., [12], presents a mathematical model for the combined maintenance and production scheduling problem using a continuous-time representation and process network based on STN representation. Recently, a novel continuous time model for resource constrained project scheduling problem has been suggested by Kopanos et al. [20].

In contrast, interest in scheduling of operations in continuous processing units is rather recent. These units introduce an additional complexity compared to the batch processing units. While in batch processing units the time needed to process a particular batch is known *a priori*, in continuous processing units the time for which a product line should be processed in a unit in the current stretch is a decision variable. Jain and Grossmann ([16]) build a mathematical model for cyclic scheduling of continuous processing plants with parallel units and decaying performance. Alle et al. ([1]) has developed a continuous time mathematical model for cyclic scheduling of multi product multistage continuous processing plants. The short term scheduling of refinery operations and mixed integer models based on continuous time formulations have been discussed in [17]. Bose and Bhattacharya ([2]) discuss a discrete time mathematical model for scheduling in cascaded continuous processing units using state task network. When the planning horizon is longer than what the mathematical models can handle in reasonable time, the branch and bound heuristic algorithm discussed in [3] might be useful. Luo and Rong ([22]) propose decomposition approaches for solving the short term scheduling problem in refineries. Pochet and Warichet ([26]) consider the cyclic scheduling problem of mixed plants and propose a tighter continuous time formulation for maximizing productivity. Chen et. al. ([4]) provides a comparative study of continuous time models for the crude oil scheduling problem in refineries.

Evidently, a continuous time model for continuous processing units would be more desirable since the quality of solutions obtained from discrete time models depend on the size

of the time grain. This paper proposes a continuous time mathematical model for short term scheduling of operations in a set of cascaded continuous processing units, collectively responsible for a set of products. Each of the products may be scheduled to be shipped several times by specified amounts during the planning horizon. We use an extension of the concept of State Task Network (STN) as proposed by Kondili et al. ([19]), and the concept of event points to arrive at the formulation. The rest of the paper is organized as follows. Section 2 provides a generic description of the problem. Section 3 develops the continuous time mathematical model for the problem. Section 4 discusses our experiences with the model when applied to a refinery situation. Limitations of the proposed model and future directions of work are discussed in the concluding remarks in section 5.

2 The Problem

The scheduling problem discussed here is similar to the problem discussed in Bose and Bhattacharya, [2]. A generic description of the problem would be as follows. The problem, or variations of it, is quite common in continuous processing industries like refineries and pharmaceuticals.

A set of n continuous processing units are responsible for producing m final products. The input stream for each final product needs to be processed by a sequence of units, fixed for that product. Each of the inputs and intermediate streams has dedicated fixed capacity storage tanks before and after every processing unit. The presence of intermediate storage tanks obviates the need for the units to process the same product line in tandem. Figure 1 is an example of a setup with 3 units and 4 products.

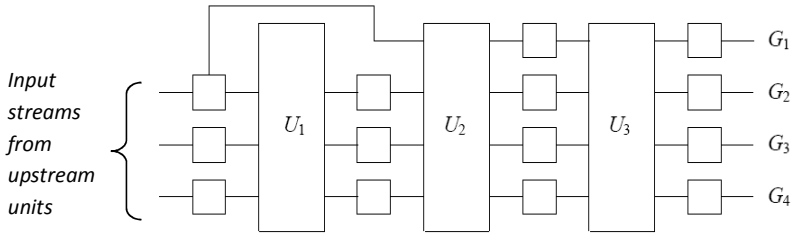


Figure 1: An example set up with 3 units and 4 products

Each of these n units can process only one product line at a time. However, the inputs corresponding to all the product lines to be processed by this set of units are arriving simultaneously and continuously at fixed rates from some upstream units that are responsible for many other products as well. Thus, at any particular point in time, the input for a product line that is not being processed by the first unit in the sequence at that point in time has to accumulate in the designated tank, and would spill if the corresponding tank does not have sufficient room. It may be noted that inputs to the subsequent units in the sequence need not be spilled since the immediately preceding unit processes only one product line at a time. The processing of a unit involves splitting the feed with the help of reagents into intermediate streams according to a yield percentage fixed for a product line for the unit. The intermediate streams that are relevant for the final products under consideration get deposited in the respective tanks on the output side. What happens to the other output fractions is beyond the purview of this discussion. The processing capacity (known as the feedrate measured in MtPD, Metric tons Per Day) of a unit for a product line is fixed, but varies from product to product.

To do justice to the amount of costly reagents added to a unit during a changeover, a stream taken up for processing in a unit must run for a minimum period of time, called *minimum run-length*. A processing unit switches processing from one product line to another for one or more of the following reasons: (a) there is not enough material in the input tank for

the current stream; (b) there is not enough room in the output tank for the current stream; (c) to avoid spillage of inputs; (d) to produce intermediate or final products for meeting the shipment target. Thus one product line can be processed in a unit in a number of stretches interleaved with the processing of other streams. The length of each such stretch, called a *run-length*, of a product line is a decision variable. Every changeover in a unit from one product line to another has its associated cost.

The finished products coming out of the last unit of a sequence also get stored in fixed capacity tanks to be shipped according to some pre-specified shipment schedule. There can be several planned shipments for a product during the scheduling horizon. Penalty is incurred if the required amount of a finished product is not ready by the specified due date. Shortfall in one shipment of a product cannot be compensated by providing more during the next shipment of the same product.

Thus, we have three factors to balance – the per unit spillage penalty for the input stream of each product line, the cost of changeovers, and the per unit penalty for failing to meet the shipment schedule. Schedules of the units collectively try to balance these factors by processing each product line in several run-lengths. Longer run-lengths are preferred to reduce changeover cost while shorter stretches may bring down the spillage penalty and/or the shipment failure penalty. Given the scenario and a scheduling horizon of H days, the aim is to determine the start times and end times of different stretches of different streams (product lines) in different units so that the total cost is minimized.

3 Model Formulation

The complexity of arriving at a continuous time model for the continuous processing units is contributed to by the difficulty in tracking resource violations, like an input tank getting empty or an output tank getting full. Tracking the duration of spillages or the shortfalls in shipment quantities add to this complexity. To develop the mathematical model we use the constructs of State Task Network ([19]) and introduce the concept of virtual tasks. State Task Network (STN) uses two constructs – '*state*' for denoting input (or output) feed to (or from) a processing activity, and '*task*' for denoting the processing activity. Thus, in essence, tasks consume and/or produce states. In the given scenario, states map to the materials stored in the tanks and the tasks correspond to the processing activities of the processing units. One processing unit is responsible for multiple tasks representing processing of different product lines in the unit. The advantage of using STN is that when the setup of the units and/or the product-flows change, it can be subsumed in the model in a straightforward manner by corresponding adjustments of the state and task sets.

We augment the STN by introducing two *virtual* tasks – virtual shipment task and virtual spillage task. These virtual tasks take place in virtual units that are fed by *states* in the STN but do not have output *states*. For each product line we define a virtual task that takes care of the possible spillage of the corresponding input state. Each of the virtual spillage tasks has a processing cost associated with it for processing in virtual units. This acts as a surrogate for the spillage cost. In a similar manner, we define a virtual shipment task for every scheduled shipment of every product. For the scheduling horizon under consideration, the number of virtual shipment tasks is known beforehand since the shipment schedule is available. These shipment tasks undergo processing in virtual units and have zero processing time. Each shipment task is scheduled when the corresponding shipment is due, and is supposed to process the respective input state by the amount specified in the schedule. The task incurs a cost only when it cannot process the amount specified. This acts as a surrogate for the shipment failure penalty. The difference between the two types of tasks, regular and virtual, is worth noting – the 'regular' tasks do not incur any processing cost, and take part in changeovers, whereas the virtual tasks are not involved in changeovers, and may incur processing cost.

To track resource violations we use the concept of *event points*. An '*event*' happens when something changes in the system – be it switching to a new task in a unit and/or taking up a

virtual shipment task. An '*event point*' is the point in time when an event happens. It is important to note that more than one event can happen at the same event point, for example two units can changeover to new product lines at the same point in time. Virtual spillage tasks are not considered as events in this formulation, since these tasks – and the corresponding costs – can be tracked from the mass balance equations of the corresponding input states between consecutive event points. In order to solve an instance of the scheduling problem, we specify the number of event points to be used, and allow the model to distribute these event points on the continuous time line so that the cost of the resultant schedule is optimal with that number of event points. Every schedule must utilize all the event points given as input. We solve the model repeatedly, each time allowing one more event point than the previous iteration. We continue the iterations till the objective value does not improve any further or till it satisfies a certain stopping criteria.

Table 1 lists the sets, parameters and variables used in our formulation. $Y_{i\eta}$ are binary decision variables. $Y_{i\eta}$ takes the value of 1 if task *i* is processed between event points η and η +1, otherwise it is 0. The variables $V_{j\eta}$ represent changeover events. The value of $V_{j\eta}$ is 1 if a task that gets processed in unit U_j at event point η is different from the task that is processed at event point $\eta+1$. $B_{i\eta}^{spill}$ is the amount of raw material spilled between event points η and η +1 by task *i*, $i \in I_{spill}$. B_i^{ship}, for $i \in I_o$, is the amount on which shipment penalty will be charged. To keep our formulation simple, without loss of generality, we consider the changeover cost to be sequence independent, and we ignore the changeover time. We also consider all units up and running for the entire planning period.

Table 1*: Notations for the mathematical formulation*

Sets
G : *Set of streams*
I : *Set of tasks*
E : *Set of states*
J : *Set of real units* (*excludes the virtual units that process shipment and spillage tasks*)
N : *Set of event points*, $\eta \in [\eta_1, ..., \eta_{last}]$
I^g : *Set of tasks associated with the stream* g, $I^g \subset I, g \in G$
I_j : *Set of tasks that can be performed in unit* j, $I_j \subset I,\ j \in J$
I_e^p : *Set of tasks that produce state* e, $I_e^p \subset I, e \in E$
I_e^c : *Set of tasks that consume state* e, $I_e^c \subset I, e \in E$
I_{spill} : *Set of spillage tasks*, I_{spill}
I_o : *Set of virtual shipment tasks*, $I_o \subset I$
E^g : *Set of states associated with stream* g, $g \in G$.
E_{inp} : *Set of input states*, $E_{inp} \in E$
E_{int} : *Set of intermediate states*, $E_{int} \in E$
E_{fin} : *Set of final states*, $E_{fin} \in E$

Parameters

r_i : *Per unit processing cost of task* $i, i \in I_{spill}$

a_e : *Rate of arrival of state* $e, e \in E_{inp}$

C_e : *Capacity of tank corresponding to state* $e, e \in E$

q_j : *Cost of changeover from one stream to another in unit* $j,\ j \in J$

$duedt_i$: *Date at which shipment task* $i, i \in I_o$ *is due*

d_i : *Amount required to be processed by the shipment task* $i, i \in I_o$ *at* $duedt_i$

u_i : *Per unit penalty of the amount by which task* $i, i \in I_o$, *fails to meet the target* d_i

p_i : *Percentage yield of task* $i, i \notin I_{spill}$ *and* $i \notin I_o$

f_i : *Feed rate of task* $i, i \notin I_{spill}$ *and* $i \notin I_o$

b : *Minimum runlength*

Variables

$$Z_{i\eta} = \left\{ \begin{array}{l} 1 \ \textit{if shipment task } i, i \in I_o \textit{ takes place at event point } \eta \\ 0 \ \textit{otherwise} \end{array} \right\}$$

$S_{e\eta}$: *Material available in state e at the beginning of event point* η

T_η : *Time at which event point* η *occurs*

$$Y_{i\eta} = \left\{ \begin{array}{l} 1 \textit{ if task } i \textit{ is processed between event points } \eta \textit{ and } \eta+1, i \notin I_{spill}, I_o \\ 0 \textit{ otherwise} \end{array} \right\}$$

$$V_{j\eta} = \left\{ \begin{array}{ll} 1 & \textit{if } Y_{i\eta} = 1 \textit{ and } Y_{i,\eta+1} = 0, i \in I_j \\ 0 & \textit{otherwise} \end{array} \right\}$$

$B_{i\eta}^{spill}$ = *Amount processed by task* $i, i \in I_{spill}$, *between event point* η *and* $\eta+1$

B_i^{ship} = *Amount by which task* $i, i \in I_o$, *fails to complete its target* d_i

$\mathrm{D}_{i\eta}$ = *The duration for which task i,* $i \notin I_{spill}$ *and* $i \notin I_o$, *is processed between* η *and* $\eta+1$

3.1 Constraints

3.1.1 Constraints for E_{inp} states

Input streams for all the product lines flow in simultaneously, and wait in the designated fixed capacity tanks for the corresponding states. If a tank for an input state fills up, the excess amount spills. The stock in the tank corresponding to the states $e \in E_{inp}$ at the start of the event point $\eta+1$ is equal to the stock at the start of event point η adjusted by the amount arriving between event points η and $\eta+1$ minus the withdrawal that takes place in case the state is consumed by a task or is spilled during the period between event points η and $\eta+1$. The expression for updating stock of input states is given by:

$$S_{e,\eta+1} = S_{e\eta} + a_e(T_{\eta+1} - T_\eta) - \sum_{\substack{i \in I_e^c, i \notin I_o, \\ i \notin I_{spill}}} f_i \mathrm{D}_{i\eta} - \sum_{i \in (I_{spill} \cap I_e^c)} B_{i\eta}^{spill} \qquad \forall e \in E_{inp}, \eta \qquad (1)$$

For feasibility reasons, federate f_i is always much higher than arrival a_e for all tasks *i*, $i \in I_e^c$ and $e \in E_{inp}$. Thus, spillage and processing of a state $e \in E_{inp}$ cannot take place simultaneously. Hence, $\mathrm{D}_{i\eta} > 0$ and $B_{i\eta}^{spill} > 0$ cannot be simultaneously true. If state $e \in E_{inp}$ is processed in the first unit of the relevant product line between event point η and η +1 ($Y_{i\eta}$ = 1 i.e. $\mathrm{D}_{i\eta}$ is positive; see equation (11)), then the equation reduces to $S_{e,\eta+1} = S_{e\eta} + a_e(T_{\eta+1} - T_\eta) - f_i \mathrm{D}_{i\eta}$, where $i \notin I_{spill}$ and $i \in I_e^c$. If state *e* spills during the period between event points η and $\eta+1$, then the equation reduces to $S_{e,\eta+1} = S_{e\eta} + a_e(T_{\eta+1} - T_\eta) - B_{i\eta}^{spill}$, where $i \in I_{spill}$ and $i \in I_e^c$. Since $S_{e\eta}$ has to be less than or equal to the corresponding capacity C_e (see equation (4)), the variable $B_{i\eta}^{spill}$ will assume a positive value, and since task $i \in I_{spill}$ has a positive processing cost, $S_{e,\eta+1}$ will be forced to assume the value of C_e. If there is no spillage from the tank corresponding to state *e* between event points η and $\eta+1$, and if state *e* is also not processed by the unit between event point η and $\eta+1$ ($Y_{i\eta}$ = 0 i.e. $\mathrm{D}_{i\eta}$ = 0), then equation (1) reduces to $S_{e,\eta+1} = S_{e\eta} + a_e(T_{\eta+1} - T_\eta)$ indicating an increase in the stock position of *e* in the tank by an amount $a_e(T_{\eta+1} - T_\eta)$.

3.1.2 Constraints for E_{int} states

The stocks in the intermediate states can be updated using the following constraint:

$$S_{e,\eta+1} = S_{e\eta} + \sum_{i \in I_e^p} p_i f_i \mathrm{D}_{i\eta} - \sum_{i \in I_e^c} f_i \mathrm{D}_{i\eta} \qquad \forall e \in E_{int}, \eta \qquad (2)$$

The stock at the start of event point $\eta+1$, $S_{e,\eta+1}$, depends on the opening stock at the start of event point η, buildup of stock by outflow $p_i f_i \mathrm{D}_{i\eta}$ (where $i \in I_e^p$) from the preceding unit if any, and withdrawal of stock $f_i \mathrm{D}_{i\eta}$ (where $i \in I_e^c$) by the next unit if any.

3.1.3 Constraints for E_{fin} states

Stocks at the finished product states need to be updated by a separate expression to account for the shipments and the possible shortfalls.

$$S_{e,\eta+1} = S_{e\eta} + \sum_{i \in I_e^p} p_i f_i \mathrm{D}_{i\eta} - \sum_{i \in (I_o \cap I_e^c)} (d_i - B_i^{ship}) Z_{i,\eta+1} \qquad \forall e \in E_{fin}, \eta \qquad (3)$$

Stock for state *e*, $e \in E_{fin}$, at event point $\eta+1$ is the stock that was available for that state at event point η plus the production, if any, of the state between η and $\eta+1$ minus any shipment that takes place at $\eta+1$. In the last term at most one $Z_{i,\eta+1}$ can be 1 (see equation (12)). B_i^{ship} is the shortfall, if any, of the targeted amount d_i of state *e*.

In this context, it would be worthwhile to note that shipments scheduled on a day are assumed to take place at the start of the day, *i.e.*, the production in the last unit of a product line on the day of shipment is not available for shipment.

3.1.4 Capacity Constraints

The stock in any tank at any point of time cannot exceed its capacity and hence

$$0 \le S_{e\eta} \le C_e \qquad \forall e, \eta \qquad (4)$$

3.1.5 Changeover Constraints

At any point of time, a unit can changeover to a maximum of one other stream. Thus,

$$V_{j\eta} \le 1 \qquad \forall j, \eta \qquad (5)$$

The value of $V_{j\eta}$ should be 1 when there is a changeover in U_j. This requires a constraint of the following form:

$$V_{j,\eta+1} = 1 - \sum_{i \in I_j} Y_{i\eta} Y_{i,\eta+1} \qquad \forall \eta, j \qquad (6)$$

These constraints force $V_{j,\eta+1}$ to 1, when $Y_{i\eta}$ is 1 and $Y_{i,\eta+1}$ is 0 for $i \in I_j$.

3.1.6 Event Point Constraints

$$\sum_{i \in I_o} Z_{i\eta} + \sum_j V_{j\eta} \ge 1 \qquad \forall \eta \qquad (7)$$

Constraint (7) mathematically represents the definition of event point. Accordingly, the above constraint forces either a changeover in at least one unit at event point η or forces the date of a shipment to coincide with the time of an event point.

3.1.7 Minimum Run-length Constraints

The minimum run-length is given by the following equation:

$$T_{\eta_1} V_{j\eta_1} - T_\eta V_{j\eta} \ge b V_{j\eta_1} V_{j\eta} - M(1 - V_{j\eta_1}) V_{j\eta} \qquad \forall j, \eta, \eta_1 > \eta \qquad (8)$$

Through this constraint we ensure that any two changeovers in unit U_j will have a gap of at least b periods. For every combination of $V_{j\eta} = 1$ and $V_{j\eta_1} = 1$, $\eta_1 > \eta$ the constraint reduces to $T_{\eta_1} - T_\eta \ge b$. The term $M(1 - V_{j\eta_1})V_{j\eta}$ is needed to account for the situation when $V_{j\eta} = 1$ and $V_{j\eta_1} = 0$.

It must be noted that the event points are non-overlapping. To ensure that, we add the following constraint:

$$T_{\eta+1} - T_\eta \ge \delta \qquad \forall \eta \qquad (9)$$

Where δ is any small positive quantity.

3.1.8 Unit Allocation Constraints

Exactly one task can be assigned to unit *j* for processing during event point η. So

$$\sum_{i \in I_j} Y_{i\eta} = 1 \qquad \forall \eta \qquad (10)$$

It may be noted that had we allowed shutdown of the units, then the equality constraint would have to be replaced by the inequality (≤) constraint.

3.1.9 Duration Constraints

$$(T_{\eta+1} - T_{\eta})Y_{i\eta} = \mathrm{D}_{i\eta} \qquad \forall i,\eta \qquad (11)$$

If $Y_{i\eta} = 1$, then the duration for which task *i*, $i \notin I_o$ and $i \notin I_{spill}$ is processed between event points η and $\eta+1$, $\mathrm{D}_{i\eta}$ is equal to $T_{\eta+1} - T_{\eta}$; otherwise $\mathrm{D}_{i\eta}$ is equal to 0.

3.1.10 Shipment Date Constraints

$Z_{i\eta}$ is 1 if the time T_{η} of event point η coincides with the due date of shipment task *i*, $i \in I_o$, otherwise it is 0. Further the time of no more than one event point should coincide with the due date of task $i \in I_o$. We achieve this by specifying the following set of constraints:

$$\begin{aligned} &(T_{\eta} - duedt_i)Z_{i\eta} = 0 && \forall i \in I_o, \eta \\ &\sum_{\eta} Z_{i\eta} = 1 && \forall i \in I_o \end{aligned} \qquad (12)$$

Please recall that the shipment tasks belonging to the set I_o do not require any processing time. They are assumed to be instantaneous.

3.1.11 Horizon Constraints

In our formulation we stipulate the first event point to occur at the *beginning* of the scheduling horizon i.e. $T_1 = 1$ and the last event point to be at the *end* of scheduling horizon. To ensure that the last event point actually marks the end of the scheduling horizon, $T_{last} = H$, the value of *H* fed into the model is one more than the length of the scheduling horizon. Thus,

$$\sum_{\eta=1}^{\eta_{last}-1} (T_{\eta+1} - T_{\eta}) = H - 1 \qquad (13)$$

Remaining event points get distributed within the scheduling horizon. Further, we add the following constraints:

$$T_{\eta} \le H \qquad \forall \eta \qquad (14)$$

$$T_{\eta} \ge 1 \qquad \forall \eta \qquad (15)$$

$$\sum_{\eta} \sum_{i \in I_j} \mathrm{D}_{i\eta} \le H - 1 \qquad \forall j \qquad (16)$$

To avoid possible resource violations in the next scheduling horizon, we stipulate that at the end of the scheduling period under consideration, the streams running in the units must have completed the minimum run-length.

3.2 Objective Function

The objective is to develop a schedule that would lead to least overall cost. The objective function to be minimized is given by:

$$\min \sum_{\eta} \sum_{i \in I_{spill}} r_i B_{i\eta}^{spill} + \sum_{\eta} \sum_{j} q_j V_{j\eta} + \sum_{i \in I_o \cap I_e^c} u_i B_i^{ship} \quad (17)$$

The first term gives the cost of spillage, the second term represents the cost of changeovers and the third term denotes the penalties for shipment failures.

The above formulation has nonlinear terms in some of the constraints. In most of the cases non linearity involve bilinear products of continuous and binary variables. Equation (6), however, involves bilinear product of two binary variables. Employing standard linearization techniques ([6], [10]) it is possible to remove all the non-linearity from the proposed formulation.

4 Computational Experience

The model was tried out in the Lube Block of a refinery in India. The Block has three processing units, and is responsible for three products. Each product needs to be processed by all the three units in the same sequence. Table 2 shows the fixed data collected from the refinery. The cost parameters have been masked, maintaining the relative importance of these parameters. The tank capacities are assumed to be 10000 Mt for all tanks for easy comparison of the input data for the cases considered.

Table 2: Fixed Data

Number of Units: 3 {U_1: FEU; U_2: SDU; U_3: HFU}
Number of Input states: 3 {1:IO; 2:HO; 3:DAO}
Number of Final states: 3 {1:IN; 2:HN; 3:BN}
Minimum Run Length *(b)*: 3 days
Capacity (Mt): 10000 (For all streams for all levels)
Changeover cost (q_j): 20 (Same for all *j*)

Stream *(g)*	Input Rate ($a_e, e \in E_{inp}$ & $e \in E^g$) (MtPD)	Feed rate of stream *g* in unit *j* (MtPD)			Yield Percentage of stream *g* in unit *j*			Spillage Penalty (r_i)	Shipment Penalty (u_i)
		FEU	SDU	HFU	FEU	SDU	HFU		
1	620	1250	780	546.0	60	70	99	10	14
2	255	1030	576	403.2	50	70	99	12	12
3	310	1050	521	364.7	65	70	99	14	10

The values of V_{j1} and $V_{j,\eta_{last}}$ were predefined to 1 for all *j*. The model, therefore, constrains the last run of the planning period in a unit to complete the minimum run-length. The model was solved using GAMS/XPRESS on NEOS ([5]).

To schedule operations in the units for a given scheduling horizon, we need to input (a) the number of event points to be used, (b) the schedule of shipments during the period under consideration, and (c) the stock positions in all the tanks at the start of day 1. The tanks before the first unit of the sequence are designated as level 0 tanks, and tanks after unit U_j are level *j* tanks. Thus, spillage is possible only from level 0 tanks and shipments take place from level 3 tanks.

To generate the test cases, we first chose a base case with a scheduling horizon of two weeks (H = 15) as given in Table 3. A close look at the input data would reveal that more than one stream would start spilling within first *b* (minimum run length) days of the scheduling horizon, if not processed. The Input Rate column (Table 2) along with the Level 0 stocks (Table 3) considered together, show that stream 1, steam 2, and stream 3, if not processed in U_1, would start spilling on day 6, day 3 and day 1 respectively. Since any stream picked up first for processing must run for at least 3 (*b*) days, some spillage is unavoidable. Again, given the stock positions at level 3 at the start of day 1, Unit 3 would require a total of 13.82 days of

processing to fully satisfy both the shipments. As discussed in section 3.1.3, shipments due on 14th must be ready by the end of day 13. Further, to meet the requirement of stream 1, Unit 3 has to process it only for one and half days ((required – available)/(U_3 feederate * yield percentage for stream 1)). But, if scheduled, stream 1 has to be run for 3 days to satisfy the minimum run length. Thus, there is bound to be some shipment failure penalties as well.

Table 3: Base case results with two weeks scheduling horizon

Case No.	Input Data								Solution details for different event points		
	Stock Position At start of day 1 (Mt)				Scheduled Shipments (Mt)						
	Levels				Day	Streams			η	Solution Cost	Time (Sec)
	0	1	2	3		1	2	3			
Base _H15	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	8 14	- 1800	2000 3000	2200 1000	5 6 7 8 9 10	19089.31 15929.50 9749.47 9729.50 9749.50 9769.47	0.006 1.217 4.010 7.318 26.007 46.622

An appropriate choice for the number of event points in a continuous time model is an unresolved issue. However, like in most other continuous time models, we also follow the approach of starting with a small number of event points, and then keep on solving the problem iteratively with one additional event point at each iteration. As can be seen from the results of the base case (Table 3), the solution kept on improving till 8 event points, and then started deteriorating with each additional event points. The time required to solve the MILP goes up with event points. Figures 3 and 4 are the Gantt Charts for the base case schedules with event points 8 and 9 respectively.

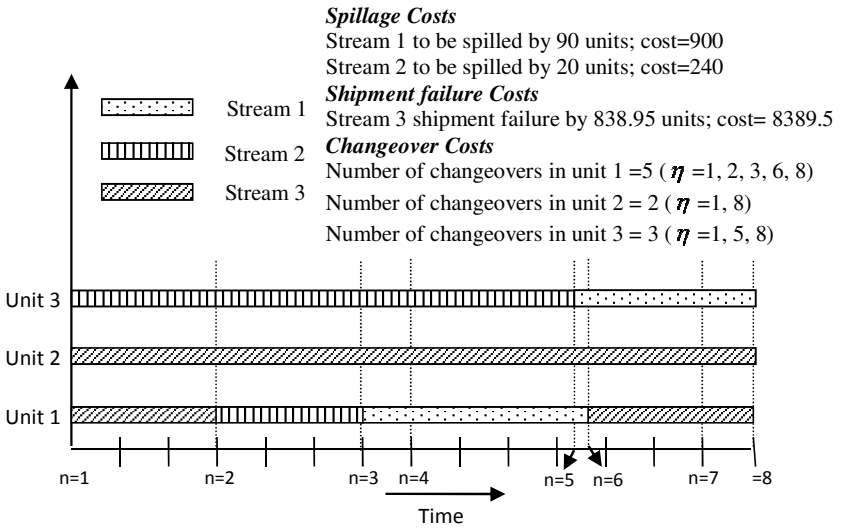


Figure 3: Gantt chart for the base case with η = 8

Every schedule must utilize all the event points given as input. With η = 8, two event points were at the beginning and at the end of the scheduling horizon, the last one being at the start of day 15. Changeovers in units took place at event points 2, 3, 5 and 6. Event points 4

and 7 flushed with the start of day 8 and day 14 respectively to enable the shipments due on those days. When we specify nine event points (Figure 4) no rearrangements could be found to improve the spillage and shipment penalties. The additional event point was accommodated in the schedule by processing stream 2 followed by stream 1 in U_2 instead of a continuous stretch of stream 3 when there were 8 event points.

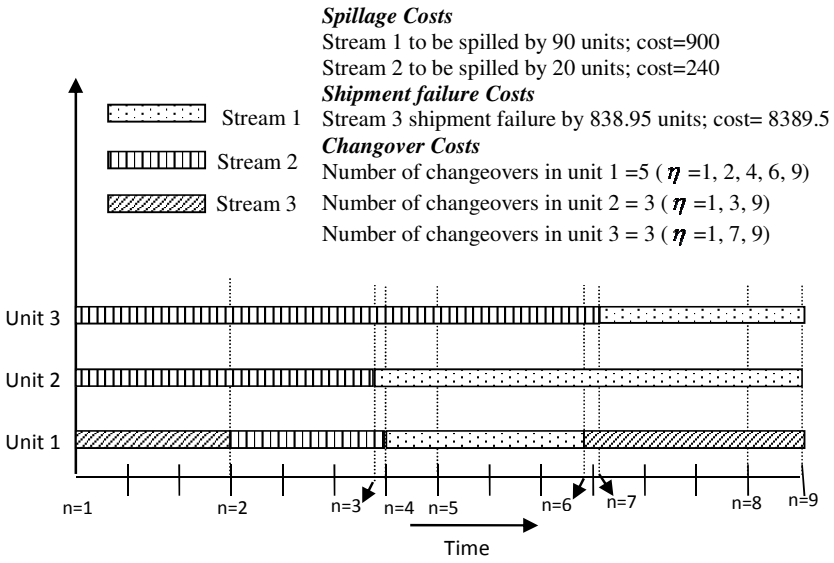


Figure 4: Gantt chart for the base case with η = 9

We generated six additional test cases by altering the initial stock positions and/or the shipment schedule of the base case. The first three cases have identical initial stock positions as the base case, and hence, some amount of spillage is unavoidable. These three cases differ in the shipment schedules both in number of shipments and the quantities to be shipped. In the last three cases initial stock positions are altered in a manner so that spillage is not inevitable. The only difference between case 5 and case 6 is that some additional 1000 units of stream 2 finished product is available in level 3 tank in case 5.

Table 4 shows the input data and the results of the cases with two weeks planning horizon.

A quick look at the Solution Costs (objective function value) would reveal that, in each case, as we increase the number of available event points, the objective function value improves till a particular number of event points, η*, and then it either worsens or remains unaltered. Are we hitting the optimal solution for the case at the corresponding η*? In cases 4 and 5,we can conclude to have reached the optimal solutions with six event points since, in both cases, all spillage and shipment penalties could be avoided, and the final cost is due to changeovers only. As we introduce more event points, since all the event points must be utilized, the solution gets be forced to introduce additional changeovers. In cases 1 and 2, just like the base case, both spillage and shipment shortfall are unavoidable. However, in case 3, since the shipment targets are less compared to cases 1 and 2, the model could avoid shipment shortfall. But, the Level 0 stock positions being the same, the spillage penalty incurred was the same. Similarly, in case 6, the spillage could be avoided as in case 5, but with lesser amount of stream 2 product at level 3 compared to case 5, some amount of shipment shortfall was inevitable. Thus, one is tempted to believe that we do hit the optimal solution at η*, though a formal proof of the same would be in order.

Table 4: Test case results with scheduling horizon of two weeks (H = 15)

Case No.	Stock Position At start of day 1 (Mt)				Scheduled Shipments (Mt)				Solution details for different event points		
	Levels				Day	Streams			η	Solution Cost	Time (Sec)
	0	1	2	3		1	2	3			
1	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	14	1500	5000	3140	5 6 7 8 9	17278.9 8360.0 8340.0 8360.0 8380.0	0.854 2.131 4.558 5.302 15.074
2	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	8 14	- 1500	3000 2000	2200 1000	5 6 7 8 9	22439.9 14540.0 8340.0 8340.0 8360.0	0.587 1.220 3.120 6.906 9.239
3	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	14	5000	3140	1500	5 6 7 8 9	12934.3 1400.0 1380.0 1380.0 1380.0	0.861 1.514 3.243 7.042 18.130
4	6370 8255 7810	8750 7424 8000	2000 2403 1635	1000 1000 2361	14	5900	-	2700	5 6 7 8 9	738.1 200.0 220.0 240.0 240.0	1.039 1.588 1.716 1.787 2.404
5	6370 6255 6810	3750 4424 5000	2000 5403 4635	1000 2000 2361	8 14	- 1500	3000 2000	2200 1000	5 6 7 8 9	2669.9 180.0 200.0 220.0 240.0	0.478 1.225 1.719 2.184 0.276
6	6370 6255 6810	3750 4424 5000	2000 5403 4635	1000 1000 2361	8 14	- 1500	3000 2000	2200 1000	5 6 7 8 9	7160.0 7160.0 7180.0 7200.0 7220.0	0.414 1.231 3.290 5.133 13.651

We tried out our model for longer planning horizons as well. With extended planning horizons, we need to provide more event points, and the model becomes sluggish with increase in event points. Table 5 gives the results of the base case for planning horizons of 21 and 30 days with one and two additional shipments, respectively. The model spends about seven hours with 14 event points in case of H = 22, and five hours with 13 event points in case of H = 31, and the solutions are still improving.

5 Concluding Remarks

The paper has developed a continuous time model based on global event points for a class of scheduling problems frequently encountered in continuous process plants. The model provides reasonably good solution within reasonable time for scheduling horizons of up to two weeks. Surely the model would help the short term planners with a tool to conduct what-if analyses for various possible shipment schedules.

The model needs to be further worked upon in two different counts. Firstly, it is only our conjecture that once the solution stabilizes (or worsens) with increase of event points, it cannot be improved further by allowing more event points. But it needs to be mathematically shown that allowing further event points would not improve the solution. Secondly, the model assumes sequence independent setup cost which is not a reality. The consequent complexity

can complicate the model further and make it run slower. The assumption of instant changeovers can be relaxed easily so long it is sequence independent.

Table 5: Scheduling with H= 22 and H = 31

Case No.	Stock Position At start of day 1 (Mt)				Scheduled Shipments (Mt)				Solution details for different event points		
	Levels				Day	Streams			η	Solution Cost	Time (Sec)
	0	1	2	3		1	2	3			
H22	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	8 14 21	- 1800 1900	2000 3000 1200	2200 1000 -	9 10 11 12 13 14	12281.8 10981.6 10309.3 8999.0 8969.1 8949.1	74.370 54.094 245.972 1547.440 3641.455 25680.458
H31	6370 9255 9810	3750 4424 5000	2000 5403 4635	1000 1000 2361	8 14 21 30	- 1800 1900 1650	2000 3000 - 1200	2200 1000 1500 1100	9 10 11 12 13	34798.4 25467.2 19279.6 17510.8 14394.3	52.030 921.651 1225.760 4181.931 17898.417

References

1. Alle, A., Papageorgiou, L. G., and Pinto, J. M., A mathematical programming approach for cyclic production and cleaning scheduling of multistage continuous plants, *Computers and Chemical Engineering*, *28*, 3–15 (2004).
2. Bose, S. K., and Bhattacharya, S., A state task network model for scheduling operations in cascaded continuous processing units, *Computers and Chemical Engineering*, *33*, 287–95 (2009).
3. Bose, S. K., and Bhattacharya, S., A two pass heuristic algorithm for scheduling 'Blocked Out' units in continuous processing industry, *Annals of Operations Research*, *159*(1), 293–313 (2008).
4. Chen, X., Grossmann, I., and Zheng, Li, A comparative study of continuous-time models for scheduling of crude oil operations in inland refineries, *Computers and Chemical Engineering*, 44, 141-167 (2012).
5. Czyzyk, J., Mesnier, M., and Moré, J., The NEOS Server, *IEEE Journal on Computational Science and Engineering*, *5*, 68–75 (1998).
6. Floudas, C. A., *Nonlinear and mixed-integer optimization*, Oxford University Press (1995).
7. Floudas, C. A., and Lin, X., Continuous-time versus discrete-time approaches for scheduling of chemical processes: A review, *Computers and Chemical Engineering*, *28*, 2109–29 (2004).
8. Giménez, D. M., Henning, G. P., and Maravelias, C. T., A novel network-based continuous-time representation for process scheduling: Part I. Main concepts and mathematical formulation, *Computers and Chemical Engineering*, *33(9)*, 1511-1528 (2009).
9. Giménez, D. M., Henning, G. P., and Maravelias, C. T., A novel network-based continuous-time representation for process scheduling: Part II. General framework, *Computers and Chemical Engineering*, *33(10)*, 1644-1660, (2009).
10. Glover, F., Improved linear integer programming formulations of nonlinear integer problems, *Management Science*, *22*, 455–60 (1975).
11. Grossmann, I. E., Quesada, J., Raman, R., and Voudouris, V., Mixed integer optimization techniques for the design and scheduling of batch processes, *Batch Processing Systems Engineering*. Springer, Berlin, 451–94 (1996).

12. Hazaras, M. J., Swartz, C. L. E., and Marlin, T. E., Flexible maintenance within a continuous-time state-task network framework, *Computers and Chemical Engineering*, *46*, 167-177 (2012).
13. Ierapetritou, M. G., and Floudas, C. A., Effective continuous-time formulation for short-term scheduling. 2. Continuous and semicontinuous processes, *Ind. Eng. Chem. Res.*, *37*, 4360–74 (1998).
14. Ierapetritou, M. G., Hene, T. S., and Floudas, C. A., Effective continuous-time formulation for short-term scheduling. 3. Multiple intermediate due dates, *Ind. Eng. Chem. Res.*, *38*, 3446–61 (1999).
15. Janak, S.L. and Floudas, C.A., Improving unit-specific event based continuous-time approaches for batch processes: Integrality gap and task splitting, *Computers and Chemical Engineering*, *32(4–5)*, 913-955 (2008).
16. Jain, V., and Grossmann, I. E., Cyclic scheduling of continuous parallel-process units with decaying performance, *AIChE Journal*, *44*, 1623–36 (1998).
17. Jia, Z., and Ierapetritou, M. G., Efficient short term scheduling of refinery operations based on continuous time formulation, *Computers and Chemical Engineering*, *28*, 1001–19 (2004).
18. Kallrath, J., Planning and scheduling in the process industry, *OR Spectrum*, *24*, 219–50 (2002).
19. Kondili, E., Pantelides, C. C., and Sargent, R. W. H., A general algorithm for short-term scheduling of batch operations - 1. Mixed integer linear programming formulation, *Computers and Chemical Engineering*, *17*, 211–27 (1993).
20. Kopanos, G. M., Kyriakidis, T. S., and Georgiadis, M. C., New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems, *Computers and Chemical Engineering*, *68*, 96-106 (2014).
21. Liu, Y. and Karimi, I.A., Novel continuous-time formulations for scheduling multi-stage batch plants with identical parallel units, *Computers and Chemical Engineering*, *31(12*), 1671-1693 (2007).
22. Luo, C. P., and Rong, G., Hierarchical approach for short-term scheduling in refineries, *Ind. Eng. Chem. Res. 46*, 3656–68 (2007).
23. Marchetti, P. A., and Cerdá, J., A general resource-constrained scheduling framework for multistage batch facilities with sequence-dependent changeovers, *Computers and Chemical Engineering, 33,* 871–86 (2009).
24. Mendez, C. A., Cerdá, J., Grossmann, I.E., Harjunkoski, I., and Fahl, M., State-of-the-art review of optimization methods for short-term scheduling of batch processes, *Computers and Chemical Engineering, 30,* 913–46 (2006).
25. Mendez, C., Henning, G. P., and Cerda, J., Optimal scheduling of batch plants satisfying multiple product orders with different due dates, *Computers and Chemical Engineering*, *24*, 2223–45 (2000).
26. Pochet, Y., and Warichet, F., A tighter continuous time formulation for the cyclic scheduling of a mixed plant, *Computers and Chemical Engineering*, *32(11)*, 2723-2744 (2008).
27. Schilling, G., and Pantelides, C. C., A simple continuous time process scheduling formulation and a novel solution algorithm, *Computers and Chemical Engineering*, *20*(suppl.), S1221–S1226 (1996).
28. Shaik, M.A., and Floudas, C.A., Unit-specific event-based continuous-time approach for short-term scheduling of batch plants using RTN framework, *Computers and Chemical Engineering*, *32(1–2)*, 260–274 (2008).
29. Wang, S., and Guignard, M., Redefining Event Variables for Efficient Modeling of Continuous-Time Batch Processing, *Annals of Operations Research, 116*, 113–26 (2002).

MISTA 2015

Scheduling optimization of a real flexible job shop including side constraints regarding maintenance, fixtures, and night shifts

Karin Thörnblad · Ann-Brith Strömberg · Michael Patriksson · Torgny Almgren

Abstract We present a generic iterative procedure for the scheduling of a real flexible job shop, the so-called multitask cell at GKN Aerospace Engine Systems in Sweden. A time-indexed formulation of the scheduling problem is presented—including side constraints regarding preventive maintenance, fixture availability, and unmanned night shifts. This paper continues the work in "Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance" [Thörnblad et al., European Journal of Industrial Engineering, to appear, 2015], with an improvement of the iterative solution procedure, and the inclusion of constraints regarding night shifts during which only unmanned processing is allowed to be scheduled. Schedules resulting from our procedure and from the use of two priority dispatching rules are compared. The gain of including the night shifts constraints is significant. Despite the added complexity, our methodology produces near-optimal schedules for industrial data instances for the coming shift within an acceptable practical time frame.

1 Introduction

The multitask production cell at GKN Aerospace Engine Systems, Sweden contains ten resources, five of which being multi-purpose machines capable of performing three types of processing tasks: turning, milling, and drilling. The problem of optimally scheduling this production cell is recognized as a *flexible job shop scheduling problem* (FJSP) [33].

Karin Thörnblad
GKN Aerospace Engine Systems
E-mail: karin.thornblad@gknaerospace.com

Ann-Brith Strömberg
Chalmers University of Technology and University of Gothenburg
E-mail: anstr@chalmers.se

Michael Patriksson
Chalmers University of Technology and University of Gothenburg
E-mail: mipat@chalmers.se

Torgny Almgren
GKN Aerospace Engine Systems
E-mail: torgny.almgren@gknaerospace.com

In [35] we propose an iterative solution procedure which produces near-optimal schedules for real instances of the FJSP modelling the multitask cell, including side constraints describing a limited fixture availability and required preventive maintenance (PM). This article improves the modelling as well as the iterative procedure developed in [35]. We introduce constraints regarding night shifts—during which only unmanned procesing is allowed to be scheduled—and further improve the iterative solution procedure. The resulting schedules are compared with schedules constructed using two priority dispatching rules, one of which is often utilized in practice, and the other is a built-in scheduling method in the control system of the multitask cell.

2 Literature review

In a *job shop* each job follows a predetermined sequence of operations [26, Chapter 2]. Each operation must be scheduled for processing in a designated machine during a predefined amount of processing time. A *flexible job shop* is a generalization of a job shop such that each operation may be scheduled in any of the machines in a given subset of the resources [7, Chapter 4].

Since the job shop scheduling problem (JSP), and hence also the FJSP, are NP-hard [6], these problems have gained interest from researchers ever since the late 1950's when [36], [4], and [21] proposed the first mathematical models for similar scheduling problems. Since the computation times required for solving JSPs and FJSPs by exact methods previously have been considered too long for practical purposes, a lot of research has focused on finding approximate solutions to these problems using heuristic methods [5].

Nowadays, due to the development of mathematical optimization theory and practice,—and of computer hard- and software—it is possible to find near-optimal schedules for real industrial instances using exact methods. In [35], we solve, within an acceptable time frame, a real FJSP for the coming work shift in the GKN multitask cell.

The objective that is most often utilized for scheduling problems is the minimization of the *makespan*, i.e., the time between the start of the first operation and the completion of the last operation of the schedule [17]. In [34], we noted a large difference in the performance of scheduling models depending on which objective was considered. Hence, it is of great importance that the evaluation of scheduling models are made with respect to objective functions that are well suited for the real applications modelled. In ibid. we argue that the minimization of the makespan is badly suited for a *dynamic environment*, in which the job shop needs to be repeatedly rescheduled; this is also the case in most real applications. Instead, we propose to minimize the weighted sum of the completion times and the total weighted tardiness, which—when the tardiness weight for each given job is a non-increasing function of its due date—has proven to work well in a dynamic environment. The rescheduling policy proposed is a *hybrid event-driven policy*, including a periodic rescheduling with respect to a rolling time horizon, and an immediate rescheduling at any urgent event (as, e.g., a machine break-down). In the survey of *dynamic scheduling* [23] several dynamic scheduling policies are described.

In the operations research literature, there are many mathematical optimization models of the JSP and the FJSP employing variables similar to those utilized by Manne [21] in 1960. In the evaluations made by Pan [25] and Demir and İşleyen [9] for the JSP and the FJSP, respectively, the Manne models were found to be the best. Both of these studies were conducted with the objective of minimizing the makespan

and the comparisons were made with several alternative models employing variables similar to those utilized by Wagner [36] and Bowman [4]. The Bowman models are called *time-indexed models* since they are based on a time discretization of the planning horizon. The decision variables used in [4] equal 1 if the corresponding job is processed by a specific resource during a specific time period, and 0 otherwise. An alternative definition of time-indexed decision variables is utilized by Sousa and Wolsey [30]: a variable equals 1 if the corresponding job *starts* in a specific discrete time step, and 0 otherwise.

According to [9] the best Manne model available is found in [24]. [34] compare this model with an iterative scheduling procedure employing a time-indexed model with decision variables according to [30]. The iterative scheduling procedure solves the time-indexed model for iteratively smaller time steps, i.e., with an increasing accuracy. The value of the makespan for the best schedule found in one iteration is used to determine the length of the time horizon for the next iteration in order to keep the total number of time steps required for the model in each iteration of the procedure as small as possible. When the objective to minimize the weighted sum of the completion times and the tardiness was employed, our iterative scheduling procedure outperformed the Manne model. However, [1] compared a Manne model (therein referred to as a "disjunctive model") with a time-indexed model for a JSP with non-fixed resource availability constraints (i.e., including PM activities), and in their case the Manne model performed the best. The latter comparison was, however, made for the objective of minimizing the makespan, and the time-indexed model was solved as is, i.e., without an iterative procedure for determining a suitable value of the time horizon.

To the best of our knowledge, there is no published mathematical optimization model of an FJSP extended to take fixture availability, PM, and/or unmanned night shifts into account. The only published work found concerning an FJSP including fixture availability constraints are [27] and [32]; both describe simulation-based scheduling approaches; the latter studies the multitask cell at GKN and includes fixture availability constraints and unmanned night shifts.

The constraints describing the scheduling of necessary PM activities are often called *availability constraints* in the literature. [13] categorize availability constraints into two types: *fixed* and *non-fixed*. When employing fixed availability constraints, the PM activities are already scheduled at fixed time slots, during which the resources are unavailable for processing. We consider non-fixed availability constraints, i.e., the starting time of the period of unavailability is flexible within a time window and is determined simultaneously with the production scheduling. [13], [37], and [28] consider the FJSP with non-fixed availability constraints and three objectives: makespan, *total workload*, and *critical machine workload*; a hybrid genetic algorithm, a filtered beam search algorithm, and a GRASP algorithm, respectively, is proposed.

Besides [32] we have found no published work considering the opportunity to schedule unmanned processing during night shifts, although the problem is not unique for the multitask cell at GKN: for example, [29] find it desirable to utilize the unmanned night shifts as much as possible. A similar problem arises when scheduling personnel having varying skills and working in shifts; see [2] for a recent survey of this field of research.

In Section 3, the scheduling problem is described in detail and the notation of the mathematical model is presented. Section 4 describes a number of priority dispatching rules. In Section 5, the time-indexed model for the FJSP, including the limited fixture availability, the required PM activities, and the unmanned night shifts, is presented.

The iterative solution procedure is described in Section 6, and in Section 7 the computational results based on real data are presented. Finally, in Section 8, conclusions are drawn.

3 Problem description

The multitask cell presently executes about 30 types of jobs on eight products. Each job consists of a set of operations that must be processed according to a predefined sequence. The first and the last operation of each job consist of the mounting into and demounting out of fixtures in one of the three set-up stations. Hence each job occupies a fixture throughout the whole job, and since the availability of each fixture type is limited, this constraint must be included in the scheduling problem. For most operations there is a flexibility in the choice of processing resource, that is, most operations are allowed to be processed in a subset of the resources.

The products typically visit the multitask cell multiple times on their way to completion, which implies the need for precedence relations between jobs. Unmanned processing is allowed at the start and/or at the end of some operations. Therefore these operations may be scheduled with the unmanned part during night shifts when no operators are present in the multitask cell. The jobs considered in the scheduling problem consist both of the jobs present in the multitask cell at the time of scheduling and of some jobs on their way to the cell. The release date of the first operation of each job equals the corresponding product's expected arrival time at the multitask cell. How many jobs to include is determined by the time $T^{\texttt{new}}$ when the scheduling problem is planned to be solved afresh with new data; all jobs with release dates less than $T^{\texttt{new}}$ should be considered in the scheduling problem. Note that since the optimal scheduling of the jobs considered may result in a makespan that is substantially longer than the planned time of rescheduling, but after time $T^{\texttt{new}}$ this schedule is of poor quality since the jobs with release dates longer than $T^{\texttt{new}}$ should be considered for the scheduling of this part of the planning horizon. All resources, except for the set-up stations, must be maintained periodically within a time window. A resource is occupied by the PM task while it is maintained.

The multitask cell is described more in detail in [33, 35]. The notation of sets, parameters, and indices used in this paper is summarized in Table 1. The assumptions regarding the flexible job shop of the multitask cell are given by the following list of items:

1. Job j has n_j operations that must be processed in a predefined order.
2. The operations $i \in \mathcal{N}_j := \{1, \ldots, n_j\}$ of the jobs $j \in \mathcal{J}$ are non-preemptive, i.e., once started, an operation must be completed.
3. The execution of operation i of job j requires a machine selected from a subset $\mathcal{M}_{ij} \subseteq \mathcal{K}$ of the set of available machines (i.e., resources) $\mathcal{K}$.
4. Time is discretized into the set $\mathcal{T}$ of time steps. We will use the terms "time" and "time step" to denote points in time.
5. Resource $k \in \mathcal{K}$ is available for processing from time a_k and onwards.
6. The transportation times between the machines inside the multitask cell are neglected.
7. All the machines can be maintained simultaneously.
8. At the time when a maintenance task $j \in \mathcal{J}^{\text{maint}}$ is performed on a machine, no operation can be processed on that machine.

9. Each maintenance task must start within a predefined time window.
10. Each job $j \in \mathcal{S}_f \subset \mathcal{J}$ occupies a fixture of type $f \in \mathcal{F}$ during all of its operations.
11. The number of available fixtures of each type f is limited.
12. During the unmanned (night) shifts, solely unmanned processing is allowed to be scheduled.
13. Some jobs $(j, q) \in \mathcal{Q} \subset \mathcal{J} \times \mathcal{J}$ are subject to precedence constraints with a time lag v_{jq}, i.e., job j has to be completed at least v_{jq} time steps before job q starts.

Table 1 Nomenclature

Sets	*Descriptions*
$\mathcal{N}_j$	set of operations of job j; $\mathcal{N}_j = \{1, \ldots, n_j\}$
$\mathcal{J}$	set of jobs
$\mathcal{J}^{\text{maint}}$	set of preventive maintenance (PM) tasks
$\mathcal{Q}$	set of pairs of jobs with precedence constraints; $\mathcal{Q} \subset \mathcal{J} \times \mathcal{J}$
$\mathcal{K}$	set of machines (i.e., resources)
$\mathcal{K}^{\text{setup}}$	set of set-up stations
$\mathcal{M}_{ij}$	set of machines allowed to process operation i of job j, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$
$\mathcal{T}$	set of time steps; $\mathcal{T} = \{0, \ldots, T-1\}$
$\mathcal{F}$	set of fixture types
$\mathcal{S}_f$	set of jobs that use fixtures of type $f \in \mathcal{F}$; $\mathcal{S}_f \subset \mathcal{J}$
$\mathcal{P}$	set of night shifts
Parameters	*Descriptions*
n_j	total number of operations of job j, $j \in \mathcal{J}$
p_{ij}	processing time of operation i of job j, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$
r_{ij}	release date (time) of operation i of job j, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$
$\bar{r}^n(\bar{r}^n)$	start (end) time of night shift n, $n \in \mathcal{P}$
$\hat{r}_{ijn}(\bar{r}_{ijn})$	intermediate deadline (release date) of operation i of job j before (after) night shift n, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$, $n \in \mathcal{P}$
δ_{ij}	remaining processing time of job j at the start of operation i, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$
d_j	due date (time) of job j, $j \in \mathcal{J}$
a_k	time when resource $k \in \mathcal{K}$ becomes available
T	number of time steps in the time horizon
ℓ	length [hours] of each time step
λ_{ijk}	$= 1$ if operation i of job j can be processed on resource k, i.e., if $k \in \mathcal{M}_{ij}$, $= 0$ otherwise, $i \in \mathcal{N}_j$, $j \in \mathcal{J}$, $k \in \mathcal{K}$
γ_f	total number of fixtures of type f, $f \in \mathcal{F}$
κ_{jk}	$= 1$ if PM task j should be performed in resource k, $= 0$ otherwise, $j \in \mathcal{J}^{\text{maint}}$, $k \in \mathcal{K} \setminus \mathcal{K}^{\text{setup}}$
τ_{jk}	start time of the time window for PM task j in resource k, $j \in \mathcal{J}^{\text{maint}}$, $k \in \mathcal{K} \setminus \mathcal{K}^{\text{setup}}$
Δ	length (in time steps) of the PM time window
$\alpha_j(\beta_j)$	objective weight for the completion time (tardiness) of job j, $j \in \mathcal{J}$
ε	objective weight for the first operation of each job j, $0 < \varepsilon \ll \alpha_j$, $j \in \mathcal{J}$
$\varepsilon^{\text{maint}}$	objective weight for the PM tasks

During a (possibly empty) time interval at the start and/or end of each operation a certain amount of *unmanned processing* is allowed. Hence, also parts of unmanned jobs may be scheduled during night shifts (see item 12 above). In fact, for some operations unmanned processing is allowed throughout the whole operation. Should this property hold for all operations, the night shifts could be scheduled analogously with the day shifts. However, the first ($i = 1$) and last ($i = n_j$) operations of each job j, i.e., the mounting into and demounting out of fixtures, are always entirely manual. Currently, for about half of the operations that are performed in any of the five multipurpose machines, an operator must be present during part of the processing time. Two partly unmanned scheduled operations are illustrated in Figure 1.

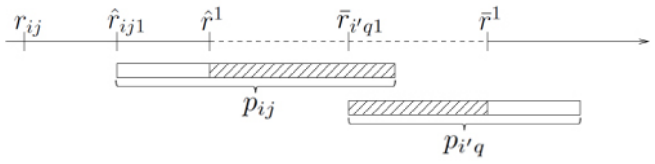


Fig. 1 Night shift 1 starts (ends) at time $\hat{r}^1$ ($\bar{r}^1$). Operation i (i') of job j (q) with processing time p_{ij} ($p_{i'q}$) is scheduled such that its allowed unmanned period (striped) is maximally utilized.

We denote by $\hat{r}^n$ ($\bar{r}^n$) the start (end) time of night shift n. Further, we denote by p_{ij}^{end} (p_{ij}^{start}) the amount of unmanned processing time allowed towards the end (at the beginning) of operation i of job j. An *intermediate deadline* for night shift n is then defined as

$$\hat{r}_{ijn} := \hat{r}^n - (p_{ij} - p_{ij}^{\text{end}}), \tag{1a}$$

and an *intermediate release date* for night shift n is defined as

$$\bar{r}_{ijn} := \bar{r}^n - p_{ij}^{\text{start}}. \tag{1b}$$

In Figure 1, operations i and i' are scheduled such that the night shift is maximally utilized, i.e., operation i of job j is scheduled at its intermediate deadline, $\hat{r}_{ij1}$, and operation i' of job q at its intermediate release date $\bar{r}_{i'q1}$.

The precedence relations between jobs (Item 13 above) exist due to the fact that the products typically visit the multitask cell multiple times on their way to completion. After job j is completed, the corresponding product is transported to and processed in other workshops in the factory during v_{jq} time steps before returning to the multitask cell for processing of job q, $(j,q) \in \mathcal{Q} \subset \mathcal{J} \times \mathcal{J}$.

The release date r_{1j} of the first operation of job j equals the expected arrival time of the corresponding part at the multitask cell. The release date of operation i is then calculated as $r_{ij} := r_{i-1,j} + p_{i-1,j}$, $i = 2, \ldots, n_j$.

4 Priority dispatching rules

The most well-known and widely used dispatching rule is probably the first–in, first–out (FIFO) priority rule; it is also currently used as a decision support in the detailed production planning of a majority of the workshops at GKN Aerospace Engine Systems. Research on dispatching rules has been active for several decades and over 100 dispatching rules are proposed in the literature; see [15]. For an extensive summary and discussion on priority dispatching rules, see [3] and [14]. The research on dispatching rules is still a relevant topic: these rules are widely used in practice (e.g., at GKN), researchers still propose new dispatching rules [20], and priority dispatching rules are often included in meta-heuristics developed for production planning problems [8].

Dispatching rules can be classified in various ways; a distinction can, e.g., be made between *static* and *dynamic* rules [26]. Dynamic rules are time dependent, i.e., the priorities change over time, whereas static rules (as, e.g., the FIFO priority rule) are not.

The built-in scheduling algorithm in the control system of the multitask cell is based on a critical ratio (CR) (see [18]) defined as

$$\mathrm{CR}_j(t) := \begin{cases} \min\limits_{i\in\mathcal{N}_j} \left\{ \left(1 + (d_j - t)\,|\mathcal{M}_{ij}|\right) \left(1 + \sum_{i'=i}^{n_j} p_{i'j}\right)^{-1} \right\}, & d_j > t, \\ \min\limits_{i\in\mathcal{N}_j} \left\{ \left[1 + (t - d_j)\,|\mathcal{M}_{ij}| \left(1 + \sum_{i'=i}^{n_j} p_{i'j}\right)\right]^{-1} \right\}, & d_j \le t, \end{cases} \tag{2}$$

for all $j \in \mathcal{J}$, where the sums of processing times over operations equal the total remaining processing time for the corresponding job. The job is assigned the minimum critical ratio among its operations. At time t the job j possessing the lowest value of $\mathrm{CR}_j(t)$ is given the highest priority. When a job is late, i.e., $d_j \le t$, the job is given a higher priority, since then $\mathrm{CR}_j(t)$ is decreased. If a machine is available at time t, then the operation corresponding to the job with the highest job priority is scheduled in this machine, provided that all precedence constraints and other side constraints are fulfilled.

In this article, we will compare the schedules found by our iterative scheduling procedure with those constructed by the static FIFO rule and the dynamic CR rule based on (2).

5 The time-indexed formulation

Time-indexed models are based on a discretization of the planning horizon, which is divided into a set $\mathcal{T} := \{0, \ldots, T-1\}$ of intervals, each of length ℓ hours. The number T of time intervals must be large enough such that the time horizon $[0, T\ell]$ can contain an optimal schedule. We define the decision variables as

$$x_{ijku} = \begin{cases} 1, & \text{if operation } i \text{ of job } j \text{ starts at time } u \text{ in resource } k, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $i \in \mathcal{N}_j$, $j \in \mathcal{J}$, $k \in \mathcal{K}$, and $u \in \mathcal{T}$. Similar variable definitions have been employed by Dyer and Wolsey [10] and Kedad-Sidhoum et al. [19] for single and parallel machine scheduling problems, respectively. The set of feasible solutions to a general

FJSP is defined by the following constraints:

$$\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{N}_j}\sum_{\mu=(u-p_{ij}+1)_+}^{u} x_{ijk\mu} \le 1, \qquad k\in\mathcal{K},\ u\in\mathcal{T}, \tag{4a}$$

$$\sum_{k\in\mathcal{M}_{ij}}\sum_{\mu=0}^{u} x_{ijk\mu} - \sum_{l\in\mathcal{M}_{i+1,j}}\sum_{\nu=0}^{u+p_{ij}} x_{i+1,jl\nu} \ge 0, \qquad u\in\{0,\dots,T-p_{ij}\},\ i\in\mathcal{N}_j\setminus\{n_j\},\ j\in\mathcal{J}, \tag{4b}$$

$$\sum_{k\in\mathcal{M}_{ij}}\sum_{u\in\mathcal{T}} x_{ijku} = 1, \qquad i\in\mathcal{N}_j,\ j\in\mathcal{J}, \tag{4c}$$

$$\sum_{k\in\mathcal{K}\setminus\mathcal{M}_{ij}}\sum_{u\in\mathcal{T}} x_{ijku} = 0, \qquad i\in\mathcal{N}_j,\ j\in\mathcal{J}, \tag{4d}$$

$$x_{ijku} = 0, \qquad u\in\{0,\dots,\max\{r_{ij},a_k\}-1\},\ k\in\mathcal{M}_{ij},\ i\in\mathcal{N}_j,\ j\in\mathcal{J}, \tag{4e}$$

$$x_{ijku} = 0, \qquad u\in\{T-\delta_{ij}+1,\dots,T-1\},\ k\in\mathcal{M}_{ij},\ i\in\mathcal{N}_j,\ j\in\mathcal{J}, \tag{4f}$$

$$x_{ijku}\in\{0,1\}, \quad i\in\mathcal{N}_j,\ j\in\mathcal{J},\ k\in\mathcal{K},\ u\in\mathcal{T}, \tag{4g}$$

where $(b)_+ := \max\{0,b\}$, $b\in\mathbb{R}$. In the model (4), all parameters are assumed to be integer-valued; this sometimes necessary approximation of the real data is discussed in Section 6. The constraints (4a) ensure that at most one operation at a time is scheduled in each resource. The constraints (4b) define the precedence relations between the operations of a job. The constraints (4c) ensure that each operation is scheduled to be processed exactly once in an allowed resource. The constraints (4d) assign the value 0 to all variables corresponding to an operation and the set of resources in which it may not be processed. These constraints are redundant; we discovered, however, in [34] that an optimal solution was found earlier (w.r.t. clocktime) when they were included. The constraints (4e) ensure that no operation is scheduled before its release date or before an allowed resource is available, and the constraints (4f) eliminate the possibility to schedule an operation too late such that it or its succeeding operations cannot be completed before the end of the planning horizon. The inclusion of the constraints (4g) ensures that the variables only take on binary values.

The time-indexed model formulated in [35] employs a parameter λ_{ijk}, which equals 1 if operation i of job j is allowed to be processed in resource k, and 0 otherwise. In this alternative formulation, the set $\mathcal{M}_{ij}$ utilized in the current model is replaced by the set $\mathcal{K}$ in the constraints (4b)–(4c), (4e)–(4f) (the constraints (4d) are removed), and the constraints

$$\sum_{u\in\mathcal{T}} x_{ijku} \ \le\ \lambda_{ijk}, \quad i\in\mathcal{N}_j,\ j\in\mathcal{J},\ k\in\mathcal{K}, \tag{5}$$

are added to the model. Before the preprocessing is performed in the optimization solver AMPL-CPLEX12 [12, 16], the model employing the inequalities (5) comprises more constraints than the model (4). However, after the preprocessing is made by the solver, we have observed that these two models contain the same numbers of variables and constraints. The retaining of the constraints (4d) however enables a higher degree of parallelization of the computations; as a result, the computation time required to solve

the model (4) hence is shorter (w.r.t. clocktime) than when employing the parameters λ_{ijk} and replacing the constraints (4d) by (5).

5.1 Constraints unique for the multitask cell

The precedence relations with a time lag between two jobs that are to be performed on the same physical component, are modelled similarly to the precedence constraints (4b) between the operations of the same job:

$$\sum_{k\in\mathcal{M}_{n_{jj}}}\sum_{\mu=0}^{u} x_{n_jjk\mu} - \sum_{k\in\mathcal{M}_{1q}}\sum_{\nu=0}^{u+v_{jq}} x_{1qk\nu} \geq 0, \qquad u=0,\ldots,T-v_{jq},\ (j,q)\in\mathcal{Q}. \quad (6)$$

In the multitask cell, the three set-up stations, denoted by $\mathcal{K}^{\text{setup}}$ and in which the physical parts are mounted into and demounted out of fixtures, are identical. All such operations can thus be performed in any set-up station. In order to eliminate the corresponding mathematical symmetry, these resources are treated as one, with a common capacity of three units. The constraints (4a), for $k\in\mathcal{K}^{\text{setup}}$, are hence reformulated as

$$\sum_{k\in\mathcal{K}^{\text{setup}}}\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{N}_j}\sum_{\mu=(u-p_{ij}+1)_+}^{u} x_{ijk\mu} \leq |\mathcal{K}^{\text{setup}}|, \qquad u\in\mathcal{T}, \quad (7a)$$

and the range for the index k in the constraints (4a) is hence altered to $\mathcal{K}\setminus\mathcal{K}^{\text{setup}}$, according to

$$\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{N}_j}\sum_{\mu=(u-p_{ij}+1)_+}^{u} x_{ijk\mu} \leq 1, \qquad k\in\mathcal{K}\setminus\mathcal{K}^{\text{setup}},\ u\in\mathcal{T}. \quad (7b)$$

5.2 Side constraints regarding fixtures, preventive maintenance, and night shifts

In the multitask cell the number of fixtures of type f simultaneously in use is limited to γ_f. Since a part is mounted into the fixture during all the operations of a job, the limited fixture availability is formulated as

$$\sum_{j\in\mathcal{S}_f}\sum_{k\in\mathcal{K}}\left(\sum_{\mu=0}^{u} x_{1jk\mu} - \sum_{\nu=0}^{(u-p_{n_jj})_+} x_{n_jjk\nu}\right) \leq \gamma_f, \qquad f\in\mathcal{F},\ u\in\mathcal{T}. \quad (8)$$

A PM task occupies the corresponding machine during the whole maintenance operation. Hence, it can be regarded as a job consisting of one operation. Let $\mathcal{J}^{\text{maint}}$ denote the set of PM tasks. Therefore, for each PM task $j\in\mathcal{J}^{\text{maint}}$, $n_j=1$ with a duration p_{1j}. Further, at least κ_{jk} occurrences of PM task j on resource k must start during a predefined time window of length Δ, i.e.,

$$\sum_{\mu=\max\{\tau_{jk},a_k\}}^{\min\{\tau_{jk}+\Delta-1,T-p_{1j}\}} x_{1jk\mu} \geq \kappa_{jk}, \qquad j\in\mathcal{J}^{\text{maint}},\ k\in\mathcal{K}\setminus\mathcal{K}^{\text{setup}}, \quad (9a)$$

where the decision variables x_{1jku} correspond to PM task $j \in \mathcal{J}^{\text{maint}}$. Due to the objective function (see Section 5.3), in an optimal solution the constraints (9a) will be fulfilled with equality whenever possible. No PM tasks need to be performed in the set-up stations. Since no operation can be processed in a resource while maintained, the capacity constraints (7b) are altered as

$$\sum_{j \in \mathcal{J} \cup \mathcal{J}^{\text{maint}}} \sum_{i \in \mathcal{N}_j} \sum_{\mu=(u-p_{ij}+1)_+}^{u} x_{ijk\mu} \leq 1, \qquad k \in \mathcal{K} \setminus \mathcal{K}^{\text{setup}},\ u \in \mathcal{T}. \tag{9b}$$

For some alternative formulations of PM scheduling, see [35].

To take the unmanned night shifts $n \in \mathcal{P}$ into account the constraints

$$x_{ijku} = 0, \qquad k \in \mathcal{M}_{ij},\ u \in \{\hat{r}_{ijn}+1, \ldots, \bar{r}_{ijn}-1\},\ i \in \mathcal{N}_j,\ j \in \mathcal{J},\ n \in \mathcal{P}, \tag{10a}$$

are introduced, where the parameters $\hat{r}_{ijn}$ and $\bar{r}_{ijn}$ are defined in (1). The constraints (10a) ensure that operation i of job j is not scheduled to start during the interval between its operation-specific intermediate deadline, $\hat{r}_{ijn}$, and release date, $\bar{r}_{ijn}$. Note that this interval may intersect with daytime, due to the allowed time of unmanned processing at the start and/or end of the operation. Each PM task requires an operator present in the multitask cell throughout its duration, which is modelled by the constraints

$$x_{1jku} = 0, \qquad k \in \mathcal{K}, u \in \left\{\hat{r}^n - p_{1j}+1, \ldots, \bar{r}^n - 1\right\},\ j \in \mathcal{J}^{\text{maint}},\ n \in \mathcal{P}, \tag{10b}$$

where $\hat{r}^n$ ($\bar{r}^n$) denotes the starting (ending) time of night shift n.

5.3 The objective function

As stated in [34] the minimization of the makespan is not suitable as an objective if the flexible job shop is operating in a dynamic environment. In fact, some jobs then risk never being processed since nothing prevents a certain job from being scheduled close to the end of each subsequent schedule—if the rescheduling is performed before the last job of the previous schedule has started. This risk is reduced if the objective function includes the tardiness $T_j \geq 0$ of job j, multiplied by a weight $\beta_j \geq 0$ being proportional to the job's delay. Further, at the start of the planning horizon, typically not all jobs are yet released, but are expected to arrive at the workshop at given release dates. For an instance with at least one job possessing a very late release date—such that all the other jobs can be finished before the earliest possible completion time of this job—the minimization of the makespan would yield an abundance of optimal solutions, some of which would result in very poor throughput times for the remaining jobs.

We propose an objective function focusing on minimizing the total weighted tardiness. Since instances with no tardy jobs may very well occur in real industrial cases, we also include a sum of weighted completion times $C_j \geq 0$ in the objective, such that the resulting objective function becomes that to

$$\text{minimize} \quad \sum_{j \in \mathcal{J}} \left(\alpha_j C_j + \beta_j T_j\right), \tag{11}$$

where $T_j := (C_j - d_j)_+$, and $\alpha_j > 0,\ j \in \mathcal{J}$, are objective weights for the completion times of the jobs. We define the tardiness objective weights as $\beta_j := B(1 - d_j/|d_D|)_+$,

where D denotes the job having the largest absolute due date while not being an outlier, and the parameter B satisfies $B \gg \alpha_j$, since tardiness is meant to be the main objective (see [35]). The completion time can be expressed as

$$C_j := \sum_{k \in \mathcal{M}_{n_j j}} \sum_{u \in \mathcal{T}} (u + p_{n_j j}) x_{n_j jku} \tag{12}$$

and the objective function for the model (4) is formulated as to minimize

$$z'(x) := \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{M}_{n_j j}} \sum_{u \in \mathcal{T}} \Big(\big[\alpha_j (u + p_{n_j j}) + \beta_j (u + p_{n_j j} - d_j)_+ \big] x_{n_j jku} - \varepsilon u x_{1jku} \Big). \tag{13}$$

The term $-\varepsilon u x_{1jku}$ is included in (13) in order to schedule the first operation of each job as late as possible, but since the completion times of the jobs should not be affected by this term the relations $0 < \varepsilon \ll \alpha_j$, $j \in \mathcal{J}$, must hold. This inclusion also facilitates the rescheduling and is hence suitable in a dynamic environment: fewer fixtures are then tied up for jobs whose succeeding operations have not yet started at the time of rescheduling; see [35]. If the scheduling problem includes PM tasks, yet another term needs to be included in the objective function, according to

$$z(x) := z'(x) + \varepsilon^{\text{maint}} \sum_{j \in \mathcal{J}^{\text{maint}}} \sum_{k \in \mathcal{K} \setminus \mathcal{K}^{\text{setup}}} \sum_{u \in \mathcal{T}} x_{1jku}, \tag{14}$$

where $0 < \varepsilon^{\text{maint}} \ll \alpha_j$, $j \in \mathcal{J}$. The problem of scheduling the multitask cell including side constraints regarding PM, fixtures, and night shifts can now be formulated as that to minimize (14), subject to the constraints (4b)–(4g), and the side constraints (6)–(10). We have computationally tested this time-indexed model of the FJSP including different subsets of the side constraints; see Table 2.

Table 2 Definitions and notation for the models implemented and tested: TI – time-indexed model; F – fixture availability; M – PM activities; N – unmanned night shifts.

Notation	*Description*	*Model*
TI-FMN	FJSP including all side constraints	minimize (14) subject to (4b)–(4g), (6)–(10)
TI-FM	FJSP including fixture and PM constraints	minimize (14) subject to (4b)–(4g), (6)–(9)

6 The iterative solution procedure

As stated in [31], the time-indexed formulations are flexible and capable of accounting for many scheduling features, such as all the side constraints proposed in Section 5.2. There are, however, two major drawbacks in utilizing a discrete time representation, namely

i. the approximation of time, and
ii. the large numbers of binary variables and constraints required.

In order to decrease the gap between the optimal objective value of the approximate solution to the time-indexed model and the real optimal objective value, we have developed a *squeezing procedure* [34]. This procedure retains the sequence of operations scheduled in each of the machines, while the starting times of the operations are reset such that each operation starts as early as possible (i.e., without violating any constraints). In the last stage of the squeezing procedure, the first operation of each job is then scheduled as late as possible in order to reflect the last term of the objective function (13).

The numbers of binary variables and constraints depend on the choice of the time horizon T and the length ℓ of the time steps. Our *iterative procedure* is designed to keep the number of variables and constraints at a minimum, by iteratively decreasing the value of ℓ and by utilizing the solution from the previous iteration to feed the next iteration with a feasible solution and an appropriate value of T. The iterative procedure described in [34, Algoritm 1] is generic in the sense that it can be used to solve any time-indexed model.

At iteration s of the iterative procedure, let ℓ^s and T^s denote the length and the number of time steps in the planning horizon, respectively, and let the parameter values be scaled and rounded up or truncated, respectively, according to

$$p_{ij} := \left\lceil \frac{\widetilde{p_{ij}}}{\ell^s} \right\rceil ;\ r_{ij} := \left\lceil \frac{\widetilde{r_{ij}}}{\ell^s} \right\rceil ;\ \hat{r}_{ijn} := \left\lfloor \frac{\widetilde{\hat{r}_{ijn}}}{\ell^s} \right\rfloor ;\ \bar{r}_{ijn} := \left\lceil \frac{\widetilde{\bar{r}_{ijn}}}{\ell^s} \right\rceil ; \tag{15a}$$

$$\hat{r}^n := \left\lfloor \frac{\widetilde{\hat{r}^n}}{\ell^s} \right\rfloor ;\ \bar{r}^n := \left\lceil \frac{\widetilde{\bar{r}^n}}{\ell^s} \right\rceil ;\ d_j := \left\lfloor \frac{\widetilde{d}_j}{\ell^s} \right\rfloor ;\ i \in \mathcal{N}_j,\ j \in \mathcal{J},\ n \in \mathcal{P} ; \tag{15b}$$

$$v_{jq} := \left\lceil \frac{\widetilde{v_{jq}}}{\ell^s} \right\rceil ,\ (j,q) \in \mathcal{Q} ; \quad a_k := \left\lceil \frac{\widetilde{a_k}}{\ell^s} \right\rceil ,\ k \in \mathcal{K} ; \tag{15c}$$

$$\tau_{jk} := \left\lceil \frac{\widetilde{\tau_{jk}}}{\ell^s} \right\rceil ,\ \ j \in \mathcal{J}^{\text{maint}},\ \ k \in \mathcal{K} ; \quad \Delta := \left\lfloor \frac{\widetilde{\Delta}}{\ell^s} \right\rfloor , \tag{15d}$$

where the superscript "$\sim$" indicates the respective original (non-discretized) parameter values.

The process of the iterative procedure is defined in Figure 2. In step 1, the input data, denoted $\texttt{data}^1$, is generated, employing the assignments in (15) for $s = 1$. In the computational tests, we used $\ell^1 := \lceil \widetilde{p_{\max}}/2 \rceil$, where $\widetilde{p_{\max}} := \max_{j \in \mathcal{J},\, i \in \mathcal{N}_j} \{\widetilde{p_{ij}}\}$. The value of ℓ^1 is chosen sufficiently large such that most of the processing times p_{ij} are valued 1. Most instances can then be solved by the time-indexed model in a few seconds, even for large values of T. Consequently, we let $T^1 := \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{N}_j} p_{ij}$, which theoretically can be too short (if, e.g., the time lags v_{jq} are very long as compared to the processing times), but in most cases this time horizon is more than sufficiently long and hence considerably longer than the makespan of an optimal schedule. If this value of T^1 is too small, the problem is infeasible in the first iteration, and T^1 then has to be assigned a larger value.

In Step 2 in Figure 2, for $s = 1$, the model TI-FM is solved. The reason why we solve the problem without considering unmanned night shifts in the first iteration is that it may be impossible to find feasible solutions when the data, generated according to (15) employing a large value of ℓ^1, is very coarse. In our previous work [33, p. 35] the value of T^1 was determined by a problem-specific heuristic, which is here replaced by the first iteration.

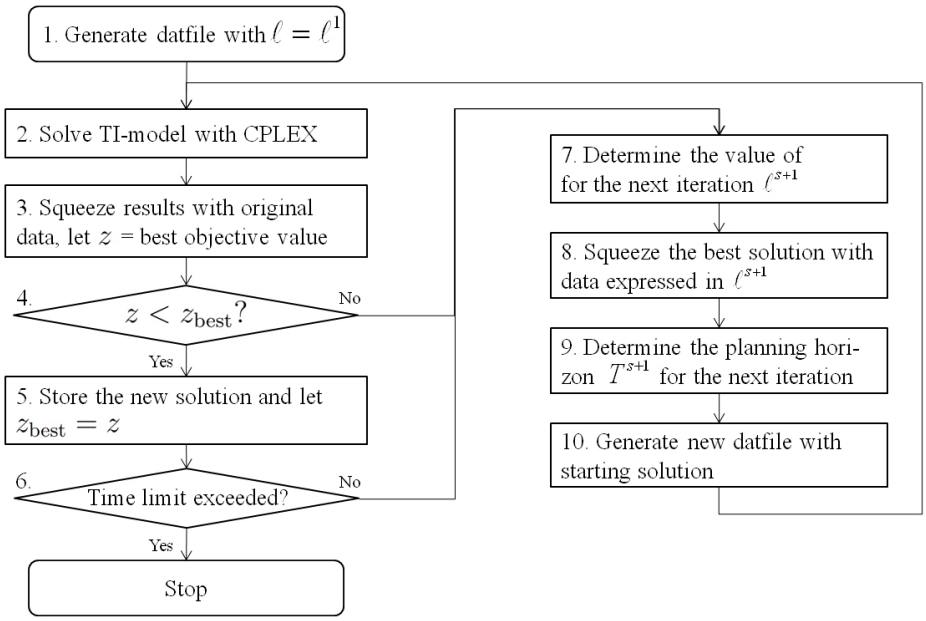


Fig. 2 The process of the iterative procedure.

In step 3, the solution obtained by having solved the time-indexed model in step 2 is squeezed with respect to the original data and the resulting objective value, z^s, is stored. In step 4, it is checked whether the value of z^s is the best one found so far. If this is the case, then the new solution is stored, and $z^{\text{best}} := z^s$ in step 5. Step 6 checks if the time limit is exceeded. If this is the case, the procedure is terminated. If not, $s+1$ replaces s in step 7 and the new value of ℓ^s is determined (see also Table 4). In step 8, $\texttt{data}^s$ is generated according to (15). In step 9, the best solution found so far is squeezed with respect to $\texttt{data}^s$. The resulting solution is stored and used as a starting solution for the model TI-FMN at iteration s. Finally, in step 10 the value of T^s is determined according to

$$T^s := \max\left\{\left\lceil \frac{1}{\ell^s}\left(\widetilde{C_{\max}} + \widetilde{p_{\max}}\right)\right\rceil, C_{\max}\right\}, \tag{16}$$

where $C_{\max}$ denotes the makespan of the starting solution expressed in time steps of length ℓ^s, and $\widetilde{C_{\max}}$ denotes the makespan (expressed in hours) of the solution found in iteration s and squeezed with respect to the original data. The reason for not letting T^s equal $C_{\max}$ is that the makespan of the optimal solution in iteration s may exceed $C_{\max}$; see [33, pp. 34–36] for a discussion. This is more likely to happen when the difference $\ell^{s-1} - \ell^s$ is small, since then the likelihood is large that $C_{\max}$ is close to the makespan of the optimal solution in iteration s. For our computations a time limit was used as a termination criterion, since the main focus of this work is to investigate the practical usefulness of this procedure. [34, 35] used the degree of accuracy as termination criterion, i.e., the computations were stopped at a desired length of the time step.

The squeezing procedure resets all starting times of the operations such that each operation starts as early as possible and without violating any constraints. As mentioned above, the model TI-FM is solved in step 2 of the first iteration. Provided there are no deadlines for PM tasks that can't be met due to the coarse rounding of data, the squeezing procedure is always able to find a feasible schedule of the model TI-FMN for the second iteration (where the night shift constraints (10) have been included). This is due to the fact that the squeezing procedure may schedule some operations after the last night shift if these operations cannot be scheduled between two night shifts (due to the coarse rounding of data). If there are PM deadlines that can't be met, either the model TI-FM must be used also in the second iteration, or a smaller value of ℓ^s must be employed such that the PM deadlines can be met.

In [34], the general FJSP without side constraints was investigated; the coding of the squeezing procedure was then fairly simple. Here, the side constraints (6)–(10) make the coding (in [22]) of the squeezing procedure much more complicated. We hence concluded that using the iterative procedure without the squeezing—as done in [35]—could be worth considering. To investigate the benefits of the squeezing procedure, we have tested the iterative procedure with, and without, the squeezing. This investigation is discussed below.

7 Computational tests and results

Through the manufacturing site's Enterprise Resource Planning (ERP) system, real data was collected from the multitask cell during 2012 and 2013. Nine out of in total twelve real scenarios are identical to the scenarios employed in [35], except for the data describing the unmanned night shifts. At the time, some personnel was working during the night, and in [35] we assumed that the work force then was sufficient for the schedules produced. Since the beginning of 2013, the night shifts are, however, unmanned. In this paper, we use the scenarios from 2012 as if they included the work force of today, i.e., with no personnel during the night shifts. In the multitask cell, there are about 30 types of jobs consisting of 3–7 operations to be performed on eight products.

The computations were carried out using AMPL-CPLEX12 [12, 16] on a computer with two 2.66GHz Intel Xeon X5650 processors, each with six cores (24 threads), and a total memory of 48 Gbytes of RAM.

Section 7.1 describes the test setting, and in Section 7.2 we report on the results from our computational tests and compare the schedules obtained by our iterative procedure with those constructed by the FIFO and the CR priority rule, respectively.

7.1 The experimental setup

The objective weights (see Section 5.3) used in the computations are equivalent to those used in [35], i.e., $\alpha_j := 1$, $B := 10$ (implying that $\beta_j \in [0, 20]$, $j \in \mathcal{J}$, except for very much delayed outliers, for which $\beta_j > 20$ may hold), $\varepsilon := 0.001$, and $\varepsilon^{\text{maint}} := 0.0001$.

For the rescheduling, we propose the hybrid event-driven policy described in Section 2. The scheduling procedure needs to produce near-optimal schedules for the period chosen within an acceptable time frame. For the case of the multitask cell, the manager of the cell has agreed that a reasonable clock time to spend on the computing of a near-optimal schedule for the coming shift is around 15 minutes. From each of the twelve real scenarios, an instance for the coming shift including all jobs and PM tasks with release dates $\widetilde{r_{1j}} \leq 8$h was created; the sizes of the twelve resulting instances vary between 20 and 33 jobs.

As discussed in [35], the makespan of an optimal schedule for a specific instance may cover a period that is substantially longer than the coming shift. In order to guarantee that a high quality schedule for the subsequent shift is produced, a rescheduling must, however, be performed at the start of this shift; the new instance considered for this rescheduling also includes the jobs that are expected to arrive at the cell during this subsequent shift. The details in the later part of the schedule are hence not of practical interest, since the corresponding jobs are likely to be moved when rescheduled at a later point in time. Therefore, we consider the first two night shifts only in each instance, i.e., $\mathcal{P} = \{1, 2\}$.

In Table 3 the variants of the scheduling procedures tested are defined. The

Table 3 Test settings. For all test settings denoted TI-FMN_x, the model TI-FM was employed in iteration 1, and the model TI-FMN in iterations $2, 3, \ldots$.

Test setting	*Time limit* (s *clock time*)	*Implementation of the iterative procedure*
TI-FMN_2h	7200	including squeezing
TI-FMN_15min	900	including squeezing
TI-FMN_no_squeeze_2h	7200	without squeezing
TI-FMN_no_squeeze_15min	900	without squeezing
TI-FM_2h	7200	including squeezing

lengths ℓ^s of the time intervals chosen for each of these scheduling procedures are indicated in Table 4. TI-FMN_15min was computed with all three choices of values for ℓ^s, i.e., such that $\zeta \in \{1.8, 2.0, 2.2\}$; see Table 4. The value $\zeta = 1.8$

Table 4 Values of the step lengths (ℓ^s) and `mipgap` limits (g^s) employed in the computations. The value of ζ is used as a suffix for the test settings listed in Table 3.

		ℓ^s (h)		
s	g^s (%)	$\zeta = 1.8$	$\zeta = 2.0$	$\zeta = 2.2$
1	5	12	12	12
2	2	3	4	4
3	1	1.66667	2	1.81818
4	0.5	1	1	0.82645
$r \geq 5$	$g^{r-1}/2$	ℓ^{r-1}/ζ	ℓ^{r-1}/ζ	ℓ^{r-1}/ζ

was chosen since good results were achieved using this value in [34]. However, for the case of TI-FMN_no_squeeze, only $\zeta = 2.0$ was chosen since the iterative procedure without the squeezing procedure works best for integer values of ℓ^{s-1}/ℓ^s: the starting solution, which is computed by multiplying the time in-

dex u by ℓ^{s-1}/ℓ^s in the solution from the previous iteration expressed in the variables x_{ijku}, may otherwise be infeasible, which will result in an increased computation time. For TI-FMN_2h, $\zeta \in \{1.8, 2.0\}$ was employed, and TI-FM_2h was computed using $\zeta = 1.8$.

The iterative procedure was terminated at a time limit. The termination criterion employed in each iteration was either the total time remaining until the time limit or that `mipgap` $\leq g^s$.[1] Since the approximation of the data is less coarse for smaller values of ℓ^s, the value of g^s was also decreased in each iteration; see Table 4.

7.2 Test results

In order to demonstrate the effect of the squeezing procedure we solved the instances for the coming shift using the setting TI-FMN_15min_2.0. Since this setting yielded slightly better results than TI-FMN_15min_1.8 for most of the instances, we also tested TI-FMN_15min_2.2. These results are reported in Table 5, along with those for the setting TI-FMN_no_squeeze_15min, for the twelve coming shift scenarios; see also Figure 3.

Table 5 Computational results for the settings TI-FMN_15min_ζ, $\zeta \in \{1.8, 2.0, 2.2\}$, and TI-FMN_no_squeeze_15min_x, x $\in$ {hybrid, 2.0}. The relative differences, `diff`, to z^{best2h}, according to (17) are listed for the twelve coming shift scenarios. "hybrid" refers to squeezing the solution from the setting TI-FMN_no_squeeze_15min; "0" means that optimality is verified; bold numbers indicate the best result for each scenario.

		TI-FMN_15min			TI-FMN_no_squeeze_15min	
Scenario	$\lvert\mathcal{J}\rvert$	ζ=1.8 (%)	ζ=2.0 (%)	ζ=2.2 (%)	*hybrid* (%)	ζ=2.0 (%)
1	25	0.32	**0.00**	4.08	**0.00**	4.49
2	21	**0.00**	0.05	0.05	0.05	0.88
3	26	1.51	**0**	0.90	1.26	3.04
4	26	2.19	**0.01**	2.26	**0.01**	1.31
5	30	1.97	**1.95**	2.97	2.04	6.44
6	25	0.74	0	2.31	**−0.09**	3.06
7	33	1.23	**0.47**	0.79	**0.47**	3.69
8	27	**0.38**	0.52	0.39	0.50	3.07
9	30	2.56	2.56	4.57	**1.95**	8.26
10	24	7.19	1.27	9.87	**0.64**	9.29
11	27	11.59	**10.56**	11.23	10.63	25.24
12	20	8.61	1.56	8.03	**0.12**	0.65
Average	26.2	3.19	1.58	3.95	**1.46**	5.78

Since the results in the last iteration are computed for different values of ℓ^s—and some of the solutions are also squeezed—the values of `mipgap` are not compared. Instead, the numbers are compared with the best feasible objective value out of all test settings employing the time limit of two hours, denoted

[1] The `mipgap` is defined as the relative difference between the best lower bound (LB) and the best objective value, z, found. The definition used by CPLEX version 12 is `mipgap` := $\frac{|z-\text{LB}|}{10^{-10}+|z|} \cdot 100\%$.

z^{best2h}. The values listed in Table 5 are calculated as

$$\texttt{diff} := \frac{z^{\hat{s}} - z^{\text{best2h}}}{z^{\text{best2h}}} \cdot 100\%, \tag{17}$$

where $\hat{s}$ denotes the iteration at which the time limit was reached. In this comparison of the test settings of two hours duration, nine out of the twelve best results were found when employing the setting TI-FMN_2h_2.0. For a few of the instances, CPLEX ran out of memory before reaching the time limit of two hours, whence the last solution obtained was used.

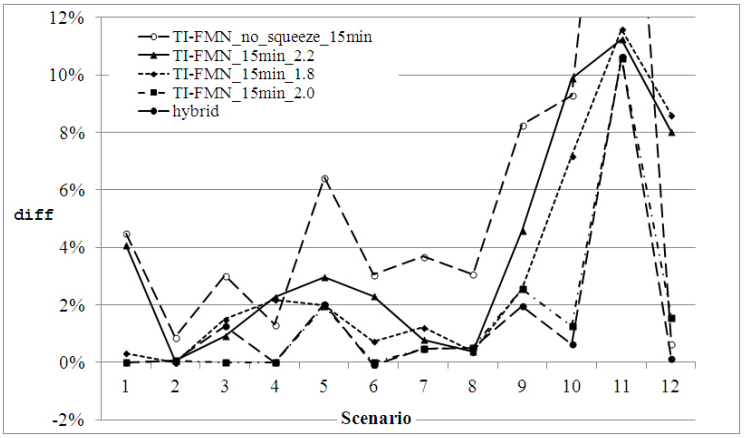


Fig. 3 The relative differences, `diff`, to z^{best2h}, according to (17) and listed in Table 5, plotted for all coming shift scenarios. The vertical axis is truncated at 12%, such that the value 25.24% (scenario #11, setting TI-FMN_no_squeeze_15min) is not visible.

We conclude from the results reported in Table 5 that the choice $\zeta = 2.0$ results in a slightly better performance than $\zeta = 1.8$ and $\zeta = 2.2$. During the computational tests made in [34], the setting $\zeta = 1.8$ performed the best for the benchmark test instances employed. However, the best choice of ℓ^s for each iteration depends on the instance data, and the data from the multitask cell differs a lot from the benchmark data used in [34]. The lowest average values of $\ell^{\hat{s}}$ over the twelve scenarios, i.e., the average of the time steps employed at the time limit, are attained when employing the setting $\zeta = 2.0$; see Table 6. A low value of ℓ^s, means that the solutions obtained by the time-indexed model typically are closer to the corresponding real optimal solutions; this is due to the approximation of the data being less coarse. The setting $\zeta = 2.0$ yielded the best results despite the risk that an unfavourable data pattern might be reproduced in each iteration when employing this setting: for example, with $\ell^1 = 4$h, $\ell^2 = 2$h, and $\ell^3 = 1$h a processing time of 4.1h is rounded up—with big errors—to 2, 3, and 5 time steps, respectively, while 4.0h is rounded up—with zero errors—to 1, 2, and 4 time steps, respectively.

Table 5 also reveals the effect of the squeezing procedure: the column entitled "hybrid" contains the results obtained by squeezing the solution obtained in the last iteration of TI-FMN_no_squeeze_15min. This hybrid setting yields

Table 6 The average of $\ell^{\hat{s}}$ over the twelve coming shift scenarios, where $\hat{s}$ denotes the iteration for which the time limit was reached. "—" indicates that the corresponding setting was not tested; bold numbers indicate the best result for each setting.

ζ	1.8	2.0	2.2
TI-FMN_15min_ζ	0.79h	**0.49h**	0.65h
TI-FMN_no_squeeze_15min_ζ	—	**0.59h**	—
TI-FMN_2h_ζ	0.52h	**0.34h**	—
TI-FMN_no_squeeze_2h_ζ	—	**0.34h**	—

equally good results as the setting TI-FMN_15min_2.0, while the value of `diff` is on average 4% higher when no squeezing is employed. In our implementation of the setting TI-FMN_15min new data is generated in each iteration, but for the setting TI-FMN_no_squeeze_15min, no data needs to be generated and exported during the computations other than the final result. The latter is of course more convenient and therefore, for instances resembling the multitask cell data instances, we recommend the hybrid setting of TI-FMN_no_squeeze_15min. While preparing the results presented in [34], the iterative solution procedure utilizing squeezing in each iteration was, however, observed to be a clear winner. This is partly explained by the fact that the gain of the squeezing procedure is typically larger for larger values of ℓ^s. For example, for the twelve scenarios reported in Table 5, the average relative difference between the objective values of the squeezed and the non-squeezed solutions obtained in the second iteration, with ℓ^2=4h, is 45%. The benchmark data instances generated by Fattahi et al. [11] and tested in [34] contain operations with very long processing times, which in turn require long planning horizons and therefore also relatively large values of ℓ^s for the first iterations.

7.2.1 The gain from the night shift constraints

The iterative scheduling procedure was run with the TI-FM_2h_1.8 setting, i.e., without taking the unmanned night shifts into account. In these tests, the solution obtained in each iteration but the last was squeezed—disregarding the night shifts, i.e., assuming that all operations requiring manual work could be performed also during the night. After the last iteration, which was terminated at the pre-specified time limit, the solution was squeezed taking the night shift constraints (10) into consideration such that a feasible schedule was obtained. Hence, this last solution is not squeezed—but rather expanded (in the time dimension)—by the squeezing procedure. The value of the resulting schedule was then compared with z^{best2h}, i.e., the best schedule obtained within two hours of computing, by calculating the relative difference `diff`, according to (17). The value of `diff` ranges between 1.87% and 27.63% over the twelve coming shift scenarios, with an average of 9.90%. Hence, for the case of the multitask cell, the gain—in terms of objective value—of including the night shift constraints in the scheduling procedure is around 10% (this figure of course being dependent on the instance data). For the operations in the multitask cell, the share of allowed unmanned processing ranges from 0% to 100%. When 100% unmanned processing is allowed for all operations, the night shift constraints are obviously not needed.

In Figure 4, the gain of including the night shift constraints is illustrated for scenario #12 comprising 20 jobs. In (a)—the solution obtained by TI-FM_2h_1.8 and then squeezed—the third last operation in machine MC3 cannot be scheduled before the second night shift due to the required manned processing in the later part of this operation. The resulting makespan is 100.5 hours. In the schedule corresponding to z^{best2h}, illustrated in (b), the same operation (here, the second last operation in MC3) is scheduled to start during the first night shift and being completed before the second night shift starts. The resulting makespan is 79.3 hours. The three last operations in MC3 in (a) are of the same kind and occupy the same type of fixture. Since there is only one such fixture, these operations cannot be scheduled in parallel. The dark grey bars at the beginning of the planning period represent the parameters a_k, i.e., the time steps when resource k is occupied.

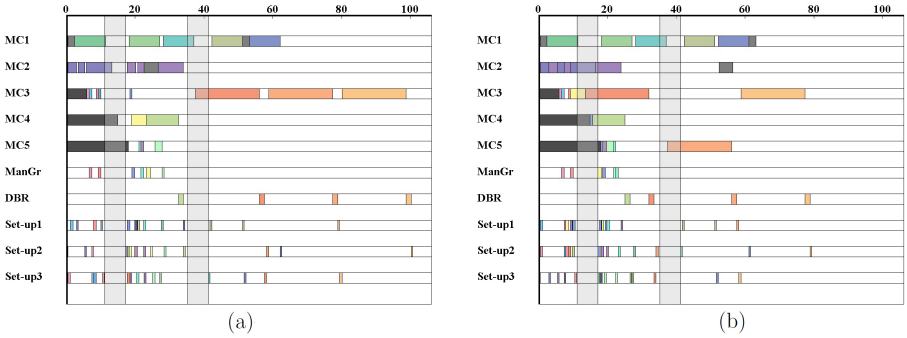


Fig. 4 The gain of the night shift constraints illustrated for scenario #12. The night shifts are marked by vertical light grey, transparent bars. (a) The feasible schedule obtained from the setting TI-FM_2h_1.8. The objective value from the last squeezing is 9386. (b) The best schedule obtained [the setting TI-FMN_no_squeeze_2h (hybrid) in this case]. The objective value from the last squeezing, $z^{\text{best2h}} = 7354$.

7.2.2 Comparison with the FIFO and the CR priority dispatching rules

The construction of schedules by the priority dispatching rules was coded such that each possible scheduling occasion from time zero to the planning horizon was considered. At any scheduling occasion in a resource k, (e.g., when an operation is completed in this resource) the operation currently possessing the highest priority among the operations available for scheduling (w.r.t., e.g., precedence and fixture constraints) is scheduled in this resource. As the FIFO rule is static, the priorities remain unchanged throughout the scheduling procedure—in contrast to the CR rule [defined in (2)].

The value of the objective function was computed for each of the schedules constructed by the dispatching rules and the relative difference to z^{best2h} was computed according to (17). The relative differences between the objective values from the FIFO (CR) schedules and the best solution found (z^{best2h}) were on average 37.4% (41.1%). Remarkably, the CR rule performs worse than the FIFO

rule; hence the use of the built-in scheduling method in the control system of the multitask cell is not recommended.

Figure 5 shows four schedules for scenario #8 constructed by (a) FIFO, (b) CR, (c) TI-FMN_15min_2.0, and (d) the best solution found [the squeezed solution of TI-FMN_no_squeeze_2h (hybrid)], with makespans of 91h, 107.1h, 76.1h, and 73.3h, respectively. For this scenario the solution with the shortest makespan was also the best solution found (w.r.t. objective value); note that this is *not* a general property. For scenario #8, the iterative procedure termi-

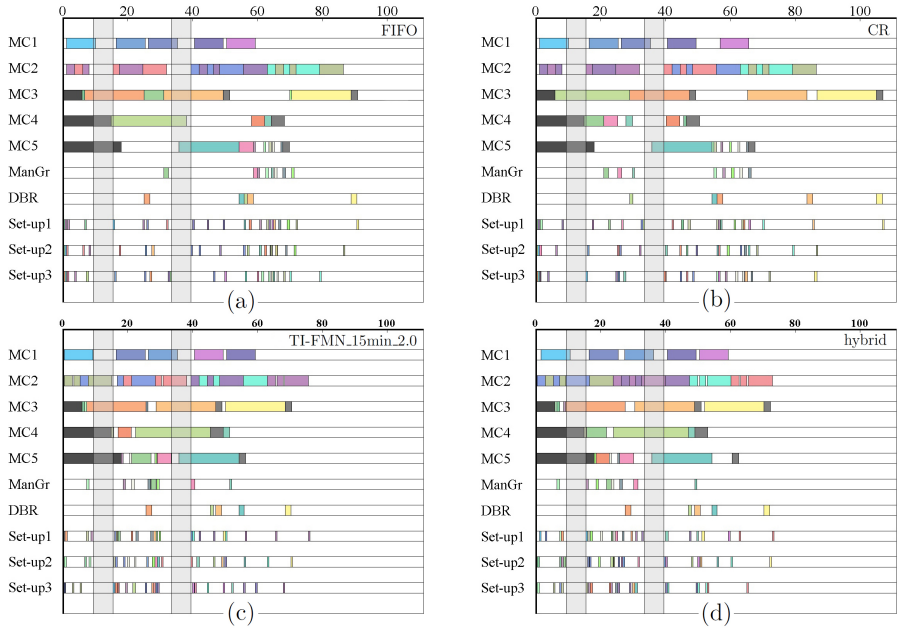


Fig. 5 The schedules of scenario #8 constructed by (a) FIFO, (b) CR, (c) TI-FMN_15min_2.0, and (d) the best solution found [the squeezed solution of TI-FMN_no_squeeze_2h (hybrid)]. The night shifts are marked by vertical light grey, transparent bars. The dark grey bars at the beginning of the schedules represent unavailable resources. The grey bars close to the end of the schedules represent PM activities.

nated at the pre-specified time limit at iteration $s = 6$, with $\ell^6 = 0.25$h for both settings TI-FMN_15min_2.0 and TI-FMN_no_squeeze_2h. Note that in the squeezed schedule produced using the setting TI-FMN_no_squeeze_2h [Figure 5(d)] there is no idle time in the machine MC2 during any of the night shifts, while none of the other scheduling procedures manage to completely fill the night shifts in MC2. The relative difference between z^{best2h} and the objective values of the TI-FMN_15min_2.0, the FIFO, and the CR schedule, according to (17), was 0.52%, 58%, and 65%, respectively. The idle times between operations in the four schedules in Figure 5 are due to either the precedence constraints (4b) and (6), the limited fixture availability (8), the night shift constraints (10), or the jobs' release dates (4e).

Figure 6 shows the relative difference, according to (17), between the objective value obtained by each of the scheduling procedures tested and z^{best2h}. The

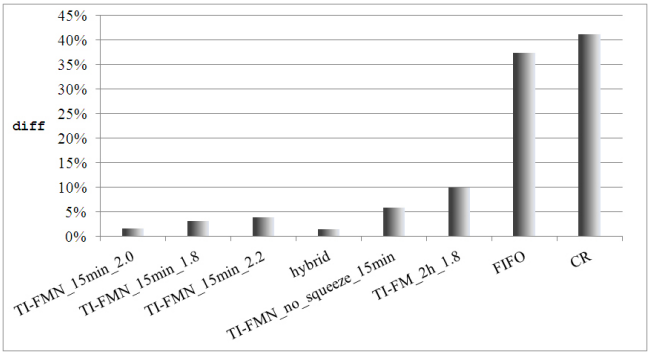


Fig. 6 The average over the twelve coming shift scenarios of the relative differences, `diff`, according to (17), between the objective values obtained by the different scheduling procedures and z^{best2h}.

night shift constraints (10) are disregarded during the first construction phase of the schedules corresponding to the three right-most bars. In order to obtain feasible schedules, the schedules were then run through the squeezing procedure taking the night shifts into account. From these results, it is clear that our proposed scheduling procedure—considering the night shift constraints—finds significantly better schedules within the acceptable time frame (15 minutes). If the proposed hybrid event-driven policy discussed in Section 7.1 is applied—with a rescheduling at the start of each shift—then this difference will become even larger, since the machines will then be available for processing (via the parameter a_k) at times such that the coming unmanned shift can be better utilized (by the jobs that were scheduled but not processed during the previous shift).

In contrast to the computations performed in [35], where the iterative procedure was terminated at a pre-specified value of ℓ^s, here we let the iterative procedure run until a pre-specified time limit. In previous work employing this procedure [34, 35] we considered no night shift constraints, which may have a big impact on a job's completion time, depending on whether the required manual processing of an operation can be scheduled before or after a night shift. In [34] we noted that a near-optimal solution often was found early in the iterative procedure. Due to the addition of the night shift constraints, this property has not been observed while preparing the results presented here; however, in most iterations an improved feasible solution has been achieved. Based on the results presented in Table 5, in which in half of the cases the final objective values from the settings TI-FMN_15min_2.0 and TI-FMN_no_squeeze_2h differ less than 0.5% from z^{best2h}, we conclude that our proposed scheduling method is able to produce near-optimal schedules for the coming shift in the multitask cell.

8 Conclusions

We present a generic iterative procedure for the solution of time-indexed formulations of discrete optimization problems. This procedure is applied to a real flexible job shop scheduling problem originating from a production cell at GKN Aerospace Engine Systems, Sweden. A time-indexed formulation of the problem is presented including side constraints regarding preventive maintenance, fixture availability, and unmanned night shifts. We propose an objective function that minimizes a weighted sum of the jobs' completion times and tardiness. We have developed a (non-generic) squeezing procedure, which can be used in either each or just the last iteration of the iterative procedure. The squeezing procedure reduces the difference between the objective value of the approximate solution obtained from the time-indexed model and the optimal objective value sought.

Computational results show that the gain—in terms of objective values—of including the night shift constraints is around 10%. This gain would be even greater if the proposed scheduling principle with a hybrid event-driven policy is applied: the operations previously scheduled will be rescheduled such that the possibility to fully utilize the first unmanned night shift at the time of rescheduling is increased.

The proposed scheduling procedure has been compared with two priority dispatching rules: the static first–in, first–out (FIFO) rule and a dynamic critical ratio (CR) rule. The choice of these two rules for comparison is based on the facts that FIFO is currently in use in most workshops at GKN and that the CR rule is an existing built-in scheduling method in the control system of the production cell specifically studied in this article. The objective values obtained after running our iterative procedure for 15 minutes were on average 0.71 times the corresponding values computed by the FIFO or CR rules. Hence, there is a large potential in replacing these simple priority dispatching rules by a more sophisticated scheduling principle, such as our time-indexed mathematical optimization model implemented in our iterative procedure. We conclude that our iterative procedure is able to produce near-optimal schedules for industrial data instances for the coming shift within an acceptable time frame.

Acknowledgements This research was financially supported by GKN Aerospace Engine Systems, The Swedish Research Council (grant no. 621-2007-4716), NFFP (National Aviation Engineering Research Programme, grant no. 2009-01281), and VINNOVA (through Chalmers Transport Area of Advance).

References

1. Azem, S., Aggoune, R., Dauzere-Peres, S.: Disjunctive and time-indexed formulations for non-preemptive job shop scheduling with resource availability constraints. In: M. Helander, M. Xie, R. Jiao, K.C. Tan (eds.) Proceedings of 2007 IEEE International Conference on Industrial Engineering and Engineering Management, pp. 787–791 (2007)
2. Van den Bergh, J., Belin, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013)

3. Blackstone Jr., J.H., Phillips, D.T., Hogg, G.L.: A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research **20**, 27–45 (1982)
4. Bowman, E.H.: The schedule-sequencing problem. Operations Research **7**(5), 621–624 (1959)
5. Brucker, P.: The job-shop problem: Old and new challenges. In: P. Baptiste, G. Kendall, A. Munier-Kordon, F. Sourd (eds.) Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007), Paris, France, pp. 15–22 (2007)
6. Brucker, P., Jurisch, B., Krämer, A.: Complexity of scheduling problems with multi-purpose machines. Annals of Operations Research **70**, 57–73 (1997)
7. Brucker, P., Knust, S.: Complex Scheduling, 2nd edn. Springer-Verlag, Berlin, Germany (2012)
8. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms–I. Representation. Computers & Industrial Engineering **30**(4), 983–997 (1996)
9. Demir, Y., İşleyen, S.K.: Evaluation of mathematical models for flexible job-shop scheduling problems. Applied Mathematical Modelling **37**, 977–988 (2013)
10. Dyer, M.E., Wolsey, L.A.: Formulating the single machine sequencing problem with release dates as a mixed integer program. Discrete Applied Mathematics **26**(2–3), 255–270 (1990)
11. Fattahi, P., Saidi Mehrabad, M., Jolai, F.: Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing **18**(3), 331–342 (2007)
12. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Brooks/Cole Publishing Company/Cengage Learning, Belmont, CA, USA (2002)
13. Gao, J., Gen, M., Sun, L.: Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. Journal of Intelligent Manufacturing **17**(4), 493–507 (2006)
14. Haupt, R.: A survey of priority rule-based scheduling. OR Spektrum **11**(1), 3–16 (1989)
15. Hopp, W.J., Spearman, M.L.: Factory Physics, 3d edn. McGraw-Hill/Irwin, New York, NY, USA (2008)
16. IBM Corp.: IBM ILOG CPLEX V12.1 User's Manual for CPLEX. Armonk, NY, USA (2009)
17. Jain, A.S., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research **113**(2), 390–434 (1999)
18. Jansson, T.: Resource utilization in a multitask cell. Master's Thesis, Department of Mathematical Sciences, Chalmers University of Technology, Göteborg, Sweden (2006)
19. Kedad-Sidhoum, S., Rios-Solis, Y., Sourd, F.: Lower bounds for the earliness–tardiness scheduling problem on single and parallel machines. European Journal of Operational Research **189**(3), 1305–1316 (2008)
20. Mainieri, G.B., Ronconi, D.P.: New heuristics for total tardiness minimization in a flexible flowshop. Optimization Letters **7**(4), 665–684 (2013)
21. Manne, A.S.: On the job-shop scheduling problem. Operations Research **8**(2), 219–223 (1960)

22. MATLAB: Release 2011b. The MathWorks Inc., Natick, MA, USA (2011)
23. Ouelhadj, D., Petrovic, S.: A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling **12**(4), 417–431 (2009)
24. Özgüven, C., Özbakir, L., Yavuz, Y.: Mathematical models for job-shop scheduling problems with routing and process plan flexibility. Applied Mathematical Modelling **34**(2), 1539–1548 (2010)
25. Pan, C.-H.: A study of integer programming formulations for scheduling problems. International Journal of Systems Science **28**(1), 33–41 (1997)
26. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 4th edn. Springer, New York, NY, USA (2010)
27. Rahimifard, S., Newman, S.T.: Simultaneous scheduling of workpieces, fixtures and cutting tools within flexible machining cells. International Journal of Production Research **35**(9), 2379–2396 (1997)
28. Rajkumar, M., Asokan, P., Vamsikrishna, V.: A GRASP algorithm for flexible job-shop scheduling with maintenance constraints. International Journal of Production Research **48**(22), 6821–6836 (2010)
29. Slomp, J., Zijm, W.H.M.: A manufacturing planning and control system for a flexible manufacturing system. Robotics and Computer-Integrated Manufacturing **10**(1–2), 109–114 (1993)
30. Sousa, J.P., Wolsey, L.A.: A time indexed formulation of non-preemptive single machine scheduling problems. Mathematical Programming **54**, 353–367 (1992)
31. Stefansson, H., Sigmarsdottir, S., Jensson, P., Shah, N.: Discrete and continuous time representations and mathematical models for large production scheduling problems: A case study from the pharmaceutical industry. European Journal of Operational Research pp. 383–392 (2011)
32. Syberfeldt, A., Karlsson, I., Ng, A., Svantesson, J., Almgren, T.: A web-based platform for the simulation-optimization of industrial problems. Computers & Industrial Engineering **64**(4), 987–998 (2013)
33. Thörnblad, K.: On the optimization of schedules of a multitask production cell. Licentiate Thesis, Chalmers University of Technology and University of Gothenburg, Göteborg, Sweden (2011)
34. Thörnblad, K., Strömberg, A.-B., Patriksson, M., Almgren, T.: A competitive iterative procedure using a time-indexed model for solving flexible job shop scheduling problems. Available at www.optimization-online.org/DB_HTML/2013/08/3991.html (2013)
35. Thörnblad, K., Strömberg, A.-B., Patriksson, M., Almgren, T.: Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance. European Journal of Industrial Engineering, to appear (2015)
36. Wagner, H.M.: An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly **6**, 131–140 (1959)
37. Wang, S., Yu, J.: An effective heuristic for flexible job-shop scheduling problem with maintenance activities. Computers & Industrial Engineering **59**(3), 436–447 (2010)

MISTA 2015

Single Track Train Scheduling

Jonas Harbering · Abhiram Ranade · Marie Schmidt

Abstract In this work we consider the Single Track Train Scheduling Problem. The problem consists in scheduling a set of trains from opposite sides along a single track. The track passes intermediate stations and the trains are only allowed to pass each other at those stations. This problem has a close relation to minimizing the makespan in a job shop scheduling problem with two counter routes and no preemption. We develop a lower bound on the objective value of the train scheduling problem which provides us with an easy solution method in some special cases. The contrast in complexity to the analogous job shop scheduling problem is highlighted. Additionally, we prove the pseudo-polynomial solvability for a more general setting of the train scheduling problem.

1 Introduction

In this paper we consider a scheduling problem which is motivated by a railway application: the scheduling of trains on a single bi-directional track. This problem occurs in passenger transportation in rural areas or when scheduling freight trains, as outlined in [19] and references therein.

In its basic version, the *Single-Track-Train-Scheduling Problem (STTS)* reads as follows: we are given a single track, running from left to right, which has to be passed by P^l trains from the left and P^r trains from the right in the least time possible. The track is divided into several block sections, each block can be occupied by only one train at the same time. However, between the blocks we have stations which have unlimited capacity. Here trains can wait in order to let trains from the opposite direction pass.

Jonas Harbering
Georg-August-University of Göttingen
E-mail: jo.harbering@math.uni-goettingen.de

Abhiram Ranade
Indian Institute of Technology
E-mail: ranade@cse.iitb.ac.in

Marie Schmidt
Erasmus University Rotterdam
E-mail: schmidt2@rsm.nl

The translation to common machine scheduling terminology is quite straightforward: trains correspond to jobs, blocks correspond to machines, the block traversal time corresponds to processing time on a machine, and our objective is to minimize the makespan.

When translating the (STTS) to a machine scheduling problem, we obtain a special case of job-shop scheduling, since we have two subsets of jobs corresponding to trains coming from the left and trains coming from the right, which pass the blocks/machines in reverse order. This is called *job shop scheduling with counter-routes in the machine scheduling context. Furthermore, we have the requirement that a train ride on a block cannot be interrupted, this is referred to as* no preemption *in machine scheduling.* The (STTS) can hence be interpreted as a *job shop scheduling problem with two counter routes and no preemption* (abbreviated as $F^{\pm}||C_{\max}$ following common scheduling notation, see [9]), under the additional assumption that the processing time of a job on a machine is the same for all jobs. This relation is detailed in Section 2.2.

While $F^{\pm}||C_{\max}$ is strongly NP-hard for three machines already [9], the assumption that processing times depend only on the machines (or, to say it in the words of the train scheduling example: that block traversing times are the same for all trains) makes the problem considerably easier. In fact, we are going to show that for any fixed number of blocks, the problem can be solved polynomially in the number of trains and the maximal block length by dynamic programming.

For the case of three blocks and equal train numbers from both sides, as well as for some other special cases, we are even able to prove a closed-form expression for the minimal makespan.

The remainder of the paper is structured as follows. In Section 2 we give a short literature overview on relevant contributions in machine scheduling including some related complexity results and in single-track train scheduling. In Section 3 we formally introduce the problem. In Section 4 we derive a lower bound on the solution value. Using this bound, in Section 5 we discuss special cases where the lower bound can be reached. For the remaining cases we develop a dynamic programming approach in Section 6 and briefly discuss how constraints like limited station capacities and waiting times at stations can be included.

2 Related Problems

2.1 Train Scheduling

The general problem of scheduling or timetabling has many different facets. Since problems arising in train scheduling *on networks* are often of quite different nature, we concentrate on *single-track* train scheduling in this overview.

The problem of scheduling trains on a single track has attracted attention very early already. One of the first to lay a ground for such research is [13]. There, two way traffic systems, similar to the ones considered here, are analyzed. The main difference to our work is that they aim at determining the minimal number of trains that serve a certain train system ensuring periodic schedules. Subsequently, a model allowing for different train speeds and including delays is considered in [22]. There, train systems are analyzed in order to compute the average traveling time per train, depending on the number of trains with different priorities using the same track sequence. Also in [15], delays in train scheduling are considered. The authors aim at a tool which is suitable both for realtime decision support and for timetable evaluation. They also develop a non-linear mixed integer model to minimize the

average weighted travel time which takes into account several different timetabling aspects. Finally, they solve the developed model with a branch-and-bound approach. Similarily, [18] develop a detailed model for scheduling trains on a single track. In their model, the computation of the line capacity is reduced to computing the capacity only on a bottleneck segment. For this bottleneck segment they are able to determine the main relations between input parameters and the capacity of the line. In [25] a mixed integer linear programming model is stated in which trains may start at trains may only pass each other at stations. Different headways are considered for consecutive trains on tracks and at stations. In order to minimize the total travelling time, a branch-and-bound procedure is proposed. Using a similar model to [25], [8] propose a mixed integer linear model with varying departure and traveling times. Additionally, this model takes into account that headways between trains in the same direction are possibly shorter than headways for trains in different directions. They aim at minimizing the total arrival times of all trains at all stations. The model is then solved by a heuristic. Finally, [23] considers a very similar model (to [8,25]) with the objective of minimizing the arrival time of the last train at its destination, i.e. the makespan. Most constraints are adapted from [25] while some problem specific constrains are added in order to decrease computation time. The proposed model is then solved by a heuristic approach. A slightly different problem is discussed in [4]. Here, two trains in opposing directions are scheduled on a two-track segment. Now a part of one of the tracks fails. The question to be answered is how can trains be scheduled on the same track segment in opposing direction without deviating too much from the previous schedule.

See [23] for a broad overview on scheduling on a single bidirectional line and [7,10] for more general train scheduling surveys.

The models in all of the discussed works model real-world train scheduling constraints in varying degree of detail. Our simplified train scheduling problem could, e.g., be considered to be a special case of the models described in [7,8,23,25] and the solution methods described there for more general problems could also be applied to solve our special case. However, to the best of our knowledge, there is so far no complexity analysis for such problems. Hence, the possibility of solving these problems exactly in polynomial time, based, e.g., on combinatorial algorithms or linear programming, has not been ruled out yet. With this work, we aim to lay a ground for filling this gap.

Train scheduling is closely related to machine scheduling. In fact, even more general timetabling problems can be modeled using disjunctive graphs, which are also frequently used to model machine scheduling problems. See [3] for an introduction to this modeling approach and [5] for an application of it to timetabling.

In the next section, we detail how the (**STTS**) can be interpreted in a machine scheduling context and how complexity results from machine scheduling transfer to our problem.

2.2 Relation to Machine Scheduling

The problem (**STTS**) can be interpreted as a machine scheduling problem as follows: Let a set of machines $M_1, \dots, M_n$ and a set of jobs $J^l_1, \dots, J^l_{P^l}, J^r_1, \dots, J^r_{P^r}$ be given. Then the aim is to feasibly schedule the jobs $J^l_1, \dots, J^l_{P^l}$ on the sequence of machines $M_1 - \cdots - M_n$ and the jobs $J^r_1, \dots, J^r_{P^r}$ on the sequence $M_n - \cdots - M_1$ while minimizing the makespan. Note that such a sequence of machines is called a *route* in machine scheduling.

The following conditions must hold for a schedule to be feasible: Jobs can only be processed by one machine at a time, machines can only process one job at a time and once the processing of a job on a machine has started, the job can not be interrupted. This problem is called the *job shop problem with two counter routes and no preemption* ($F^{\pm}//C_{max}$). This abbreviation corresponds to the usual three field representation which is used for classifying machine scheduling problems. It means that the problem is a flow shop problem (F) with two counter routes ($\pm$) and the objective is to minimize the makespan (C_{max}). See [9] for an extensive analysis of flow and job shop problems, including this problem.

In our problem, the trains are represented by the jobs and the blocks are represented by the machines. An important difference between (STTS) and $F^{\pm}//C_{max}$ is that in $F^{\pm}//C_{max}$ the processing times of different jobs on a machine can differ, while in (STTS) we assume all trains i to have the same traversal time p_i on each block. Hence, in the above-mentioned classification scheme for scheduling problems, (STTS) corresponds to the problem $F^{\pm}/p_{ij} = p_i/C_{max}$. Hereby, $p_{ij} = p_i$ means that the processing time p_{ij} of job $j \in \{J^l_1, \ldots, J^l_{P^l}, J^r_1, \ldots, J^r_{P^r}\}$ on machine $i \in \{M_1, \ldots, M_n\}$ is independent of the job. This restriction is well-studied in job-shop scheduling ([14, 16, 20]) but has to the extent of our knowledge not been considered in conjunction with counter routes.

The complexity of $F^{\pm}//C_{max}$ is well researched in the literature: For $n = 2$ the job shop problem with two counter routes and no preemption can be solved in polynomial time. E.g., [17] give an $O\left((P^l + P^r)log(P^l + P^r)\right)$ algorithm for a problem equivalent to $F^{\pm}n//C_{max}$. We conclude that the problem (**STTS**) can be solved in polynomial time for $n = 2$ and that this result would even hold if every train block combination had a different traversal time.

Still, already for $n = 3$ machines, job shop scheduling with two counter routes is NP-hard if processing times on the machines may differ for different jobs. This result immediately follows from the NP-hardness of flow shop scheduling on three machines with only one route [21]. The result even holds for the case that the number of jobs from boths sides are equal, i.e. if $P^l = P^r$, since the special case where all jobs from the right have 0 processing times is again equivalent to flow shop scheduling.

However, this complexity result does not carry over to the (STTS) where the block traversal times are equal for all jobs. Hence, the question of whether (STTS) can be solved in polynomial time or not is still open for the case of three or more blocks.

In [12] an upper bound on the objective value for job shop scheduling with two routes (not necessarily counter routes) on n machines is proven. Also [2] develop an upper bound for the counter routes problem on n machines. In [12] an earlier publication ([1]) on this topic is mentioned, however, we were not able to find this paper. See also the scheduling overview given by [9, 24] for a collection of results on the $F^{\pm}//C_{max}$ problem.

3 Model

In this work we investigate the complexity of and approaches for solving the single track train scheduling problem. This problem is described as follows.

Let a linear graph $G = (V, B)$ with stations $V = \{s_1, \ldots, s_{n+1}\}$ and undirected edges, (called *blocks* in the remainder of this paper) $B = \{b_1 = (s_1, s_2), \ldots, b_n = (s_n, s_{n+1})\}$ be given. In order to facilitate the problem description, we imagine that the tracks are going from left to right.

We define station s_1 to be the leftmost station and station s_{n+1} to be the rightmost respectively.

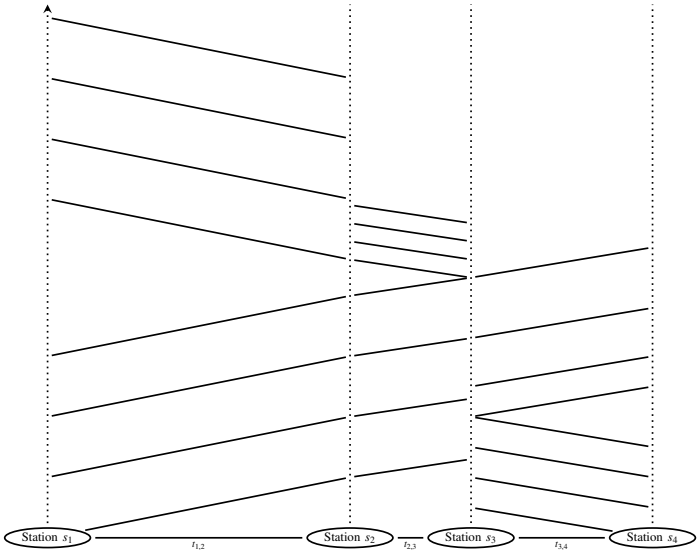


Fig. 1: Example for a space time diagram with $P^l = P^r = 4$ and $n = 4$

The traversal times of the blocks are given as $t_{i,i+1}$ for block $b_i = (s_i, s_{i+1})$ for all $i = 1, \ldots, n$. Waiting times at stations are neglected, i.e., in our model a train can pass a station without stopping. The number of trains traversing the graph from s_1 to s_{n+1} (or from left to right respectively) is specified by P^l and the number of trains traversing G in opposite direction is given as P^r. At any point in time there can at most be one train on each block. The capacity of the stations is not restricted. It means that any number of trains may be stored at any station. The described setting of course oversimplifies reality. However, the intent of this paper is to investigate to which extent the (STTS) is solvable in this very simple setting. Possible extensions and ways to make the model more realistic are discussed in Section 7.

With the above notation we can now formally state the problem of *Single Track Train Scheduling* (**STTS**).

(**STTS**)
Let the graph $G = (V, B)$ with traveling times, station and block capacities be given as above. Then the aim is to minimize the makespan for the traversal of G for P^l trains from left to right and P^r trains from right to left.

In order to illustrate scheduling strategies, we make use of space time diagrams. In such diagrams, the stops of the linear network are denoted on the horizontal axis, the time is given by the vertical axis. Lines in the diagram represent the traversal of trains. Figure 1 depicts such a diagram.

For explanatory purpose we introduce two terms. Suppose a train is at a certain point of the linear network, then we call the remainder of the network, which the train still has to traverse, its *remaining part* of the network. The time-wise distance between two trains (usually on a specific block) is called *headway*.

In the following we discuss how this problem is related to machine scheduling and to what extent complexity results from scheduling theory carry over to (STTS).

4 Lower Bound

In this section we discuss a lower bound on the objective value of the optimal solution for an instance I of (STTS). This value will help us to prove optimality of scheduling strategies for subsequent instances of (STTS).

The lower bound is given by

$$T^{lo}(I) = \max_{i=1,\dots,n} \left\{ \left(P^l + P^r\right) t_{i,i+1} + 2 \min \left\{ \sum_{j=1}^{i-1} t_{j,j+1}, \sum_{j=i+1}^{n} t_{j,j+1} \right\} \right\} \tag{1}$$

Let the block for which the maximum is attained in (1) be called the *bottleneck block*. Later on we will see that in many cases, a good scheduling strategy on the bottleneck block guarantees that the lower bound can be obtained as the makespan.

Lemma 1 *$T^{lo}(I)$ is a lower bound on the objective value of an instance I of (**STTS**).*

Proof Let us first consider one arbitrary block $b_{i,i+1}$. All trains must pass this block which makes up $(P^l + P^r)t_{i,i+1}$ of time. Additionally, the first train that passes the block needs some time to arrive at the block.

The minimum time for a train to arrive at $b_{i,i+1}$ is given by the minimum of the distances of the parts left and right of the block, i.e.

$$\min \left\{ \sum_{j=1}^{i-1} t_{j,j+1}, \sum_{j=i+1}^{n} t_{j,j+1} \right\}$$

Subsequently, all trains pass that block. Still the last train has to reach its destination. The time to reach the destination is again bounded below by the minimum of the distances of the parts left and right of the block, i.e.

$$\min \left\{ \sum_{j=1}^{i-1} t_{j,j+1}, \sum_{j=i+1}^{n} t_{j,j+1} \right\}$$

Summing up all parts we obtain for each block a value which gives a lower bound on the makespan. In particular, the maximal value of those is also a lower bound.

This lower bound will be of help to show that particular cases can be solved in polynomial time.

5 Special Cases

We will now direct to special cases for which we can specify scheduling strategies which allow to reach the lower bound on the makespan.

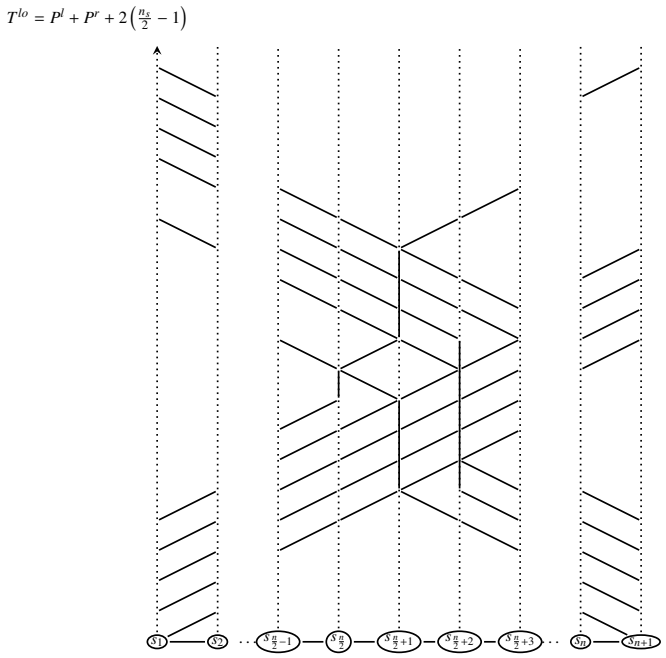


Fig. 2: Strategy for unit processing time and even number of blocks

5.1 Unit processing times and capacity restrictions at stations

Assume that all traversal times on all blocks are equal, i.e. $t_{i,i+1} = 1$ for all $i = 1, \ldots, n$. Under this condition we can find optimal schedules which do not need high station capacity at intermediate stations.

Theorem 1 *For instances of (**STTS**) with unit processing times, the makespan of an optimal schedule is given by $T^{lo}(I)$. This even holds if the capacity at all stations is bounded by three.*

Proof First, assume that no capacity restrictions are enforced. Two different cases will be considered. This is n is even and n is odd.

Consider the case where n is even. The following strategy provides an optimal schedule. See Figure 2 for a sketch of the schedule.

Let all trains from both sides start traversing the linear network at time 0 with a headway of one. If there is a conflict, i.e., if two trains would cross the same block at the same time, always let trains from the left precede those from the right, except for the case that the train from the left is the P^lth - in this case give the trains from the right precedence. After all trains from the left, except the P^lth, have traversed all blocks, all trains from the right traverse the remaining part of the network. After all trains from the right have passed, the P^lth train continues to station s_{n+1}.

The last train from the left traverses the block left of the central station $s_{\frac{n}{2}+1}$ between the first and the second train from the right. Then the train waits at the central station until all trains from the right have passed and finally heads for its destination s_{n+1}. We now have to show that indeed the lower bound $T^{lo}(I)$ is obtained with this strategy.

The first train from the left and from the right reach the central station $s_{\frac{n}{2}+1}$ at the same time $\frac{n}{2}$. After that all trains from the left traverse the central part heading for station $s_n + 1$, in particular the block right of the central station. Between time $\frac{n}{2}$ and time $\frac{n}{2} + P^l - 1$, the trains $1, 2, 3, \ldots, P^l - 1$ from the left cross the block right of the central station one after another and proceed to the right. In the meantime trains from the right have started from station s_{n+1} and traversed as many blocks as possible.

At station $s_{\frac{n}{2}+1}$ there is only one train waiting. At stations $s_{\frac{n}{2}+2}, s_{\frac{n}{2}+3}, \ldots, s_n$ there are two trains waiting as long as there are trains, the remaining trains wait at station s_{n+1}.

At time $\frac{n}{2} + P^l - 2$ the first train from the right starts traversing its remaining part of the linear network. Since then there are no more trains from the right waiting at station $s_{\frac{n}{2}+1}$ and the block $b_{\frac{n}{2}+1,\frac{n}{2}+2}$ is used by the penultimate train from the left for one more time unit, the next train heading for the left has a headway of two to the first train. Hence, in this gap, the last train from the left traverses the block $b_{\frac{n}{2},\frac{n}{2}+1}$ and thus waits at station $s_{\frac{n}{2}+1}$. Finally, all trains from the right traverse their remaining part of the network with a headway of one, including those which were waiting at the station s_{n+1}. Once the last train from the right has arrived at the central station $s_{\frac{n}{2}+1}$ (at time $\frac{n}{2} + P^l + P^r - 1$) both this train and the last train from the left finish their ride to their final station. This ride takes each train $\frac{n}{2}$ time units.

If we now consider block $b_{\frac{n}{2}+1,\frac{n}{2}+2}$ we find the following sequence of trains. Until time $\frac{n}{2} - 1$ there is no train. Then the first train from the right passes this block. After that all trains from the left but the last one traverse this block. Subsequently, all trains from the right but the first pass this block. Finally, the last train passing this block is the last train from the left. Then the block is empty until the last train from the left reaches its destination.

It is easy to see that no conflicts appear on the other blocks.

In order to determine the makespan of this schedule, we consider the block $b_{\frac{n}{2}+1,\frac{n}{2}+2}$. This block is empty until time $\frac{n}{2} - 1$. Afterwards, it is occupied during $P^l + P^r$ time units: Between time $\frac{n}{2} - 1$ and time $\frac{n}{2}$, the first train from the right passes the block. After that all trains from the left but the last one traverse the block until time $\frac{n}{2} + (P^l - 1)$. Subsequently, all trains from the right but the first pass the block until time $\frac{n}{2} + (P^l - 1) + (P^r - 1)$. Finally, the last train passing this block is the last train from the left which arrives at station $\frac{n}{2} + 2$ at time $\frac{n}{2} + P^l + P^r - 1$. Consequently, the last train from the right arrives at the leftmost station s_1 at time $\frac{n}{2} + (P^l - 1) + (P^r - 1) + \frac{n}{2} = P^l + P^r + 2\left(\frac{n}{2} - 1\right) = T^{lo}(I)$ and the last train from the left arrives at the rightmost station s_{n+1} at time $\frac{n}{2} + P^l + P^r + (\frac{n}{2} - 1) = P^l + P^r + 2\left(\frac{n}{2} - 1\right) = T^{lo}(I)$.

To see the second part of the lemma, note that at each station at each point in time there are at most two trains from the right and one from the left. Hence this schedule can be operated with a maximal station capacity of three.

Now assume that n is odd. The strategy in this case is the following. Let all trains from both side traverse the linear network with a headway of one. If there is any conflict between two trains from different directions always let trains from the left precede.

Here we find that the central block $b_{\frac{n+1}{2},\frac{n+1}{2}+1}$ is reached by a train from each side at time $\frac{n-1}{2}$. Subsequently, all trains from the left side traverse the central block followed by all trains from the right side with headway one. Note that this is only possible since at any station from $s_{\frac{n+1}{2}+1}$ to the right there are two trains waiting. Finally, the last train has traversed the central block at time $\frac{n-1}{2} + P^l + P^r$ and it takes another $\frac{n-1}{2}$ time units until it reaches the final station s_1. Hence the makespan equals $P^l + P^r + 2\left(\frac{n-1}{2}\right)$ which equals the lower bound $T^{lo}(I)$.

To see the second part of the theorem, note that at no point in time there are more than two trains waiting at a station and a third one passing in opposing direction. Hence a maximal needed capacity of three is obtained.

If, additionally, the number of trains from left to right and from right to left are equal, even lower station capacity is sufficient to obtain an optimal solution.

Lemma 2 *For instances of (STTS) with unit processing times and $P^l = P^r$, the makespan of an optimal schedule is given by $T^{lo}(I)$. This even holds if the capacity at all stations is bounded by two.*

Proof Denote $P := P^l = P^r$. Again we separate the analysis into the cases where n is even and n is odd. First assume n is even.

The strategy consists in scheduling all trains from the left and the right at the same time with a headway of two. Then, since n is even two passing trains always meet at a station and never face a conflict on the tracks, hence, no train has to wait at intermediate stations. Consequently, the makespan is $n + 2(P-1) = 2P + 2\left(\frac{n}{2} - 1\right) = T^{lo}(I)$.

In case n is odd, let the first train from the left start at time 0 and again let subsequent trains from the left keep a headway of two to the preceding train. Furthermore, let the first train from the right start at time 1 and let subsequent trains from the right keep a headway of two to the preceding train as well. Then, two passing trains always meet at a station and never face a conflict on the tracks, hence, no train has to wait at intermediate stations. Consequently, the last train from the right arrives at time $1+n+2(P-1) = 2P+2\left(\frac{n-1}{2}\right) = T^{lo}$, the last train from the left arrives at time $n + 2(P-1) = 2P + 2\left(\frac{n-1}{2}\right) - 1 < T^{lo}$.

Note that in both cases, no trains have to wait and that at most two trains are passing each other in stations. Hence, the capacity needed equals two.

We conclude this section with a remark on the situation with unit capacity which also holds if block lengths are not equal. In this section, the lower bound T^{lo} cannot be reached. Since trains cannot pass each other, the optimal strategy simply consists of letting the trains from one side pass first, and then the trains from the other side. This leads to the following lemma.

Lemma 3 *If the station capacity is restricted to one, the makespan of an optimal schedule is given by $2\sum_{i=1}^{n} t_{i,i+1} + (P^l + P^r - 2)\max_{i=1}^{n} t_{i,i+1}$.*

5.2 Restricting the Number of Blocks

In the following we consider arbitrary track lengths, but restrict the number of blocks on the track. From the machine scheduling analysis we know that the case $n = 2$ is easily solvable. Hence, we direct to the case where $n = 3$. Take Figure 3 as an illustration of the considered case.

Here, we can find a polynomially solvable case which is given by the following theorem.

Theorem 2 *For instances I of (**STTS**) with three blocks and $P = P^l = P^r$, the makespan of an optimal schedule is given by $T^{lo}(I)$.*

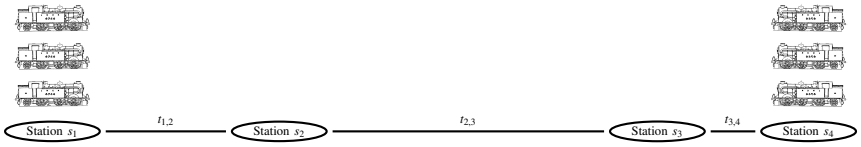


Fig. 3: Example setting for $n = 3$

Proof $T^{lo}(I)$ equals the largest of the three bounds $\overbrace{2Pt_{12}}^{(1)}$, $\overbrace{2Pt_{23} + 2\min\{t_{12}, t_{34}\}}^{(2)}$, $\overbrace{2Pt_{34}}^{(3)}$.

Suppose bound (2) is largest. Then, assume wlog. that $t_{12} \geq t_{34}$. We first construct a schedule with makespan T^{lo} blockwise, then we show its validity. The first half of the schedule is composed as follows. On block b_{12}, we have P trains leaving consecutively (i.e., with headway t_{12}) to the right. Block b_{23} is initially unused until time t_{34}. After that P trains go alternately and consecutively: first to left and then to right until all trains have passed this block at time $t_{12} + 2Pt_{23}$. On block b_{34}, starting at time zero, trains go left consecutively.

The second half of the schedule is described backwards from the time $T^{lo}(I)$. On block b_{12} we have left-going trains arriving consecutively at station s_1 (i.e., they arrive at times T^{lo}, $T^{lo} - t_{12}$, $T^{lo} - 2t_{12}$,...). Block b_{23} is unused for the last duration of t_{34}. Before that we have alternately moving trains, the last rightward, the second last leftward and so on, as described above. On block b_{34} we have right-going trains arriving consecutively at times T^{lo}, $T^{lo} - t_{34}$, $T^{lo} - 2t_{34}$,... at station s_4. Note that on blocks b_{12} and b_{34} there might be unused times around the middle of the schedule.

We now argue that this is a valid schedule, i.e.,

- no two trains are in the same block at any time and
- no train is scheduled to depart from a station before it has arrived there.

We only discuss this for the first half of the schedule; the second half is analogous. The first half is $Pt_{23} + t_{34}$ long. Clearly the P movements scheduled for block b_{23} fit in this duration, after the initial inactive period of length t_{34}. Because bound (2) is largest we know that $Pt_{12}, Pt_{34} \leq Pt_{23} + t_{34}$. Thus the P rightward (leftward) movements on block b_{12} (b_{34}) can also be accommodated, and there are no two trains at the same block at any time.

Next, as described in the schedule above, the ith rightward train must leave block b_{12} at time it_{12} and enter block b_{23} at $t_{34} + (2i-1)t_{23}$. But since $t_{12} \leq t_{23} + \frac{t_{34}}{P}$, we have that

$$it_{12} \leq it_{23} + i\frac{t_{34}}{P} \leq (2i-1)t_{23} + t_{34}.$$

Thus, the train is scheduled to enter block b_{23} only after leaving block b_{12}.

Likewise, the ith leftward train is scheduled to leave block b_{34} at it_{34} and to enter block b_{23} at $(2i-2)t_{23} + t_{34}$. This is possible since it holds that

$$it_{34} = t_{34} + (i-1)t_{34} \leq t_{34} + (i-1)t_{12} \leq t_{34} + (i-1)(t_{23} + \frac{t_{34}}{P}) \leq t_{34} + (2i-2)t_{23}.$$

The last inequality in this is derived as follows. Note first that if $P = 1$ then $i = 1$ and the proof is direct. Hence consider $P > 1$. We know that $Pt_{34} \leq Pt_{23} + t_{34}$ and thus $t_{23} \geq t_{34}(P-1)/P$. This results $t_{23} \geq \frac{t_{34}}{P}$ for $P > 1$.

Next suppose bound (1) is the largest. For ease of argument, we increase t_{23} until bound (2) equals bound (1). Then, we construct the same schedule as given above. This schedule can easily be transferred to a schedule for lower values of t_{23}. In the schedule, certain time

intervals are reserved for movements on block b_{23}. Of each such interval we only use a subinterval of length equal to the original value of t_{23}. This clearly does not change the makespan and maintains validity, i.e. trains use distinct time intervals on each block and are still in order in which they appear in each block.

The case of bound (3) being the largest is equivalent to the case of bound (1) being largest.

5.3 Can the lower bound $T^{lo}(I)$ always be achieved?

The following example shows a case for $n = 5$ in which the lower bound $T^{lo}(I)$ can not be achieved.

Example 1 Assume $P^l = P^r = 2$ and $t_{1,2} = t_{5,6} = 10$, $t_{2,3} = t_{4,5} = 3$ and $t_{3,4} = 4$. Then $T^{lo}(I) = (P^l + P^r)t_{3,4} + 2(t_{1,2} + t_{2,3}) = 42$ and the bottleneck is block $b_{3,4}$. Figure 4 shows an optimal scheduling strategy. In this strategy there is an unavoidable gap of 2 time units on the bottleneck block between the first and the second train from the right. Due to this gap, the makespan of this solution is $44 = T^{lo} + 2$.

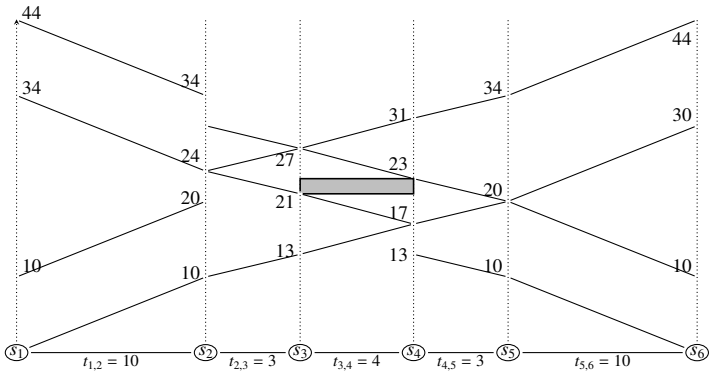


Fig. 4: Sketch for scheduling strategy for Example 1

As a general observation, we conclude that in order to reach the lower bound T^{lo}, we must be able to schedule the trains on the bottleneck block(s) without leaving a gap.

Observation 3 *If there exists a* $k = 1, 2, \ldots, 2P$ *such that at time* $\min\left\{\sum_{j=1}^{i-1} t_{j,j+1}, \sum_{j=i+1}^{n} t_{j,j+1}\right\} + (k-1)t_{jj+1}$ *less than k trains have reached the bottleneck block, there does not exist a schedule which achieves makespan* T^{lo}.

Note that this can occur, even if the longest block is the bottleneck block, as can be seen in the following example.

Example 2

Let $P^l = P^r = 2$ and $(t_{1,2}, t_{2,3}, \ldots, t_{8,9}) = (5, 2, 1, 1, 1, 1, 4)$. Then the bottleneck block is obtained as $b_{1,2}$ with $T^{lo} = 20$. When scheduling, we see that once the second train from left has passed the bottleneck, the first train from the right has not yet reached the bottleneck. Hence the lower bound is not reached. See Figure 5 where the scheduling strategy is depicted.

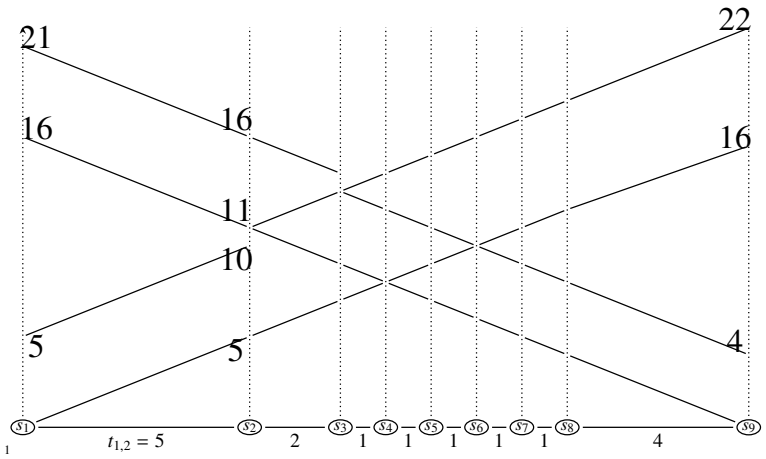


Fig. 5: Lower bound T^{lo} not reached even though longest block is bottleneck

6 Dynamic Programming

In this section we develop a dynamic programming formulation for the problem (STTS) with pseudo-polynomial running time.

Lemma 4 *If block lengths are integer, (STTS) can be solved as a shortest-path problem in an acyclic graph.*

Proof Denote by $L := \sum_{i=1}^{n} t_{i,i+1}$ the total length of the track considered. We divide the tracks into L subblocks c_k of lengths 1 with substations s'_k at every end of a subblock. We denote by I the index set of the substations which are stations and by $I(b_j)$ the index set of the substations corresponding to block b_j, including the stations at both ends of the block. Note that each substation index i belonging to a station s_j (except for $j = 1$ and $j = n + 1$) is contained in two sets $I(b_{j-1})$ and $I(b_j)$.

States: The nodes of the graph in which we are going to formulate the shortest path problem denote the states of the problem. Each state is represented by a tuple X with entries

$$X := (x_0^l/x_0^r, x_1^l/x_1^r, \ldots, x_L^l/x_L^r),$$

also written as

$$X := \frac{x_0^l \,|\, x_1^l \,|\, x_2^l \,|\, \ldots \,|\, x_{L-1}^l \,|\, x_L^l}{x_0^r \,|\, x_1^r \,|\, x_2^r \,|\, \ldots \,|\, x_{L-1}^r \,|\, x_L^r}. \tag{2}$$

Herein, x_i^l represents the number of trains from the left which have passed or reached substation s'_i already, and x_i^r represents the number of trains from the right which have passed or reached substation s'_i already.

For $n \in \mathbf{N}$ we denote $[n] := \{0, 1, 2, \ldots, n\}$.

Feasible states: Since we have P^l trains from the left and P^r trains from the right, $X^l := (x_0^l, x_1^l, \ldots, x_L^l) \in [P^l]^{L+1}$ and $X^r := (x_0^r, x_1^r, \ldots, x_L^r) \in [P^r]^{L+1}$. However, not all vectors $Y \in [P^l]^{L+1} \times [P^r]^{L+1}$ define feasible states of the problem:

- At the beginning, all trains passing from left are on the left of the track, i.e., $x_0^l = P^l$ for all feasible states X, analogously $x_L^r = P^r$ for all feasible states X.

- Since trains pass from left to right, we have $x_i^l \geq x_{i+1}^l$ for all $i = 0, \ldots, L$, analogously $x_i^r \leq x_{i+1}^r$.
- Only one train can be on a block at a time.

$$(\max_{i \in I(b_j)} x_i^l - \min_{i \in I(b_j)} x_i^l) + (\max_{i \in I(b_j)} x_i^r - \min_{i \in I(b_j)} x_i^r) \leq 1 \quad \text{for all } j = 1, \ldots, n$$

Only nodes representing feasible states will be introduced.

Transitions/arcs Two nodes $(X, \hat{X})$ are connected by a directed arc of length 1, if state $\hat{X}$ can be reached from state X in time 1. This is the case, if

- $x_i^k \leq \hat{x}_i^k$ for all $i = 0, \ldots, L$, $k \in \{l, r\}$, i.e., over time, more and more trains pass every substation of the network.
- $\sum_{i \in I(b_j)} (\hat{x}_i^l + \hat{x}_i^r) \leq \sum_{i \in I(b_j)} (x_i^l + x_i^r) + 1$ for all $j = 1, \ldots, n$, i.e., on each block there is at most one train running and it advances at most one substation.
- If for two substations s_i' and s_{i+1}', belonging to the same block b_j, $x_i^l = x_{i+1}^l + 1$, then it follows $\hat{x}_{i+1}^l = x_{i+1}^l + 1$, unless s_i' is a station. Analogously, $x_i^r = x_{i-1}^r + 1$, then $\hat{x}_{i-1}^r = x_{i-1}^r + 1$, unless s_{i+1}' is a station. This models that trains cannot stop between the stations.

These conditions ensure that the graph is acyclic.

Correspondence of solutions to paths There is a one-to-one correspondence between feasible schedules for the (STTS) and paths from node

$$\begin{array}{|c|c|c|c|c|c|} \hline P^l & 0 & 0 & \ldots & 0 & 0 \\ \hline 0 & 0 & 0 & \ldots & 0 & P^r \\ \hline \end{array}$$

to node

$$\begin{array}{|c|c|c|c|c|c|} \hline P^l & P^l & P^l & \ldots & P^l & P^l \\ \hline P^r & P^r & P^r & \ldots & P^r & P^r \\ \hline \end{array}$$

where the length of the path represents the time duration to execute the solution.

Hence, an optimal solution to (STTS) corresponds to a shortest path is the graph.

Lemma 5 *If block lengths are integer, the graph described in Lemma 4 has* $O\left(\binom{P^l+n}{n}\binom{P^r+n}{n} \max\{t_{i,i+1}\}\right)$ *nodes and* $O\left(3^n\binom{P^l+n}{n}\binom{P^r+n}{n} \max\{t_{i,i+1}\}\right)$ *arcs.*

Proof As stated in the proof of Lemma 4, the nodes of the graph correspond to feasible states X (as described in (2)). We now make an attempt on bounding the number of feasible states from above.

We denote by y_j^l the number of trains which have passed the full block b_j from left to right. y_j^r analogously denotes the number of trains which have passed the full block b_j from right to left. I.e, $y_j^l = x_{i_{\text{last}^j}}^l$ and $y_j^r = x_{i_{\text{first}^j}}^l$, where i_{first^j} and i_{last^j} denote the first and the last index in the set $I(b_j)$, respectively.

For every pair (y_j^l / y_j^r) there are $2t_{j,j+1} - 1$ feasible combinations of subindices:

$$\begin{array}{|c|c|c|c|c|} \hline y_j^l = x_{i_{\text{first}^j}}^l & x_{i_{\text{first}^j}+1}^l & x_{i_{\text{first}^j}+2}^l & \ldots & x_{i_{\text{last}^j}-1}^l & y_{j+1}^l = x_{i_{\text{last}^j}}^l \\ \hline y_j^r = x_{i_{\text{first}^j}}^r & x_{i_{\text{first}^j}+1}^r & x_{i_{\text{first}^j}+2}^r & \ldots & x_{i_{\text{last}^j}-1}^r & y_{j+1}^r = x_{i_{\text{last}^j}}^r \\ \hline \end{array},$$

namely

$$\begin{array}{|c|c|c|c|c|c|} \hline y_j^l & y_j^l - 1 & y_j^l - 1 & \ldots & y_j^l - 1 & y_j^l - 1 \\ \hline y_j^r & y_j^r & y_j^r & \ldots & y_j^r & y_j^r \\ \hline \end{array}, \begin{array}{|c|c|c|c|c|c|} \hline y_j^l & y_j^l & y_j^l - 1 & \ldots & y_j^l - 1 & y_j^l - 1 \\ \hline y_j^r & y_j^r & y_j^r & \ldots & y_j^r & y_j^r \\ \hline \end{array}, \ldots, \begin{array}{|c|c|c|c|c|c|} \hline y_j^l & y_j^l & y_j^l & \ldots & y_j^l & y_j^l \\ \hline y_j^r & y_j^r & y_j^r & \ldots & y_j^r & y_j^r \\ \hline \end{array},$$

and

$$\frac{y_j^l \mid y_j^l \mid y_j^l \mid \dots \mid y_j^l \mid y_j^l}{y_j^r - 1 \mid y_j^r - 1 \mid y_j^r - 1 \mid \dots \mid y_j^r - 1 \mid y_j^r}, \dots, \frac{y_j^l \mid y_j^l \mid y_j^l \mid \dots \mid y_j^l \mid y_j^l}{y_j^r - 1 \mid y_j^r \mid y_j^r \mid \dots \mid y_j^r \mid y_j^r}, \frac{y_j^l \mid y_j^l \mid y_j^l \mid \dots \mid y_j^l \mid y_j^l}{y_j^r \mid y_j^r \mid y_j^r \mid \dots \mid y_j^r \mid y_j^r},$$

(one case is listed twice here), if $y_j^l, y_j^r > 0$. If one or both values are 0, there are less.

Furthermore, there are

$$\sum_{i_n=0}^{P^l} \sum_{i_{n-1}=0}^{i_n} \sum_{i_{n-2}=0}^{i_{n-1}} \dots \sum_{i_1=0}^{i_2} 1 \cdot \sum_{i_n=0}^{P^r} \sum_{i_{n-1}=0}^{i_n} \sum_{i_{n-2}=0}^{i_{n-1}} \dots \sum_{i_1=0}^{i_2} 1$$

feasible combinations of the y-values. Each nested sum

$$\sum_{i_n=0}^{P^l} \sum_{i_{n-1}=0}^{i_n} \sum_{i_{n-2}=0}^{i_{n-1}} \dots \sum_{i_1=0}^{i_2} 1 = \binom{P^l + n}{n}$$

then gives a binomial coefficient, see [6] for an explanation. This leads to $O\left(\binom{P^l+n}{n}\binom{P^r+n}{n} \max\{t_{i,i+1}\}\right)$ nodes.

To count the number of arcs, let us consider possible successors $\hat{X}$ of node X in the graph. For each block, there are at most three different possibilities: either a train is moving from left to right, a train is moving from right to left, or no train is moving at all on that block (as formalized when defining the arcs above). Since we have n blocks, each node X has hence at most 3^n successors (and most will have much less). Thus, there are at most $O\left(3^n\binom{P^l+n}{n}\binom{P^r+n}{n} \max\{t_{i,i+1}\}\right)$ arcs in the graph.

Lemma 6 *The (STTS) can be solved in* $O\left(3^n\binom{P^l+n}{n}\binom{P^r+n}{n} \max\{t_{i,i+1}\}\right)$.

Proof Since the described graph is a directed and acyclic graph, we can find a shortest path in linear time in the number of edges, see, e.g., [11].

Of course, for large numbers of blocks n this is impractical and we would probably be better off using an integer programming approach as described, e.g., in [7]. However, from a theoretical point of view, the following conclusion is interesting:

Corollary 1 *For a fixed number of blocks, the (STTS) can be solved in pseudo-polynomial time, i.e., it is polynomial in* P^l, P^r *and* $\max_j t_{j,j+1}$.

Possible Extensions In the basic version of the (STTS) we assumed that stations have an unlimited number of tracks. However, limited station capacity could easily be taken into account in the dynamic programming approach. Note that for each state X of the dynamic program, the number of trains currently halting at station s_j is

$$H_j(X) := \underbrace{x^l_{i^j_{first}} - x^l_{i^j_{first}+1}}_{\text{trains from the left}} + \underbrace{x^r_{i^j_{first}} - x^r_{i^j_{first}-1}}_{\text{trains from the right}}.$$

Hence, track capacity restrictions at stations can be included by creating only the nodes which satisfy $H_j(X) \leq$ capacity of station s_j and the corresponding arcs.

We can also include simple vehicle scheduling constraints in the model. I.e., consider the situation that the trains go back and forth and traverse the track several times during a day. In this case, we would denote by P^l the number of departures from left and by P^r the number of departures from right. Introducing only nodes X which fulfill

$$x_2^l - x_1^r \leq \text{initial number of trains on left}$$

we would make sure that the number of trains which have left the left station never exceeds the number of trains which have arrived there by more than the initial number of trains on the left. It makes sure that there is always a train to leave when there is a scheduled departure. An analogous constraint can be added for the station on the right.

A similar approach could be taken when introducing simple 'balance' constraints. Particularly when transporting passengers, it could be considered unbalanced and unfair, if a large number of trains from the left could pass the full track unhindered while all trains from the right had to wait. To avoid this problem, we could, similarly to the approach described above, exclude all nodes modeling states where, e.g., x_L^l and x_0^r are too different, w.r.t. to the above stated balancedness, or where the difference between x_i^l and x_i^r at some intermediate substation s_i' is too big.

Note that all extensions described so far lead to a smaller network and hence would speed-up the dynamic programming approach.

Other extensions, like minimal waiting times at stations or different train types with different speed profiles could also be included, but *more* states (that is: more nodes) would be needed. However, the general theoretic result of pseudopolynomial solvability for a fixed number of blocks would remain valid also in these cases.

More sophisticated extensions like, e.g., periodicity or routing in stations seem to be out of the scope of this approach.

7 Conclusion

In this paper we studied a basic version of the Single Track Train Scheduling Problem, which can be considered a special case of job shop scheduling with two counter routes and no preemption. Our special case, furthermore, restricts the processing time of a job on a machine to be the same for all jobs.

We were able to prove a lower bound on the minimum makespan of the (STTS) and identify several special cases where this lower bound is tight.

In particular, we found that, although the job shop scheduling problem with two counter routes, no preemption, three machines and equal number of jobs from both sides is strongly NP-hard, that, if we have the additional requirement that the processing times of all jobs on each machine are equal (as it is the case in the (STTS)), we can specify scheduling strategies which reach a makespan equal to the lower bound.

Furthermore, we showed that for any fixed number of blocks (STTS) can be solved in pseudo-polynomial time.

This result can also be generalized for some less basic versions of single track train scheduling which even have the potential to speed-up the algorithm considerably. However, the computation time of our approach increases exponentially in the number of blocks, hence the result is more of a theoretical value and most likely not applicable in real-world train scheduling where the number of blocks is typically large.

In the general cases the lower bound may help to prove optimality or give a good estimate of distance to optimality when applying heuristic solution methods.

To what extent the results in this work can be generalized and the lower bound can be improved is currently under research.

References

1. Babushkin, A., Bashta, A., Belov, I.: Scheduling for the problem with oppositely directed routes. Kibernetika **7**, 130–135 (1974)
2. Babushkin, A., Bashta, A., Belov, I.: Scheduling for problems of counterroutes. Cybernetics and Systems Analysis **13**(4), 611–617 (1977)
3. Balas, E.: Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Operations research **17**(6), 941–957 (1969)
4. Brucker, P., Heitmann, S., Knust, S.: Scheduling railway traffic at a construction site. Springer (2005)
5. Burdett, R.L., Kozan, E.: A disjunctive graph model and framework for constructing new train schedules. European Journal of Operational Research **200**(1), 85–98 (2010)
6. Butler, S., Karasik, P.: A note on nested sums. Journal of Integer Sequences **13**(2), 3 (2010)
7. Cacchiani, V., Toth, P.: Nominal and robust train timetabling problems. European Journal of Operational Research **219**(3), 727–737 (2012)
8. Castillo, E., Gallego, I., Ureña, J.M., Coronado, J.M.: Timetabling optimization of a single railway track line with sensitivity analysis. TOP **17**(2), 256–287 (2009)
9. Chen, B., Potts, C., Woeginger, G.: A Review of Machine Scheduling: Complexity, Algorithms and Approximability. Handbook of Combinatorial Optimization (1998)
10. Cordeau, J.F., Toth, P., Vigo, D.: A survey of optimization models for train routing and scheduling. Transportation Science **32**(4), 380–404 (1998)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., et al.: Introduction to algorithms, vol. 2. MIT press Cambridge (2001)
12. Dushin, B.: An algorithm for the solution of the two-route johnson problem. Cybernetics and Systems Analysis **24**(3), 336–343 (1988)
13. Frank, O.: Two-way traffic on a single line of railway. Operations Research **14**(5), 801–811 (1966)
14. Gonzalez, T.: Unit execution time shop problems. Mathematics of Operations Research **7**(1), 57–66 (1982)
15. Higgins, A., Kozan, E., Ferreira, L.: Optimal scheduling of trains on a single line track. Transportation Research Part B: Methodological **30**(2), 147–161 (1996)
16. Hromkovič, J., Mömke, T., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. Algorithmic Operations Research **2**(1) (2007)
17. Jackson, J.: An extension of johnson's result on job lot scheduling. Naval Research Logistics Quarterly (1956)
18. Janić, M.: Single track line capacity model. Transportation Planning and Technology **9**(2), 135–151 (1984)
19. Kraay, D., Harker, P.T., Chen, B.: Optimal pacing of trains in freight railroads: Model formulation and solution. Operations Research **39**(1), pp. 82–99 (1991)
20. Lenstra, J.K., Kan, A.R.: Computational complexity of discrete optimization problems. Annals of Discrete Mathematics **4**, 121–140 (1979)
21. M. R. Garey, D.S.J., Sethi, R.: The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research (1976)
22. Petersen, E.: Over-the-road transit time for a single track railway. Transportation Science **8**(1), 65–74 (1974)
23. Rahman, S.A.A.: Freight train scheduling on a single line network. Ph.D. thesis, The University of New South Wales (2013)
24. Sevast'janov, S.: On some geometric methods in scheduling theory: a survey. Discrete Applied Mathematics **55**(1), 59 – 82 (1994)
25. Zhou, X., Zhong, M.: Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. Transportation Research Part B: Methodological **41**(3), 320–341 (2007)

MISTA 2015

An Optimization Framework for Job-shop with Energy Threshold Issue With consideration of machining operations with consumption peaks

S. Kemmoe-Tchomte • D. Lamy • N. Tchernev

Abstract In this paper the problem of the Job-shop is extended to support energy constraints. The objective is to propose scheduling tools for manufacturing systems considering consumption threshold that must not be exceeded. The operations are supposed to consume more energy at beginning and thus representing a consumption peak that is often present in machine tools. This assumption results in considering that an operation is divided into two sub-operations. The goal is then to propose the best schedule considering the energy threshold, the consumptions of operations and duration of consumption peaks as given data. A Mixed Integer Linear Model (MILP) for the problem solving is proposed; it is based on flow approach to take into account the energy threshold. Since it is difficult to find exact solutions for medium and large size problems, a metaheuristic based on a GRASPxELS is proposed. Small scale instances for the problem have been generated, and results expose the relevance of the metaheuristic approach.

1 Introduction

For several years, environmental and energy constraints have become essential criteria in decision making inside enterprises or laboratories [11]. This is also one of the major issues for governments, who care of the ecological impact of the industrial sector on the planet and on society. Thus, constraints such as footprint carbon and energy (fossil or not) consumption are now seriously taken into account. According to the International Energy Outlook proposed by the U.S. Energy Information Administration, companies consumed more than 50% of the global

Sylverin Kemmoe-Tchomte
CRCGM EA 3849, Université d'Auvergne, Clermont-Ferrand, France
E-mail: sylverin.kemmoe_tchomte@udamail.fr

Damien Lamy
LIMOS UMR CNRS 6158, Aubière, France
E-mail: lamy@isima.fr

Nikolay Tchernev
LIMOS UMR CNRS 6158, Aubière, France
E-mail: tchernev@isima.fr

delivered energy in 2010 [8]. However they tend to apply their good practices to their different sites and to their suppliers by banishment of some materials for example, which could lead to the ISO14000 certification [11]. More than 64% of the allocated energy to the industrial sector was used by non-OECD industry. However, companies' answers to this problem are still limited. Without a doubt, according to Rager et al. [23] to provide solutions to this problem, two types of measures could be taken: technological and/or organizational. Technological measures are quite expensive since they emphasis on new machines or production process, whereas organizational measures focus on improvements for the existing system, thus leading to energy efficiency. A lot of enhancements can be made on this last point in order to obtain better results, meeting industrial expectations and without investing in new machines. According to [7] three decisions could be made to build an energetically responsible production system. At first, the energy used by inactive machines could be minimized (by switching them off/on or by reducing lazy times). Another solution consists in moving activities from On-peak hours to Off-peak hours. Finally, it is possible to avoid consumptions peak which could lead to an overbilling. In this paper, this last point is treated in a Job-shop like manufacturing environment where a near optimal schedule should be proposed while an energy consumption threshold should not be exceeded. From the best of our knowledge, it is one of the first researches focusing on this problem.

In the next section, a literature review of articles concerning scheduling problems with energy constraints is proposed. In the third section, assumptions used in this study are presented. In the fourth section, the mathematical model of the problem is given. In the fifth section, a GRASPxELS metaheuristic is introduced in order to obtain near optimal solutions of the problem. Results for small scale instances are presented in section six. Finally, the last section consists in a conclusion and research perspectives

2 Related work

The literature is full of industrial problems consisting of minimizing the total treatment time, also called makespan, and other objective functions. Until recently, only a few works dealt with energy optimization as an important constraint in scheduling. However, the "Green Manufacturing" field of research is increasingly studied and hence a non-exhaustive review is proposed in this section.

[19] proposed methods and operational tools to minimize the energy consumption of a single machine. A mathematical formulation to minimize simultaneously the total completion time of a set of operations and the total energy consumption is proposed. Their model handles the different states a machine can be in: idle, running, switch ON or OFF. Later the same authors [20] extend their previous work and proposed a Greedy Randomized Adaptive Search Procedure (GRASP) which objective is to find a solution minimizing both the total energy used and the total tardiness.

[6] used a Genetic Algorithm associated to a Simulated Annealing in order to provide approach solutions to a Flow-shop problem with consideration of total energy consumption. The power used by the machines according to their state is taken into account. They suggest to turn on/off machines according to the need of the production system and respecting conditions. Thus, a machine will not be turned off if the next operation to be scheduled starts earlier than the duration of the turn off/on process. Their results are given in a Pareto graphic. However, they did not use industrial data and underlined that their model does not handle the possible breakdowns. Considering this last point, [25] noticed that an energy efficient system provides robustness and is less sensitive to breakdowns and thus they worked on the correlation between makespan, energy and robustness. In their model a machine may have variable speeds for processing operations. A machine which is processing a task quickly will consume more energy and the treatment time will be reduced, however if the machine is processing the operation more slowly, the consumption will be reduced. In this context if a breakdown occurs, the lost time

could be caught up by increasing processing speed of the machine. Their work is one of the first including robustness in the optimization of production systems under energy constraints. The model proposed by [9] consists in minimizing the carbon footprint, the makespan and the consumption peaks in a Flow-shop context with variable speeds allowed on machines. The market tool they used did not permit to obtain a single point on the Pareto frontier in a calculation day even in the case of a two stage Flow-shop with two possible speeds per machine and 36 jobs.

[7] proposed a state of the art of different practices concerning the manufacturing systems with energy constraints. They also stated that is very difficult to obtain data concerning energy consumption. As stressed in their study it is not easy to optimize recently constructed manufacture industries because they are energetically well designed; however, even a smooth optimization in less recent manufactures could lead to a strong improvement. In their work [4] proposed a solution which consists in avoiding consumption peaks on a Flexible Flow-shop. They used an Energy Aware Scheduling (EAS) module on the existing schedule obtained with an APS (Advanced Planning and Scheduling) system. The EAS does not modify the given schedule, but optimizes it from the viewpoint of energy consumption by defining a new timetable for the operations. Two approaches are used: a mix-integer linear program (MILP) and a Randomized Neighborhood Search (RANS). They noticed that the MILP can be used when a large consumption peak is allowed and quickly overtaken when the energy threshold is lowered. The possibility to link the given EAS to an existing APS without changing the system in place for a solution handling both applications at the same time is a strong advantage of their approach. They finally noted the fact that their model could be strengthened, by integration of other objectives such as variability of costs and energy need of machines which is constant in their study (unitary).

[18] took into account the variability of electricity pricing during a day, by inclusion of Time-of-Use (TOU) rates in a Flexible Flow-shop. They noticed that it is better in a flexible production system to have parallelized a fast machine with high energy consumption with a slow but economic one, and high energy machine with a slow and economic one, rather than two medium machines from the viewpoint of speediness and energy consumption. [26] used a similar approach by taking into account TOU and transitions between machine states in a single machine process. The genetic algorithm they implemented could be used in extension of an MRPII (Manufacturing Resource Planning System). However their model does not modify the sequence given in input in order to find a better makespan, and consideration of machines with variable speeds is mentioned as a future study. [17] Studied a Job-shop where both the total tardiness and the total energy consumption are minimized by reducing the idle times of machines. [32] Proposed a time indexed linear program which objective is to minimize the energy spending and the carbon footprint under a TOU pricing in a height level Flow-shop.

Finally, [21] pointed out that the industrial mind-set is still focused on the fact that optimizing energy consumption is time-consuming and too expensive which is a barrier to improvements in energy efficiency that could be made in enterprises. In addition, the recent state of the art proposed by [27] underlines the lack of decisional tools relative to energy efficiency of production systems. Their review shows that most of the studies are input-oriented and quite recent and they stress the necessity to develop more output-oriented or mixed methods.

In order to clarify the approaches proposed in the literature review, the different systems under study, the way energy is optimized and the objective functions are centralized in Table 1. While reviewing the literature a lack of study concerning the Job-shop problem with consumption peaks consideration appeared, since all studies conducted so far mainly concern the total energy minimization as seen in [17], [12] or [25]. In the next section are presented the assumptions used in this paper concerning the Job-shop with an Energy Consumption Threshold (JSECT) where the operations to be scheduled present an electricity consumption peak at their start.

Table 1: Different studies concerning energy in production systems

References	Single Machine	Flow-shop	Flexible Flow-shop	Job-shop	Flexible Job-shop	Carbon footprint	Consummation Pics	Total Energy	Energy Price (TOU)	Makespan	Total Tardiness	Robustness
Balogun and Mativenga, 2013 [2]								x				
Bruzzone et al., 2012 [4]			x				x			x	x	
Dai et al., 2013 [6]			x					x		x		
Fang et al., 2011 [9]		x				x	x			x		
He et al., 2005 [12]				x				x		x		
He et al., 2015 [13]					x			x		x		
Liu et al., 2014 [17]				x				x			x	
Luo et al., 2013 [18]			x						x	x		
Mouzon et al., 2007 [19]	x							x		x		
Salido et al., 2013 [25]				x				x		x		x
Shrouf et al., 2014 [26]	x								x			
Xu et al., 2014 [31]			x				x			x	x	
Zhang et al., 2014 [32]		x				x			x			

3 Assumptions

3.1 Job-shop assumptions

This study is based on the well-known Job-shop theoretical model (A review of the Job-shop problem is given in [14]) representing workshops with multiple paths. The Job-shop problem consists in scheduling a set of n jobs that have to be sequenced on m machines. Each job J_i (i.e.: $i \in [1,n]$) is composed of m operations O_{ij} (i.e.: $j \in [1,m]$) which are sequenced in a predefined order noted $G_i=\{O_{i1}, O_{i2}, \ldots O_{im}\}$. Each operation has to be processed on a given machine μ_{ij} during a processing time p_i and no preemption is allowed (a started operation must not be stopped before its end). The Job-shop problem consists in finding a feasible schedule by managing machine disjunctions as the machines are mutualized between jobs. One of the commonly used objectives is then to find such a schedule with a minimal global duration, also called makespan. The Job-shop problem has received a huge attention over the years, with several extensions such as transportation constraints, time-lags constraints, or financial constraints. Using the disjunctive graph introduced by [24] the operations can be modeled by vertices. Precedence constraints between operations of the same job are represented by an arc. Disjunctive constraints between two operations which require the same machine are modeled by an edge. The cost of an outgoing arc is equal to the duration of the operation. An example of a non-oriented disjunctive graph for a Job-shop scheduling problem where three jobs must be scheduled on three machines is presented on Fig. 1 where dashed edges represent disjunction constraints and each operation is performed by a given machine M_i.

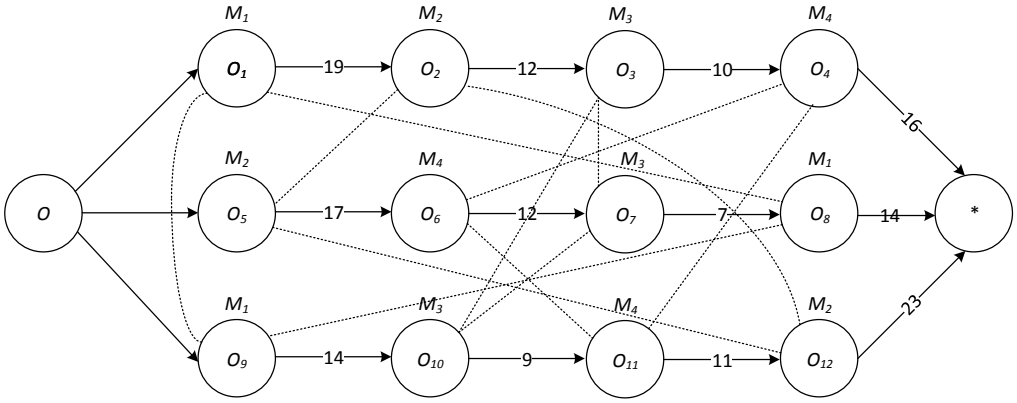


Fig. 1 : A non-oriented disjunctive graph

In the study, the graph representation proposed by [24] is extended with new arcs modelling precedence between operations when a task is delayed because of the energy used by previous and non-finished operations.

3.2 Energy assumptions

In the literature concerning energy optimization on production systems, the operations are generally represented as a simple energy block [29] which representation does not fit the reality. An energy block consists in considering an operation from the energy viewpoint, leading to a rectangle where two contiguous sides respectively represent the duration of the operation and its maximum energy consumption as shown on Fig. 2.

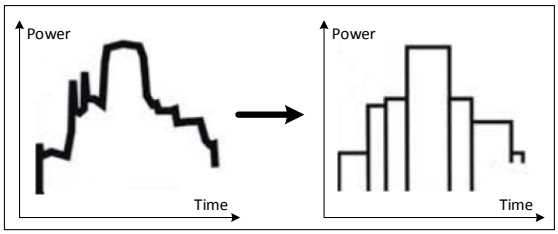


Fig. 2 : Energy Block Representation proposed by Weinert et al., 2011 [29]

Furthermore, in the literature most of the operations are considered from three points of view: basic, cutting, and idle/ready as stressed in [1]. These assumptions are used in the work of [17] when minimizing the total energy consumption. [4] considered that every operation had a unitary consumption because of the lack of industrial data, and thus represented them as a simple energy block, not taking into account the different states of the machine. The time indexed linear model they proposed showed that the lower the energy threshold is the harder it is to find an exact solution for such a problem with a linear solver. Still, their model permitted to lay the foundation of the work presented in this study. However several differences between our approaches must be enlightened. First, the basis of our problem is the Job-shop problem which is more general than a Flow-shop. But the main difference to be noticed concerns the energy assumptions. Indeed, it can be seen in the literature that machine operations have a complex energy behaviour and thus should not be only represented as a single block, especially while considering energy threshold. As stressed in [4] it is quite difficult to obtain real and precise data from the industrial sector. However, some useful information can be found concerning power profiles of machines on works related to green manufacturing, both from the viewpoint of machine or process. Thus, it can be stressed that most of the time an extra energy consumption peak is occurring at the start of the operation as shown in [7; 15] or [16].

Since the objective of this study is to minimize the makespan while considering an energy threshold constraint, it appears that a simple block representation would imply to loose time during the schedule. Indeed, based on the power profile of a 2kW fibre laser proposed in [16], it can be stressed that the laser cutting process consumes almost 37kW whereas it needs less than 20kW after the peak. Thus, if a threshold is fixed at 40kW and another operation that consumes less than 15kW must be scheduled, it means that such an operation should start at the end of the laser cutting process since both operations would consume more than 40kW if treated simultaneously (15+37=52kW > 40kW). If the laser cutting process has a long duration (approximately 400s in the given example), a huge amount of time is lost whereas the operation could have been scheduled just after the 37kW peak. Hence, in this paper are considered operations that are divided into two sub-operations in order to represent the consumption peak at the beginning, and a lower consumption during the remaining processing time. However, since operations cannot be stopped in the Job-shop, two sub operations relative to the same global operation must be linked in order to keep the integrity of the given operation. This constraint is modelled using no-wait arcs between such sub-operations, or maximal time-lags between two consecutive operations which values are equal to the opposite duration of the peak's length (i.e.: if peak duration is equal to *a*, then the time-lag max is equal to *–a* ; A study on time-lags can be found in [5]). A graphical representation of such a problem is presented in the Fig. 3.

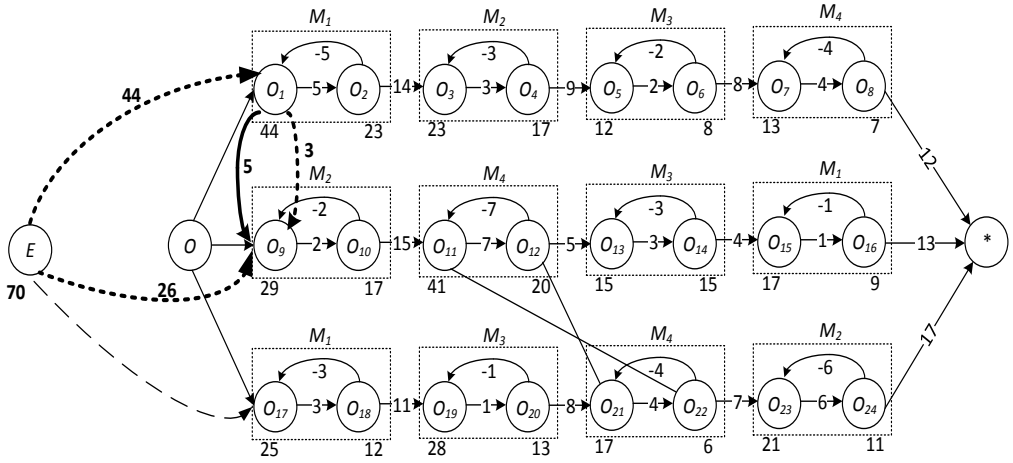


Fig. 3 : Graphical representation of a JSECT

On the Fig. 3 some extra rectangles have been added to delimitate the global operations presented on Fig.1 and to provide the energy needed for each operation. Furthermore, it can be stressed that only two disjunctive edges relative to disjunctive constraints on machines between two operations are presented on the graph, in order to ease its reading (the former operations O_6 and O_{11} in the Fig. 1 which have been transformed into operations O_{11}-O_{12} and O_{21}-O_{22}). Indeed, since operations cannot be stopped once started, these edges represent the fact that the first sub-operation (i.e.: representing the consumption peak) must be scheduled after the end of a previous operation treated by the same machine, and hence must be scheduled after the second sub-operation of such an operation. In this example, it means that the operation O_{11} should be scheduled after the end of the operation O_{22} or the operation O_{21} should come after completion of the operation O_{12}. To complete this graph representation, the energy threshold is modelled as an extra vertex with a value corresponding to the available energy that must not be exceeded. Thus, one of the objective is to efficiently allocate the energy to the machines in order to obtain the smallest possible makespan. When there is not enough energy allocable to an operation, because of running operations extra conjunctive arcs are added to model the new precedence between operations due to the energy threshold. These extra conjunctive arcs represent the duration that an operation must wait before enough energy could flow from a finished operation to the one that must be scheduled (arc in bold on Fig. 3). Thus, a conjunctive arc may be present

between operations that are not directly related to the same machine. In Fig. 3, it can be easily understood that the available energy will not be enough to schedule all possible operations as the threshold is set to *70*. Thus O_1 on machine M_1 and O_9 on machine M_2 cannot be simultaneously scheduled as the sum of their energy is equal to *73*, which is greater than 70. Hence, if a sequence of operations is given, where the first operation (O_1) of job J_1 appears before the first operation (O_9) of job J_2 then it will not be possible to start operation O_9 at 0. Indeed, if O_1 starts at *0*, since the operation O_2 is linked to the operation O_1 with a maximal time lag equal to *-5*, the operation O_2 must start directly after the end of the operation O_1. To start operation O_2, *23* energy units are transferred at the end of the operation O_1 to the vertex modelling operation O_2. Because of the energy threshold, the operation O_9 will start at 5 since operation O_1 (consumption peak of the operation on Machine M_1) has a duration equal to *5*. In this configuration, the operation O_9 would start at *5*. Thus a conjunctive arc (represented in bold) links the operation O_1 and O_9. When the operation O_1 is finished, a flow of *3* energy unit (dashed edge) comes in addition to the *26* units left in the Energy Vertex in order to start the operation O_9 (the values could be different depending on the other operations to be scheduled as mentioned before). By doing this repartition, the energy threshold cannot be exceeded during the schedule and thus the constraint is respected.

In the next section a mathematical formalization of the Job-shop with Energy Consumption Threshold is given. This model includes operations with a consumption peak at their beginning.

4 Linear programming

The model representing the problem has been built to obtain exact solutions not exceeding a given Energy Threshold by repartition of energy resources among the different machines. It relies on a flow added to the incumbent Job-shop problem.

4.1 Parameters

M : set of machines;
V : set of all the sub-operations ($|V|=2.|V|$);
i,j,k,l : indexes representing the different sub-operations to schedule, $i,j,k,l \in [\![1;|V|]\!]$;
O_i : global operation of the sub-operation i;
J_i : job of the operation i;
p_i : duration of sub-operation i;
μ_i : machine required to process sub-operation i, $\mu_i \in M$;
H : a large positive number.
E_{max} : energy threshold that must not be exceeded;
E_i : energy required for processing the sub-operation i;

4.2 Variables

C_{max} : completion date of all operations also referred as the makespan of the schedule;
s_i : starting time of sub-operation i;
$x_{i,j}$: binary variable equal to 1 if sub-operation i is realized before sub-operation j and equal to 0 otherwise;
$y_{i,j}$: binary variable equal to 1 if there is a non-null energy flow from sub-operation i to power the sub-operation j and equal to *0* otherwise;
$\varphi_{i,j}$: denotes the number of energy units directly transferred from sub-operation i to sub-operation j ;

4.3 Linear Formulation

The first line (1) of the linear program refers to the objective of the problem which is the minimization of the completion time of all operations (makespan):

$$Min\, C_{\max} \tag{1}$$

The second set of constraints (2) gives the expression of the makespan, which must be greater or equal to the end date of all the operations:

$$s_i + p_i \le C_{\max}\ ,\ \forall i \in V \tag{2}$$

The third set of constraints (3) represents the disjunctions constraints for operations occurring on the same machines. In these constraints, if two operations *i* and j of different jobs must be treated by the same machine, then *i* is treated before *j*, or *j* is treated before *i*:

$$\begin{aligned} &x_{j,k} + x_{l,i} = 1, \\ &\forall (i,j,k,l) \in V / J_k \neq J_i, k < l, i < j, O_i = O_j, O_k = O_l, \mu_i = \mu_k \end{aligned} \tag{3}$$

The fourth set of constraints (4) defines the starting dates of operations of a job according to its sequence of operations.

$$s_j - s_i \ge p_i\ ,\ \forall (i,j) \in V / j > i, J_i = J_j \tag{4}$$

The fifth set of constraints (5) ensures that, if *i* and *j* are two sub-operations referring to the same global operations, then *j* is processed directly after the end of the sub-operation *i* (i.e. no-wait constraints).

$$s_j - s_i = p_i\ ,\ \forall (i,j) \in V / i \neq j, O_i = O_j \tag{5}$$

The next constraints (6) adjust the starting dates of operations that belong to different Jobs but need the same machine, as they can't be processed simultaneously.

$$\begin{aligned} &s_j - s_i - H x_{i,j} \ge p_i - H, \\ &\forall (i,j) \in V / O_i \neq O_j, \mu_i = \mu_j, \\ &mod(i,2) = 0, mod(j,2) = 1 \end{aligned} \tag{6}$$

The constraint (7) avoids to exceed the Energy Threshold when processing the operations as it can't be allocated more energy to the operations than E_{max}.

$$\sum_{j \in V} \varphi_{0,j} \le E_{\max} \tag{7}$$

Constraints (8) ensure that the sum of energy flows from sub-operations and initial energy threshold is equal to the energy needed for the sub-operation *j*.

$$\sum_{i \in V} \varphi_{i,j} = E_j\ ,\ \forall j \in V / i \neq j \tag{8}$$

Constraints (9) ensure that the sum of energy flows from the considered sub-operation *i* to the other ones never exceeds the energy that was used for its processing.

$$\sum_{j \in V} \varphi_{i,j} \le E_i\ ,\ \forall i \in V / i \neq j \tag{9}$$

Constraints (10) ensure that if there is an energy flow from *i* to *j* $y_{i,j}$=1 (this variable is then used in Constraints (12)). If yi,j=0 then no flow is possible from i to j.

$$\varphi_{i,j} \le H y_{i,j}\ ,\ \forall (i,j) \in V / i \neq j \tag{10}$$

Constraints (11) stipulate that if there is no need of a flow from *i* to *j* ($\varphi_{i,j}$=0), then necessarily $y_{i,j}$ = 0, if $y_{i,j}$ = 1, then $\varphi_{i,j} \ge 1$:

$$y_{i,j} \le \varphi_{i,j}\ ,\ \forall (i,j) \in V / i \neq j \tag{11}$$

Constraints (12) fix the energy flow between sub-operations of a same global operation according to the energy available:

$$\varphi_{i,j} = \min(E_i, E_j), \forall (i,j) \in V / i < j, O_i = O_j \quad (12)$$

Constraints (13) adjust the starting dates of sub-operations which need to wait before the end of previous operations in order to not exceed the energy threshold and receive an energy flow from a previously scheduled operation:

$$s_j - s_i - Hy_{i,j} \geq p_i - H, \forall (i,j) \in V / J_i \neq J_j \quad (13)$$

Constraints (14) stipulate that no flow is possible between two sub-operations *i* and *j*, if *i* and *j* belong to the same job and if *i* is processed before *j*.

$$\varphi_{j,i} = 0, \forall (i,j) \in V / i < j, J_i = J_j \quad (14)$$

Constraints (15) imply that if there is a flow from *i* to *j* then there must not be flows from *j* to the predecessors of *i* in the corresponding product line.

$$y_{j,k} + y_{i,j} \leq 1, \forall (i,j,k) \in V / k \leq i, J_i = J_k, J_i \neq J_j \quad (15)$$

Constraints (16) imply that if operation *i* and *j* need the same machine, but *i* is scheduled before operation *j* then no flows are allowed from *j* to any predecessors of *i* in its product line.

$$x_{i,j} + y_{j,k} \leq 1, \forall (i,j,k) \in V / k \leq i, J_i = J_k, i \neq j, \mu_i = \mu_j \quad (16)$$

Finally, the set of constraints (17) avoids cycles between operations occurring on the same machine.

$$x_{i,j} + x_{j,k} + x_{k,i} \leq 2, \forall (i,j,k) \in V / J_i \neq J_j \neq J_k, \mu_i = \mu_j = \mu_k \quad (17)$$

As stressed by the results presented in Table 3, the resolution of such a problem as the JSECT is difficult for a solver even for small scale instances, thus an approach using a metaheuristic is proposed in the next section.

5 GRASPxELS

5.1 Principles of the GRASPxELS

The GRASPxELS is a multi-start metaheuristic proposed by [22] and is relying on a Greedy Randomized Adaptive Search Procedure (GRASP) proposed by [10] and an Evolutionary Local Search (ELS) proposed by [30]. This metaheuristic helped to quickly bring very good results to several problems. Furthermore, the combination of the GRASP and the ELS, aims to propose a better suited metaheuristic which will explore a wider range of solutions. A template algorithm of the GRASPxELS is proposed below. As stressed in the Algorithm 1, a GRASPxELS is divided into three phases: the construction phase, the local search phase and the ELS phase. During the ELS phase, neighborhood of the previous local optimum solution is explored through mutations and then ameliorated thanks to the local search. The mutation consists in permuting elements in the repetition vector used by [3] if they belong to different jobs. Finally, the different specificities corresponding to the construction and local search phase are exposed in the next sub-section as they are important part of the metaheuristic.

Algorithm 1: GRASPxELS

Procedure name GRASP×ELS
Begin
1. $S^* \leftarrow \emptyset$
2. **for** p := 1 **to** np **do**
3. $S \leftarrow$ Construction_Phase
4. S ← Local_Search_Phase
5. **if** (f(S) < f(S*)) **then**
6. $S^* \leftarrow S$
7. **endif**
8. **for** i := 1 **to** nb_ELS **do**
9. $f(nS^*) \coloneqq$ INFINITY;
10. **for** j := 1 **to** nb_N **do**
11. nS ≔ neighbor of S obtained by permutation in sequence;
12. nS ≔ Local_Seach_Phase;
13. **if** ($f(nS)$ < $f(nS^*)$) **then**
14. $nS^* \coloneqq nS$;
15. **endif**
16. **endfor**
17. $S \coloneqq nS$;
18. **if** (f(S) < f(S*)) **then**
19. $S^* \coloneqq S$
20. **endif**
21. **endfor**
22. **if** (f(S) < f(S*)) **then**
23. $S^* \leftarrow S$
24. **endif**
25. **endfor**
26. **return** S^*
end

5.2 Specificities

Construction phase:

As the main objective is to propose a solution with minimal makespan, a construction rule based on the duration of the activities is chosen. At each construction step, an activity is randomly chosen from a list of activities with small durations. The constructed sequence is then evaluated.

Local search phase:

The construction phase rarely produces local optimum thus it is useful to explore neighborhood of the constructed solution to obtain a better one. The chosen local search is relying on the neighborhood of [28] which is really fast evidenced by the results obtained in term of computation time (Table 2 and 3).

Finally, as one of the most important algorithm of this study is the evaluation of a sequence of operation for the JSECT, this procedure is presented on the next sub-section.

5.3 Evaluation of a sequence of operations

As mentioned before, a sequence of the operations relies on a repetition vector. The evaluation of such a sequence is handling both physical and energy constraints. The starting date of an operation is modified inside the evaluation function in order to respect the energy threshold. A principle algorithm of such an evaluation function is given in Algorithm 3. This algorithm returns the makespan, the starting date and the father of each operation.

Algorithm 3: Evaluation

Procedure name Evaluation

Input/output

λ: sequence to evaluate

Input

n: number of operations

Variables

i: loop index

op_M[]: last operation on machine

t_Job[]: time job has been treated

t_S[]: table for energy change dates

t_E[]: table for energy available according to dates

job: job treated

op: operation to schedule

machine: machine for the operation

father, fatherD: predecessor and disjunctive predecessor

d, dPD: end date of predecessor and disjunctive predecessor

Begin

1. Init *op_M, t_Job, t_S;* Init *t_E* with energy threshold allowed;
2. **FOR** *i* :=0 **to** *n* **DO**
3. *job* **:=** λ*.sequence*[*i*] ;
4. *op* := operation corresponding to *job*'s occurence;
5. *machine* := machine for *op*;
6. *d* := 0; *dPD* := 0;
7. *father* := -1; *fatherD* := -1;
8. **IF** *t_Job*[*job*] <> 0 **THEN**
9. //Conjunctive father
10. *father* := *vertex* – 1 ;
11. *d*:= *End*[*father*];
12. **END IF**
13. **IF** (*op_M*[*machine*] <> -1) **THEN**
14. //Disjunctive father
15. *fatherD* := *op_M*[*machine*] ;
16. *dPD*:= *End*[*fatherD*] ;
17. **END IF**
18. **IF** (*dPD* > *d*) **THEN** update father and *d*; **END IF**
19. Call Adjust_Energy_Date(*d*, *op*, *father*, *t_S*, *t_E*);
20. Save *d* and *father* into λ;
21. Increment *t_Job*[*job*];
22. *op_M[machine*] ≔ *op*;
23. **END**
24. Compute final makespan and store it into λ*;*

End

Algorithm 3 is focused on evaluating a sequence of operations according to machine disjunctions. The energy aspect of the schedule is handled during the Adjust_Energy_Date algorithm, where the starting date of an operation is updated in order to respect the given energy threshold. The principles of adjusting dates according to available energy is given in algorithm 4.

Algorithm 4: Adjust_Energy_Date

Procedure name **Adjust_Energy_Date**
Input/output
d: theoretical starting date of the operation
father : theoretical father of the operation
t_S[]: table for energy change dates
t_E[]: table for energy available according to dates
Input
op: operation to schedule
Variables
s_1, s_2, s_3: indexes for placing operation according to energy
op_placed: boolean
Begin
1. s_1:=0, s_2:=0, s_3:= 0;
2. *op_placed* := false;
3. **WHILE** not *op_placed* **DO**
4. s_1:= index in *t_E* with enough energy available for first part of op starting from s_1;
5. Check if there is enough energy during first part of *op*;
6. s_2:= index for end of first part of *op* if possible;
7. **IF** the first part of *op* is schedulable **THEN**
8. Check if there is enough energy during second part of *op*;
9. s_3:= index for end of second part of *op* if possible;
10. **IF** the second part is schedulable **THEN**
11. *op_placed* := true;
12. **ELSE**
13. s_1:= s_3
14. **END**
15. **ELSE**
16. s_1:= s_2;
17. **END**
18. **END**
19. **IF** *t_S*[*s1*] <> *d* **THEN** update *father* according to energy disjunction; **END**
20. *d*:= *t_S*[s_1];
21. insert *d*, end date of first and second part of *op* in *t_S*;
22. deduce energy used between $s_1$and s_3 in *t_E*;
End

6 Computational evaluation

At first, the linear program has been tested using the CPLEX 12.4 solver on a machine embedding a Xeon E7-8870 processor. The previous algorithms have been implemented in C++ and have been executed on a computer with an i7-4800MQ processor, running Windows 7 (Linpack Benchmark: 2277.01 MFLOPS). Parameters used in the GRASPxELS for the number of restart, the number of ELS and the number of neighbors are respectively 60, 30, 15. The instances used for the JSECT are composed by four to six jobs and four machines. These instances have been randomly generated by considering that the basic energy consumption and the consumption peak of an operation is between 1 and half of the duration of the operation, and

the duration of the consumption peak is taken between 0 and a third of the duration of the operation. An example of an instance is given on Fig. 4. All instances tested in Tables 2 and 3 are available online (see : http://damienlamy.com).

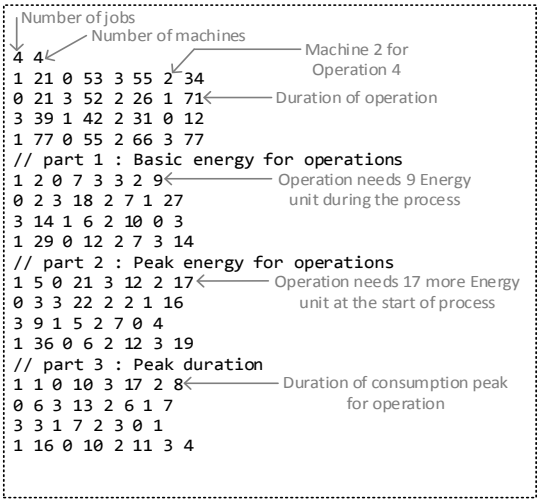


Fig. 4: An example of an instance for JSECT

For each instance, five replications have been made. The results are presented on Table 2 and Table 3. In these tables the column *Jobs x Machines* represents the number of jobs and machines of the instance. Concerning the CPLEX part of the tables, the *Energy Threshold* column represents the energy allowed for the schedule. The *BKS* column refers to the best solution found by the solver. When these solutions are proven optima, an asterisk has been added to the result. The *UB-LB* columns represent the upper bound and lower bound provided by the solver – a dash in *UB* column means that optimal solution has been found. The *Gap* column represents the percentage distance between *UP* and *LB*. The *TT* column corresponds to the time needed for the solver to found the *BKS* – When the computation time reaches 10800s the execution is stopped. In the GRASPxELS part of the tables, the *AVG_S* column represents the average solution over 5 runs. *TT_S* refers to the total execution of the metaheuristic while *TTB_S* refers to the average computation time requested to obtain the best solution found, both in seconds. Finally, the *DEV_LB* column corresponds to the deviation to the *LB* and the *DEV_UB*, in the Table 3, corresponds to the deviation to the best solution found by the solver.

Table 2: Results obtained with CPLEX and GRASPxELS on 45 instances – Part I

		CPLEX					GRASPxELS			
Jobs x Machines	**Instances**	**Energy Threshold**	**BKS**	**UB - LB**	**Gap**	**TT**	**AVG_S**	**TT_S**	**TTB_S**	**DEV_LB**
4 x 4	Inst_1	85	296*	- 296	0	3,26	296	0,15	0,000	0
		75	301*	- 301	0	3,58	301	0,17	0,004	0
		65	317*	- 317	0	13,48	317	0,21	0,000	0
	Inst_2	83	404*	- 404	0	4,95	404	0,18	0,000	0
		73	448*	- 448	0	5,45	448	0,23	0,002	0
		63	492*	- 492	0	62,13	492	0,28	0,004	0
	Inst_3	62	300*	- 300	0	5,41	300	0,23	0,002	0
		52	307*	- 307	0	11,17	307	0,38	0,004	0
		42	345*	- 345	0	90,16	345	0,64	0,012	0
	Inst_4	85	309*	- 309	0	4,49	309	0,13	0,000	0
		75	336*	- 336	0	27,22	336	0,15	0,006	0
		65	343*	- 343	0	15,72	343	0,19	0,026	0
	Inst_5	77	318*	- 318	0	2,56	318	0,14	0,000	0
		67	329*	- 329	0	7,47	329	0,19	0,000	0
		57	350*	- 350	0	26,02	350	0,32	0,002	0
Average:						18,87		0,24	0,004	

Table 3 : Results obtained with CPLEX and GRASPxELS on 45 instances – Part II

			CPLEX					GRASPxELS				
Jobs x Machines	Instances	Energy Threshold	BKS	UB - LB		Gap	TT	AVG_S	TT_S	TTB_S	DEV_LB	DEV_UB
5 x 4	Inst_1	85	344*	-	344	0	7,42	344	0,44	0,010	0	0
		75	344*	-	344	0	13,02	344	0,53	0,004	0	0
		65	370*	-	370	0	40,4	370	0,73	0,004	0	0
	Inst_2	83	425*	-	425	0	13,75	425	0,55	0,002	0	0
		73	448*	-	448	0	1755,19	448	0,69	0,002	0	0
		63	510*	-	510	0	8606,19	510	0,89	0,014	0	0
	Inst_3	66	366*	-	366	0	70,74	366	0,64	0,028	0	0
		56	375*	-	375	0	274	375	0,99	0,004	0	0
		46	468	468	395	15,6	10800	442	1,70	0,052	11,9	-5,56
	Inst_4	85	309*	-	309	0	32,94	309	0,39	0,008	0	0
		75	336*	-	336	0	389,07	336	0,47	0,016	0	0
		65	343*	-	343	0	227,76	343	0,60	0,230	0	0
	Inst_5	77	320*	-	320	0	6,53	320	0,43	0,022	0	0
		67	331*	-	331	0	83,29	331	0,60	0,008	0	0
		57	367*	-	367	0	591,34	367	0,92	0,042	0	0
Average:							1527,44		0,71	0,03	0,79	-0,37
6 x 4	Inst_1	100	406	406	405	0,25	10800	406	0,85	0,076	0,25	0
		90	406*	-	406	0	112,07	406	1,08	0,088	0	0
		80	426*	-	426	0	6022,18	426	1,48	0,054	0	0
	Inst_2	83	477	477	476	0,25	10800	477	0,95	0,010	0,21	0
		73	519	519	499	3,85	10800	519	1,24	0,168	4,01	0
		63	582	582	474	18,56	10800	572	1,71	0,234	20,68	-1,72
	Inst_3	95	433*	-	433	0	44,2	433	0,93	0,018	0	0
		85	463	463	435	6,05	10800	454	1,21	0,082	4,37	-1,94
		75	465*	-	465	0	855,29	465	1,66	0,204	0	0
	Inst_4	85	362*	-	362	0	2140,83	362	0,79	0,006	0	0
		75	411	411	340	17,27	10800	385	0,90	0,098	13,24	-6,33
		65	429	429	363	15,38	10800	429	1,34	0,016	18,18	0
	Inst_5	82	403	403	388	3,72	10800	395	0,88	0,076	1,80	-1,99
		72	408*	-	408	0	2455,89	408	1,24	0,072	0	0
		62	471	471	403	14,44	10800	441	1,92	0,104	9,43	-6,37
Average:							7255,36		1,21	0,09	4,81	-1,22

The results show that the GRASPxELS provides sound solutions. For the first instances (Table 2) the metaheuristic always found the solution provided by the solver, in less than half of a second, whereas it took more than 18 seconds in average for the solver to prove the convergence. Concerning the instances with 5 jobs and 4 machines, the solver always found the optimal solution, unless for the Inst_3 when considering the lowest energy threshold possible. On this set of instances the metaheuristic always provides a solution which quality is better or equal to the solution proposed by the solver. Concerning computation time, the metaheuristic is quite competitive since it stops in less than a second in average, when the solver needs more than 1500s in average to prove the convergence. When increasing the number of jobs to 6, the solver starts to be overtaken with only 6 proven optima. With these instances, the metaheuristic finds solutions which are always better or equal to the ones provided by the solver (more than 1% of improvement in average) in less than 2 seconds for each replication, which is approximately 6000 times faster than the time needed by the solver in average The results show that the metaheuristic is really helpful when searching for good solutions rapidly. Even if the results are not proven to be optima, their quality are always better or equal to these provided by the CPLEX solver which validates this work.

7 Conclusion

Nowadays, commonly used objective functions in scheduling problems, such as makespan or total tardiness cannot be considered as the only objectives. Environmental issues and economic reasons lead to also take into account other objectives such as minimisation of greenhouse gas emission or electricity consumption. There exists different ways of increasing energy efficiency of a production system by minimizing the total energy needed, or the consumption peaks. This study focuses on the last point, leading to the formulation of the Job-shop with an energy

threshold constraint. In this problem each operation presents two type of energy consumption, thus a model considering sub-operations that correspond to the different energy consumption is proposed. The energy threshold is handled by respecting a maximum energy capacity over the machine-network. The instances generated representing small production systems, with few jobs and machines, are not easily solved exactly. Hence a metaheuristic based on a GRASPxELS has been implemented in order to obtain faster solutions to these small scale instances. The results show that the approach is effective since the GRASPxELS returns really good results which are always equal or better to the ones provided by CPLEX, in a competitive duration. However, exact methods should be further studied, with the use of Lagrangian relaxation for example. In a future work, medium and large scale instances will be addressed in a bi-objective context thus leading to a Pareto graphical representation. It could also be interesting to add other objectives such as decreasing the total energy consumption or include TOU pricing in order to reduce the cost of the production. The possibility to have different energy behaviour for the machines should also be studied as the accuracy of the energy discretization [21]. Finally, combining the different approaches seen in the literature concerning the energy-efficient production systems could lead to an interesting and complete problem.

8 Acknowledgements

This work was financially supported by the French Public Investment Bank (BPI) and granted by the ECOTHER project.

References

1. Aramcharoen A. and Mativenga P.T., Critical factors in energy demand modelling for CNC milling and impact of toolpath strategy, Journal of Cleaner Production, 78, 63-74 (2014).
2. Balogun V.A. and Mativenga P.T., Modelling of direct energy requirements in mechanical machining processes, Journal of Cleaner Production, 41, 179-186 (2013).
3. Bierwirth C., A generalized permutation approach to Job-shop scheduling with genetic algorithms. OR spektrum, 17, 87-92 (1995).
4. Bruzzone A.A.G., Anghinolfi D., Paolucci M. and Tonelli F, Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops, CIRP Annals – Manufacturing Technology, 61, 459-462 (2012).
5. Caumond A., Lacomme P., Tchernev N., A memetic algorithm for the job-shop with time-lags, Computers and Operations Research, 35, 2331-2356 (2008).
6. Dai M., Tang D., Giret A., Salido M.A., Li W.D., Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm, Robotics and Computer-Integrated Manufacturing, 29, 418-429 (2013).
7. Duflou J.R., Sutherland J.W., Dornfeld D., Hermann C., Jeswiet J., Kara S., Hauschild M. and Kellens K., Towards energy and resource efficient manufacturing: A processes and systems approach, CIRP Annals – Manufacturing Technology, 61, 587-609 (2012).
8. Energy Administration Information (EIA), International Energy Outlook, http://www.eia.gov/., (2013).
9. Fang K., Uhan N., Zaho F., Sutherland J.W., A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction, Journal of Manufacturing Systems, 30, 234-240 (2011).
10. Feo T.A., Resende M.G.C. and Smith S.H. A, greedy randomized adaptive search procedure for maximum independent set, Operations Research;42: 860–878 (1994).
11. Gutowski T., Murphy C., Allen D., Bauer D., Bras B., Piwonka T., Sheng P., Sutherland J.W., Thurston D., and Wolff E., Environmentally benign manufacturing: Observations from Japan, Europe and the United States, Journal of Cleaner Production, 13, 1-17 (2005).
12. He Y., Liu F., Cao H. and Li C., A bi-objective model for job-shop scheduling problem to minimize both energy consumption and makespan, Journal CSUT, 12(2), 167-171 (2005).

13. He Y., Li Y., Wu T. and Sutherland J.W., An energy-responsive optimization method for machine tool selection and operation sequence in flexible machining job shops, Journal of Cleaner Production, 87, 245-254 (2015).
14. Jain A.S. and Meeran S., Deterministic Job-shop scheduling: Past, present and future, European Journal of operations research, 113(2), 390-434 (1999).
15. Kara S. and Li W., Unit Process energy consumption models for material removal processes, CIRP Annals – Manufacturing Technology, 60, 37-40 (2011).
16. Kellens K., Costa-Rodrigues G., Dewulf W., Duflou J.R., Energy and Resource Efficiency of Laser Cutting Processes, 8th International Conference on Photonic Technologies, Physics Procedia, 56, 854-864 (2014).
17. Liu Y., Dong H., Lohse N., Petrovic S., Gindy N. An investigation into minimising total energy consumption and total weighted tardiness in job shops. Journal of Cleaner Production, 65, 87-96 (2014).
18. Luo H., Du B., Huang G.Q., Chen H. and Li X., Hybrid flow shop scheduling considering machine electricity consumption cost, International Journal of Production Economics, 146, 423-439 (2013).
19. Mouzon G.C., Yildirim M.B. and Twomey J., Operational methods for minimization of energy consumption of manufacturing equipment, Wichita State University Libraries (2007).
20. Mouzon G.C. and Yildirim M.B., A Framework to Minimize Total Energy Consumption and Total Tardiness on a single Machine, 4th Annual GRASP Symposium, Wichita State University, (2008).
21. O'Rielly K. and Jeswiet J. (2014). Strategies to improve industrial energy efficiency. 21st CIRP Conference on Life Cycle Engineering, Procedia CIRP, 15, 325-330.
22. Prins C., A GRASP x evolutionary local search hybrid for the vehicle routing problem. In: Pereira FB, Tavares J, editors. Bio-inspired algorithms for the vehicle routing problem, Studies in computational intelligence, 161, 35–53 (2009).
23. Rager M., Gahm C. and Denz F., Energy-oriented scheduling based on Evolutionary Algorithms, Computers & Operations Research, 54, 218-231 (2015).
24. Roy B. and Sussman B., Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS No. 9 bis, SEMA Paris (1964).
25. Salido M.A., Escamilla J., Barber F., Giret A., Tang D. and Dai M., Energy-aware Parameters in Job-Shop Scheduling Problems, GREEN-COPLAS 2013; IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future (2013).
26. Shrouf F., Ordieres-Meré J., García-Sánchez A. and Ortega-Mier M., Optimizing the production scheduling of a single machine to minimize total energy consumption costs, Journal of Cleaner Production, 67, 197-207, (2014).
27. Trentesaux D. and Prabhu V., Sustainability in Manufacturing Operations Scheduling: Stakes, Approaches and Trends, International Federation for Information Processing, 106-113 (2014).
28. Van Laarhoven P.J.M., Aarts E.H.L. and Lenstra J.K., Job-shop scheduling by simulated annealing; Operations Research; 40(1), 113-125 (1992).
29. Weinert N., Chiotellis S. and Seliger G., Methodology for planning and operating energy-efficient production systems, CIRP Annals – Manufacturing Technology, 60, 41-44, (2011).
30. Wolf S. and Merz P., Evolutionary local search for the super-peer selection problem and the p-hub median problem, Lecture notes in computer science, Berlin, Springer, 4771, 1–15 (2007).
31. Xu F., Weng W. and Fujimura S., Energy-Efficient Scheduling for Flexible Flow Shops by Using MIP, Proceedings of the 2014 Industrial and Systems Engineering Research Conference.
32. Zhang H., Zhao F., Fang K. and Sutherland J.W., Energy-conscious flow shop scheduling under time-of-use electricity tariffs, CIRP Annals – Manufacturing Technology (2014).

MISTA 2015

Production Scheduling Based on Order Utility Functions

Yuri Mauergauz

Abstract This paper presents various aspects of scheduling based on the average orders utility criterion on the planning horizon. In this method the concept of production intensity as a dynamic production process parameter is used. The example is made for Pareto-optimal flexible job shop scheduling problem, when two criteria were simultaneously used: relative setup expenditure criterion and average orders utility criterion. The nature of average orders utility function variation is considered, and the concept of critical horizon is introduced. The software used allows scheduling for medium quantity of jobs. The result of software application is the set of non-dominant versions proposed to a user for making a final choice.

1 Introduction

Wide spread occurrence of Just-in-Time Production methodology in scheduling requires to apply the criteria, which explicitly consider possible deviations of contractually agreed due dates. When such approach is used, completing a job earlier or later than its due date deteriorates quality of scheduling. In scheduling theory, an optimality criterion is called regular, if completion time diminution of any job leads to criterion improvement. If a criterion may be improved by increasing planned time of certain job completion, such a criterion is called non-regular. Usually this criterion involves the sum of absolute deviations of job completion timing from due dates; however, as indicated by [1], other criteria are possible.

At the same time, various obstacles, which may exist inside or outside an enterprise, impede exact completion on delivery dates. Internal causes include machine breakdowns, operator's absence, design changes, lack of control and so on; the external cause is usually untimely arrival of necessary materials. Besides, changes in customer requirements to composition and quantity of commodities may be possible.

Therefore in practice shop floor scheduling is a dynamic process, and its nature essentially impacts production parameters. There are three types of shop control: completely reactive control (dispatching); predictive-reacting scheduling and robust predictive-reacting scheduling.

When dispatching is applied, production schedules are not made. Released jobs are being assigned to machines as they become available, according to the rules used at the enterprise. In this case both job timeliness and job economy are determined by dispatcher's experience.

A more up-to-date method of shop floor control involves predictive-reacting scheduling of production, which is usually implemented in two stages. At the first stage, the calculation of schedule using certain optimization criteria has to be made. At the second stage, usually already in process of completion, the schedule may be corrected, if important events emerge [14]. In such a case the corrected schedule may differ from the primary one in many respects,

Yuri Mauergauz
Sophus Group, Moscow, Russia
E-mail: prizasu@yandex.ru

and its quality may substantially worsen. In some papers, for example [4], attempts were made to estimate possible production delays and to ensure robust production process in this case.

When predictive-reacting scheduling is used, two issues arise: when to reschedule, and how to react to real-time events. Three versions of rescheduling policy are possible [14]: periodic, event driven and hybrid. In periodic and hybrid methods the concept of rolling time horizon is used [3]. A planning horizon is a time interval, which contains moments of completion of a job set, for which scheduling is made.

Duration of rescheduling period is called a planning cycle. Usually it is essentially shorter than a planning horizon. When a planning cycle decreases, the scheduling robustness increases [12], but it becomes difficult to manoeuvre resources, and more reporting is required. Therefore usually the minimal duration of planning cycle is determined by requirements to job organization in the shop. If an event arises within the cycle, which breaks the planning production process, usually the schedule is not fully revised, but the schedule is corrected as needed. After each cycle the planning horizon is shifted by the value of this planning cycle, but its value will not necessarily remain the same.

Increase of the planning horizon often makes it possible to apply so called group technology, which unites jobs of the same type, and to enlarge size of technological batches. In this case production expenses related to machine setups drop considerably. However, when the jobs of one type are united, the jobs of other types are delayed. This fact is known as “dilemma of operation planning” [13]. Solution of such problems may be only attained in multicriteria tasks. The most promising here is the research aimed at building Pareto-optimal diagrams for problem criteria.

It is evident that from the point of view of the best solution for dilemma of operation planning it is necessary to calculate Pareto-front curves on the criteria that depend of job cost and process efficiency. The criterion of relative setup expenses U and the criterion of average orders utility $\overline{V}$ may be considered for dynamic group scheduling [8]. Average utility for the whole set of orders is calculated as the sum of utility functions for all planning jobs. The average utility is a non-regular criterion of the above meaning. This paper below demonstrates that application of average utility function makes it possible to determine the rational planning horizon for each scheduling task.

The remainder of this paper is organized as follows. In Section 2 the function of current orders utility and the function of direct expenses are determined. Section 3 is dedicated to group flexible job shop scheduling. In Section 4 the choice of rational planning horizon is considered. Section 5 contains some concluding remarks.

2 Utility functions in scheduling

The customer service level may be assessed by the current order utility function V. From the manufacturer’s point of view, the order value increases proportionately to work amount, since staff engagement increases. Besides, the more is the time reserve for completing an order, the more attractive is the order, since there is an opportunity to prepare for order execution. Eventually the order time reserve is decreasing, and the order value is diminishing. Moreover, if due date has expired, the order value becomes negative. The manufacturer’s attitude to the order changes with time, and the appropriate function is named *production intensity* [7]:

$$H_i = \frac{w_i p_i}{G} \frac{1}{(d_i - t)/\alpha G + 1} \quad \text{at} \quad d_i - t \geq 0 \ \text{and}$$

$$H_i = \frac{w_i p_i}{G} [(t - d_i)/\alpha G + 1] \quad \text{at} \quad d_i - t \leq 0\,, \qquad (1)$$

where: p_i = processing time of job i; G = plan bucket duration; w_i = weight coefficient of job i; α = “psychological coefficient”; d_i = due date; t = current time.

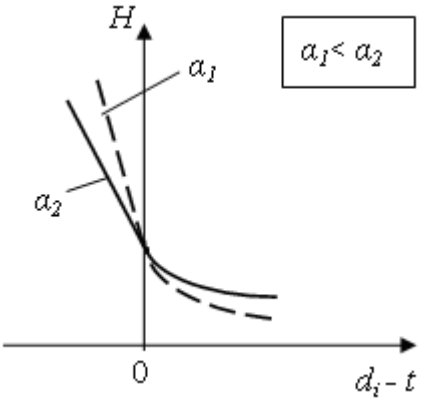


Figure 1 Production intensity diagrams

On abscissa axis in Figure 1 the time reserve is measured. The reserve is equal to subtraction between due date and current time. In the positive part of the diagram ($d_i > t$) the values of intensity decrease in hyperbolic mode with growth of available time reserve. When the time reserve is negative ($d_i < t$) and there is delay of order completion, the production intensity linearly increases. Since production intensity is dimensionless, it has no physical sense, but it has psychological sense. Indeed, when this order parameter is rising, the concern about order execution is increasing. Two curves in Figure 1 differ in the psychological coefficient value. The higher is the α coefficient, the more placid is the attitude to delays, and the lower is the intensity.

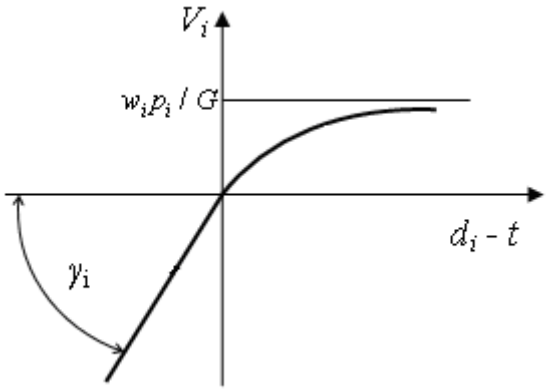


Figure 2 Current order utility function

The production intensity concept may be used for determination of the current order utility function V (Figure 2). Assume that the current utility for an order i is

$$V_i = \frac{w_i p_i}{G} - H_i . \quad (2)$$

The curve in Figure 2 for the positive value $d_i - t \geq 0$ tends to the horizontal asymptote,

$$V_i = w_i p_i / G . \quad (3)$$

In the negative part $d_i - t \leq 0$ the curve turns into the inclined straight line with

$$tg\gamma_i = \frac{w_i p_i}{\alpha G^2} . \quad (4)$$

If the order due date reserve is positive, the manufacturer expects to gain some profit; if reserve is negative and job execution delays the manufacturer, as a rule, it incurs losses. There is a great number of papers dedicated to utility changes as a function of available gain or loss. Results of such researches may be reduced to one of two versions depicted in Figure 3.

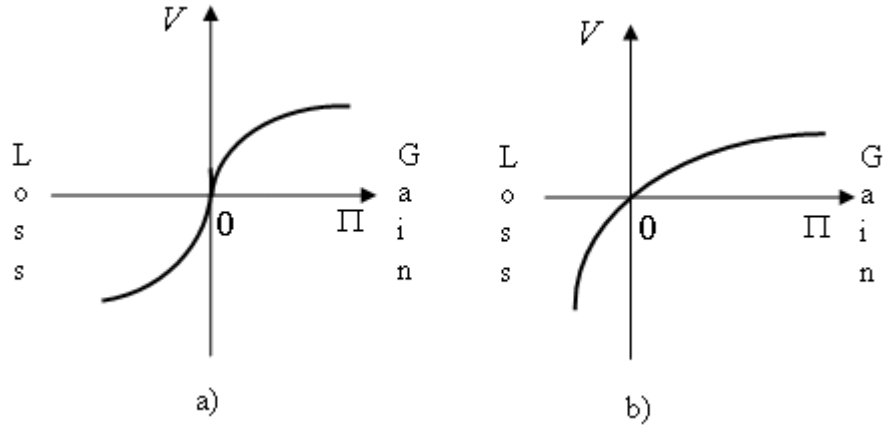


Figure 3 Possible diagrams of gain and loss utility
a) diagrams with risk averse and risk prone areas;
b) diagrams only with risk averse area.

On the abscissa axis in Figure 3 the gain value (anticipated profit Π) is set, on the ordinate axis the gain utility is set in the positive area of the abscissa axis, and the loss utility - in the negative area. The diagram 3a was named an S-mode curve as a result of a well-known research [5] awarded with the Nobel Prize on economics in 2002. Their research proved inclination of ordinary people to risk, when loss is probable (the left part of the diagram). The left part is concave, so a sign of corresponding second derivative is positive, and there is risk proneness.

In contrast to the diagram 3a, the diagram 3b shows risk aversion both for gain or loss perspectives. It is necessary to note that the diagram 3b or Grayson-Bard utility function [6] was obtained in 1957, i.e. much earlier than the diagram 3a. Differences in results of the diagrams 3a and 3b, most probably, were caused by scope people chosen for polling and by direction of money application. In the research by Kahneman and Tversky, modest people were interrogated, money amounts were negligible, and their purpose was consumption. On the contrary, Grayson-Bard function was designed for investments by large companies.

If we compare the curve in Figure 3 and the curves in Figure 2, we can see that the order time reserve is used as gain or loss. It seems to the manufacturer that the long-term order availability represents a considerable gain, but the rate of this gain growth goes down in proportion to the duration. In this positive field the order utility curve behaves entirely like the diagrams in Figure 3. The negative field in Figure 2 is similar to the loss field in Figure 3, but in contrast to the diagrams in Figure 3 there is linear diminution of order utility function in Figure 2. Accordingly, the function second derivative is equal to zero, and risk is neutral.

Due to the additivity property of production intensity and order utility function, it is possible to compute the average utility of the whole order set during a plan bucket. The value of this parameter describes timeliness of order completion and may be used as a criterion of scheduling.

Let us assume that a certain job that corresponds to the node of the scheduling versions tree at the level l is completed at the moment of time C_l. Let us also assume that the job k with processing time p_k starts at the moment t_k, which is more than or equal to C_l. Then

the average utility of the entire set of jobs J from start until completion of the job *k* in the node at the level $l+1$ equals

$$\overline{V}_{l+1,k} = \frac{1}{t_k + p_k} \int_0^{t_k+p_k} V dt = \frac{1}{t_k + p_k} (\overline{V}_l \times C_l + \int_{C_l}^{t_k+p_k} V_k dt) . \quad (5)$$

Possible versions of using the formula (5) for a single machine and rules to compute the integrals it contains are described in [9]. For some parallel machines the recurrent formula (5) may be used without changes, if completion of a job on the previous level l happens before job completion on the subsequent $l+1$ level. Otherwise, instead of the formula (5) the following formula is used [10].

.

$$\overline{V}_{l+1,k} = \overline{V}_l + \frac{1}{C_{lz}} \int_{C_m}^{t_{l+1,k}+p_k} V dt . \quad (6)$$

The bottom limit in the integral (6) is the work completion moment for the last job on the machine m. The function of negative expenses utility (loss function) may be used as the second criterion in the dilemma of operation planning. If the sequence number of planning job is n, then

$$U = \frac{1}{c}[c_s \sum_{l=0}^{n} s_l + c_j \sum_{l=0}^{n} (t_{kl} - C_l)], \quad (7)$$

where: *c* = shift cost; c_s = hour setup cost; c_j = hour idle cost; t_{kl} = moment of job *k* start after job l completion; s_l = setup time for the next job with the sequence number l in the specific schedule version.

3 Group scheduling for job shop manufacturing

As an example of scheduling based on the proposed criteria, let us consider the task for flexible job shop manufacturing. Assume there are certain jobs arriving in any sequence to each available machine M in one of different pools O, for processing of according type. Every job i refers to any of S various types, consists of R_i operations and has to be completed on due date d_i. Setting of due dates is specific for “make-to-order” manufacturing strategy.

In accordance with the well-known three-part scheduling classification, the problem to be considered is:

$$FJ \mid prec, r_i, d_i, s_{fq} \mid U, \overline{V} , \quad (8)$$

where: FJ = designation of flexible job shop manufacturing; d_i = due date of job i; r_i = release moment for job i; s_{fq} = setup duration for job q on machine of pool f; *prec* = requirement of strict sequence of operations for every job.

Let duration of each machine setup from one job to another be independent on sequence of these jobs, which is typical for machine building. There are two target functions in the formula (8), and they may both be improved only within certain limits. The Pareto compromise curve serves as such limit, because in its points the criterion U improvement (diminution) always means the criterion $\overline{V}$ deterioration (diminution).

If job execution is multistage, the process time of job i left before completion consists of process time on N_i of certain j operations

$$p_i = \sum_{j=a_i}^{N_i} p_{ij}\ ,\ N_i = R_i - a_i + 1, \tag{9}$$

where a_i = number of the first unfulfilled operation for job i.

Necessary release date of operation j for job i is determined as

$$g_{ij} = d_i - p_i / E + 1, \tag{10}$$

where E = duration of a working day.

For solving the problem (8) it makes sense to apply the method based on the MO-Greedy approach [2]. In the greedy algorithm at each step a solution with the best corresponding criterion is selected. In a single-criterion approach a version is selected with the best value of the appropriate criterion. For multi-object greedy approach the "beam search" with the problem criteria is used. In the current case, there are two criteria: average orders utility $\overline{V}$ and expenses U. For Pareto front determination by beam search the tree with nodes of intermediate solutions is constructed. At the same time at each step some versions of possible solutions that do not dominate each other are selected. The algorithm below is used.

Step 1 *(Initial computation of utility functions)*

Let us assume that the level number is l=0; the initial expense function value is U_0=0; number of nodes Z_0=1.

External cycle

Step 2 *(Determination of possible operations at next levels)*

For each node z of the constructed tree on the level l all possible operations are determined, and values g_{ijz} are computed by formulas (9,10).

Intermediate cycle

Step 3 *(Determination of necessary machines at next levels)*

For each operation k, which is possible at the moment C_{lz} and is not yet completed, the necessary machine pool is determined.

Internal cycle

Step 4 *(Utility function computation at next levels)*

For each machine m referred to pool f values $U_{l+1,z,k,m}$ and $\overline{V}_{l+1,z,k,m}$ are computed using the formulas (7) and (5, 6). The moments of machine availability have to be taken into account for computation.

End of internal cycle

End of intermediate cycle

Step 5 *(Determination of dominated tree nodes)*

If the level $l+1$ is not last, then for domination on the level $l+1$ of the tree node y with a job i over the tree node x it is sufficient to comply with the following inequations

$$U_{l+1,y} \le U_{l+1,x}\ ,\ \ \overline{V}_{l+1,y} \ge \overline{V}_{l+1,x} \text{ and } g_{l+1,y} < g_{l+1,x}\ , \tag{11}$$

besides, the first or the second inequation is strong.

Otherwise: on the last level $l+1$ domination is possible, if

$$U_{l+1,y} \le U_{l+1,x}\ ,\ \ \overline{V}_{l+1,y} \ge \overline{V}_{l+1,x}\ . \tag{12}$$

Step 6 *(Transition to the next level or stopping)*

If the level is more than the last (all operations are completed), then STOP.

Otherwise: level number increment $l = l+1$ and go to Step 2.

End of external cycle.

As it follows from (11, 12), on each level of tree construction those decisions are thrown aside that are dominated by another decision according to the problem criteria. The last

condition in (11) extends the number of possible branches of the decision tree, since it is necessary for domination that the necessary release moment g_{ij} of non-dominated branch is less than such a moment for dominated one.

Let us assume that according to a set of available orders the Master plan including several monthly plans is generated for the enterprise. Assume that shop floor planning is based on rolling horizon methodology, and the horizon is equal to some weeks. Then it is necessary for shop floor planning to know Master plan information, at least for two months. Assume the planning cycle is equal to a week, and its result is a shop task for the next week. Since the task for the previous week is not always completed, the shop plan for the next week consists of both the new task and uncompleted jobs of the previous week. In such a case the time reserve for such jobs becomes negative.

For example let us assume that 20 jobs of six various types have to be completed on a planning horizon in a shop. Assume that each job includes from three to five different operations, which have to be performed in any given sequence; assume also that in the shop there are 9 machines of five various pools. Table 1 contains a fragment of this task consisting 5 jobs.

Table 1 Task fragment

Job No.	Due date	Release date	Job type	Weight coefficient
1	-1	0	1	2
2	1	0	2	1
3	2	0	4	1
4	2	0	3	1
5	2	0	1	1

As it follows from Table 1, the job 1 had to be completed one work day earlier than the scheduled start, so there is initial tardiness. Other jobs have to be completed in two days after the start. In this case it is assumed that there is enough material for all jobs at the start moment. For every job the weight coefficient may be put in, which increases job importance. For example, weight coefficient of the job 1 is equal to 2, other coefficients are equal to 1, as a rule.

The calculation result for this example gives three non-dominated versions of schedule, one of them is shown in Figure 4 as a record on MS Excel sheet. Numbers in the sequence for the each machine show the job numbers and (through fraction symbol) – the numbers of the operations, which are performed on the machine. Numbers in brackets form groups of lots with jobs of identical type, which do not require any setup, i.e. technological batch.

	A	B	C	D	E	F	G
18	Machine 1: (6/1, 8/1), (10/1, 16/1), (5/2, 14/1), (12/1, 18/1)						
19	Machine 2: 2/1, 3/2, 13/1, (11/2, 15/2), 20/1						
20	Machine 3: 3/1, 5/1, (7/1, 11/1, 15/1, 4/4), 13/3, 17/2, 15/4, 10/3, 20/3, 19/2						
21	Machine 5: 9/1, (6/2, 8/2), 18/3, 7/3, 12/4						
22	Machine 6: 1/4, 4/2, (2/2, 9/2, 13/2), (5/3, 17/1), (3/4, 12/2, 18/2), 7/2,						
23		(16/2, 10/2), 17/3, (8/3, 19/1), 15/5, 10/5, 20/4					
24	Machine 7: 6/3, 20/2, 11/4, 19/3						
25	Machine 8: 4/3, 3/3, 5/4, 15/3, 12/3, 17/4, 8/4						
26	Machine 9: (2/3, 9/3), 14/2, 11/3, 5/5, (3/5, 18/4), 6/4, 12/5, 17/5, (10/4, 16/3), 8/5						

Figure 4: The planning result for one non-dominated versions

In Figure 5 the Gantt diagrams for the machines 1 and 3 are depicted. Rectangles in the diagrams correspond to working operations, gaps stand for idle time. Thick lines correspond to operations without setups as their job type is the same as the previous one.

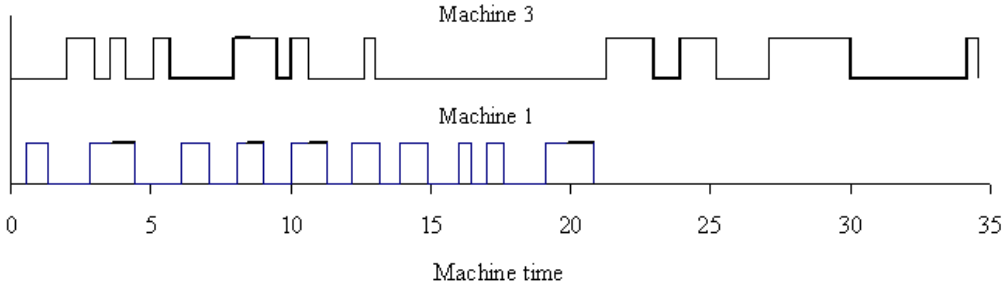


Figure 5: Gantt diagrams for two machines

Let us consider some parameters of the schedule computed by the method above. Calculated coefficient of average load for engaged machines of the pool f :

$$K_f = \frac{\sum_i \sum_j p_{ij}(f)}{Ed_{\max} M_f P_o}. \quad (13)$$

The numerator in (13) is equal to total processing time on the machines of pool f . The denominator is equal to product of the calculated time reserve in hours $Ed_{\max}$, number of machines M_f of the pool f and the normative load density P_0 .

According to the scheduling results, the planning load density for every machine based on calculated machine work time is computed:

$$P_m = \frac{\sum_i \sum_j p_{ij}(m)}{C_{\max,m} - C_{\min,m}}. \quad (14)$$

Assume coefficient of job grouping on machine m is equal to ratio of job quantity and number of setups

$$W_m = \frac{n_m}{o_m}. \quad (15)$$

Average values for machines of pool f :

$$\bar{P}_f = \frac{\sum P_m(f)}{M_f} \quad \text{and} \quad \bar{W}_f = \frac{\sum W_m(f)}{M_f}. \quad (16)$$

When in Figure 6 the diagram 1 of initial calculated load and the diagram 2 of the scheduled average load density are compared, one can notice some similarity. In the diagram 3 of group coefficient there are trends similar to trends in the diagram 2, but far more noticeable. The diagram 4 of group density, which is calculated as product of the diagram 2 and the diagram 3, is close to the diagram 1 on most sections. This fact proves that this schedule automatically groups and condenses operations on the machines of the pool with large load much more intensively than on the machines with small load. Therefore, the algorithm above can automatically determine the bottlenecks of manufacturing and ensure their optimal work.

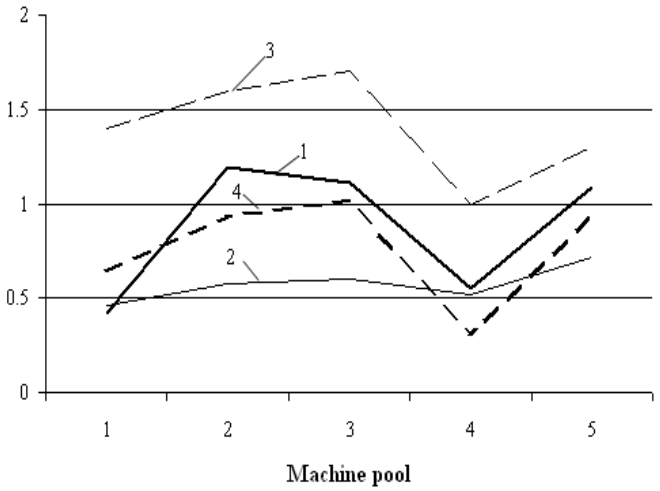


Figure 6: Distribution of schedule parameters among machine pools
1 – initial calculated coefficient of average load
2 – planned load density
3 – planned group coefficient
4 – group load density

4 Choice of the rational planning horizon

Let us consider change of average orders utility function depending on planning horizon value. Assume that the order kit for a single machine consists of 40 jobs, and each job may be referred to one of 12 various types. Job numbers are sorted on due date increasing. The criterion of relative setup expenses U and the criterion of average orders utility $\overline{V}$ may be considered as the set of criteria optimization. Assume also that all jobs may start at any time and have equal priority coefficient, and the job No. 1 is already tardy. Processing time of each job is in the interval of 1 – 3 hours, norms for setup from one job type to another one are within the limits of 0.2-0.6 hour.

	A	B	C	D	E	F	G
14	Horizon 15 jobs						
15	Schedule: 6,9, 1,3,5, 2,11, 4,8,12, 14, 7, 10, 13, 15						
16							
17	Horizon 25 jobs						
18	Schedule: 6,9, 1,3,5,19, 2,11, 4,8,18,20,12, 14,10, 7,15,24, 23, 13,22,16, 25, 21, 17						
19							
20	Horizon 30 jobs						
21	Schedule: 6,9, 1,3,5,19, 2,11, 4,8,18,20,12, 14,10, 7,15,24, 17,						
22	21, 13,16,29,22, 25, 23,30, 26, 28, 27						
23							
24	Horizon 35 jobs						
25	Schedule: 6,9,17, 1,3,5,19, 15,7,24, 4,8,12,18,20, 14,10,35,32, 21,31, 2,11,						
26	16,13,29, 25, 23,30, 22, 28, 26, 27, 33, 34						
27							
28	Horizon 40 jobs						
29	Schedule: 6,9,17, 1,3,5,19, 15,24,7,37, 4,8,12,20,18, 14,10,35,32, 39,28, 2,11,40,						
30	21,31, 16,13,29, 25,36, 23,30, 22, 27, 26, 34, 38, 33						

Figure 7: Economical schedules for various horizons

In Figure 7 the economical schedules (with small setup expenses) for different planning horizons are shown. Here the planning horizon value is determined with maximum job number for scheduling. Assume that at the start moment the machine was adjusted for jobs of type 4, which include the jobs 6, 9, 17, 38. Scheduling calculated with theory above automatically forms job groups of any type.

As it follows from Figure 7, at first, when the horizon is increasing, the economical sequence of jobs remains, new jobs gradually join the existing groups. For instance, the group including the jobs 6 and 9 of the type 4 exists until the horizon is less than 30 jobs. At the same time, the system automatically plans to execute the job 17 of the same type separately and essentially later. When the horizon becomes equal to 35, the economical sequence of jobs is partially changed. The jobs 6, 9 and 17 are planned for execution in the joint group; the jobs 2, 11 are postponed, the jobs 7, 15, 24 are planned earlier.

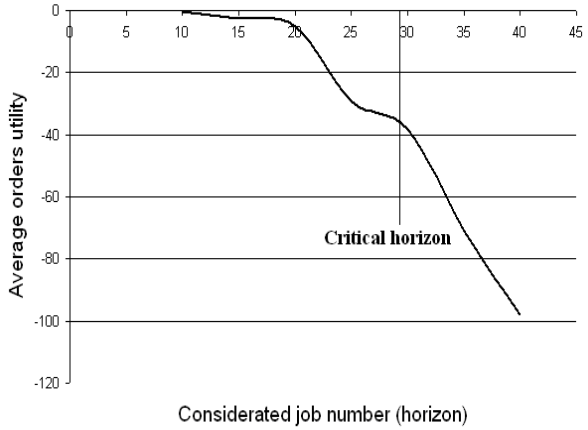
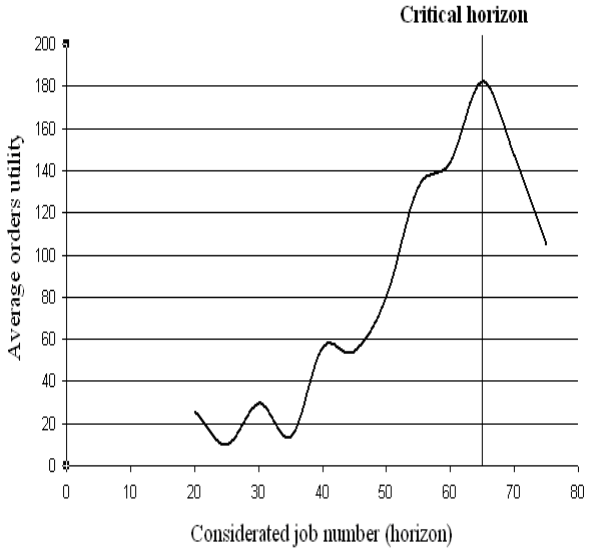


Figure 8: Dependence between average orders utility function and horizon value for single machine (overload)

In Figure 8 the dependence between the average orders utility function and the horizon value is shown. In this case the machine is overloaded, so utility function is negative as completion is often tardy. Until the horizon is equal to 30 jobs, there are utility function oscillations, after 30 jobs utility function diminishes dramatically.



. Figure 9: Dependence between average orders utility function and horizon value for parallel unrelated machines (low load)

In the next example let us consider scheduling for a shop including 6 parallel unrelated machines. Assume that one machine is of high productivity, three machines have middle productivity, and two machines have low productivity. Assume also that the shop may manufacture parts of 6 various types, and there are 75 orders for parts of these types at the moment of scheduling.

Economical schedules for each of parallel machines computed by the method above are similar to the schedule for a single machine. Such schedules include the groups of jobs for various job types. The diagram in Figure 9 is depicted for the case, when parallel machines have low load. In this case average orders utility function is positive, since job completion is not tardy. The utility oscillations are observed until number of jobs is less than 65. If the number is more, the orders utility function diminishes dramatically.

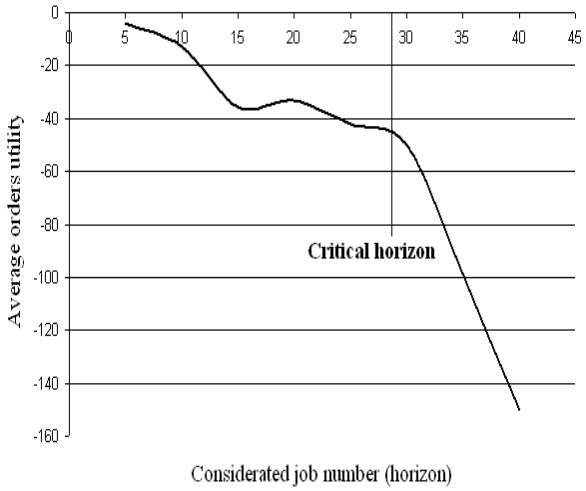


Figure 10: Dependence between average orders utility function and horizon value for flexible job shop (overload)

At last let us consider flexible job shop scheduling for the area of mechanical multistage processing of parts with any sequence of technological operations. Assume that in the shop there are 9 machines of five technological pools. Assume also that the shop gets the task for 40 jobs of six various types, and some of these jobs are in various stages of processing.

Economical schedules for each machine in the shop are similar to the schedules for the cases above. In Figure 10 the dependence between the average orders utility function and the horizon value for the shop is shown. In this Figure utility function is negative, since the shop is overloaded, and completion is often tardy. The utility oscillations are observed until number of jobs is less than 30. If the number is more, the orders utility function diminishes dramatically.

If diagrams in Figures 8-10 are compared, one can find that in any case the orders utility function at some (critical) horizon begins to diminish dramatically. Apparently, scheduling for the horizon more than the critical one, has no sense. To find the critical horizon, it is expedient to use the decision support systems [11].

5 Conclusion

We have studied some aspects of scheduling based on the average orders utility $\bar{V}$ criterion on the planning horizon. This criterion is non-regular as it takes into account both order tardiness and order early completion. In comparison with other known methods, this method provides automatic grouping of unique jobs on all active machines and at the same time takes into account the due date of all jobs. The method reveals the most loaded working centers automatically and provides for grouping of most jobs for these centers particularly.

For scheduling a set of Pareto-optimal solutions on planning horizon is constructed, and a user make the final decision based on this set. If a planning horizon changes, the calculated versions of the schedule change accordingly. When the horizon increases, the system at first automatically proposes schedule versions with increasing grouping of jobs to form technological batches. After the critical horizon value has been attained, the average orders utility function diminishes dramatically, and further increasing of the planning horizon is not expedient.

In dynamic scheduling the critical horizon value may be different in each planning cycle, and it is sensible to simulate production process for critical horizon determination. If deviations of the planned production process appear, they may be corrected in the schedule and have to be taken into account in the next planning cycle. Since the average orders utility function is a criterion of schedule quality for all jobs on the planning horizon, changes of this criterion by separate schedule corrections are usually not large and, accordingly, have small impact at the schedule structure as a whole.

In reality various additional constraints may arise in process of scheduling. For example, often it is needed to take into account the current device wear and tear, limited storage possibilities, general shipping terms, etc. In the nearest future it is planned to elaborate some solutions that correspond to listed problems.

References

[1] Baker K.R, Scudder G.D., Sequencing with earliness and tardiness penalties: A review, *Operations Research*, 38, 22-36 (1990).

[2] Canon, L.-C. and Jeannot, E., MO-Greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines, *IEEE Int. Symposium on Parallel and Distributed Processing Forum*, 57-69 (2011).

[3] Church, L. K. and Uzsoy, R., Analysis of periodic and event-driven rescheduling policies in dynamic shops, *Int. Journal of Computer Integrated Manufacturing*, 5 (3), 153-163 (1992).

[4] Jorge, L.V., Wu, S. D. and Storer, R. H., "Robustness measures and robust scheduling for job shops," *IIE Transactions*, 26, 5, 32-43 (1994).

[5] Kahneman, D. and Tversky, A., 'Choices, values and frames', *American Psychologist*, Vol. 39, 341–350 (1984).

[6] Keeney, R.L. and Raiffa, H., Decisions with Multiple Objectives: Preferences and Value Tradeoffs, pp.559. John Wiley & Sons, NY, (1976).

[7] Mauergauz, Y., Objectives and constraints in advanced planning problems with regard to scale of production output and plan hierarchical level, *Int. Journal of Industrial and Systems Engineering*, 12, 369-393 (2012).

[8] Mauergauz, Y., Cost-efficiency method for production scheduling, *Proceedings of the World Congress on Engineering 2013*, *1*, London, 587-593 (2013).

[9] Mauergauz, Y., Dynamic Pareto-optimal group scheduling for single machine, *Int. Journal of Industrial and Systems Engineering*, *16*, 537-559 (2014a).

[10] Mauergauz, Y., Dynamic Pareto-optimal group scheduling in parallel machine shop, *Int. Journal of Industrial and Systems Engineering*, *18*, 199-221 (2014b).

[11] Mauergauz, Y., Decision support tool for group job-shop scheduling problems, *Proc. of the 4th Int. Conf. on Simulation and Modeling Methodologies, Technologies and Applications*, Vienna, 397-406 (2014c).

[12] Muhlemann, A. P., Lockett, G., and Farn, C. K., Job shop scheduling heuristics and frequency of scheduling, *Int. Journal of Production Research*, 20 , 227-241 (1982).

[13] Nyhuis, P. and Wiendal, H.P., Fundamentals of Production Logistics, pp.312. Springer, Berlin (2009).

[14] Vieira, G. E., Hermann, J. W. and Lin, E., Rescheduling manufacturing systems: a framework of strategies, policies and methods, *Journal of Scheduling*, 6 , 36-92 (2003).

MISTA 2015

Energy Consumption in Single- and Multi-installment Divisible Loads Processing in Systems with Hierarchical Memory

Jedrzej Marszałkowski · Maciej Drozdowski

Abstract Energy and time performance in processing divisible loads on systems with hierarchical memory is considered. Two modes of operation are compared: single-installment and multi-installment processing. Time and energy performance trade-offs are analyzed.

1 Introduction

Energy bill has become one of the key considerations when running a big datacenter [1,3,7]. Hence, energy use optimization, or green computing, is an active research area. One of possible ways of minimizing energy use is designing energy-efficient computer applications. In this work the impact of memory hierarchy and alternative application design styles on time and energy efficiency of distributed computations is considered.

Contemporary computer systems have hierarchical memory structure. At the top-level processor registers offer the shortest access time but their total size is very limited. CPU caches and RAM offer bigger sizes, but at longer access times. The next level of memory hierarchy are SSDs, HDDs (also the networked). At the bottom of the structure is long-term computer storage on tapes, optical media, etc. Access time grows and the size of memory levels increase when progressing from the CPU registers to the external storage systems. Since many applications use huge amounts of memory which do not fit in RAM, a software developer often faces a dilemma of either employing slow memory, such as virtual memory on SSDs, HDDs or changing the algorithms and restructuring the application architecture to fit the processed data in memory and to avoid using external storage as much as possible. In this work we distinguish two approaches to managing application data structures: In the *in-core* mode the processed data structures are held in RAM and higher memory levels. In the *out-of-core* mode also external storage may be intensively used while processing the load. For example, in the out-of-core mode it is possible to process big arrays of numbers and rely on the operating system virtual memory management of the data. In the in-core mode

J. M. Marszałkowski · M. Drozdowski
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland, Tel.: +48-616553031, Fax: +48-618771525,
E-mail: {jedrzej.marszalkowski,maciej.drozdowski}@cs.put.poznan.pl

the designer would rather split the array in smaller pieces and process it on remote computers at the cost of additional communication. In this paper we assume that computation is already distributed. Each machine may receive its share of load once which may lead to substantial use of the out-of-core memory. This way of processing will be called *single-installment* processing. Alternatively, the load may be delivered to a machine in many chunks which is referred to as *multi-installment* processing. It has an advantage of starting computations earlier, and possibly using only in-core computations. A disadvantage is high utilization of communication subsystem and a need for redesigning the application. For the purposes of modeling both types of processing divisible load theory will be applied.

Divisible load theory (DLT) is a scheduling and performance model of parallel and distributed computations on big volumes of data, here referred to as load. The key assumptions of DLT are that load can be divided into parts of arbitrary sizes and these parts can be processed independently in a distributed system. It means that discrete nature of the processed data structures may be ignored due their small sizes compared to the big size of the entire processed load. Moreover, dependencies between the processed data units, if any, can be ignored or eliminated. Surveys on divisible load theory, its extensions and practical applications can be found, e.g., in [2,4,8,9]. In the above books and reviews results of the extensive research on load distribution algorithms minimizing idle times are presented. DLT also proved [4,6] to be quite successful in representing real applications because difference between measured application duration times and DLT expectations were on the level of 1% and better.

Further organization of the paper is the following. In the next section we formulate mathematical models for time and energy performance of single- and multi-installment distribution and load processing. Section 3 outlines results of performance studies. In Section 4 we conclude and discuss possible directions of further research.

2 Mathematical Model

It is assumed that load of size V resides on a file server P_0 which distributes the load to m homogeneous processors $P_1, \ldots, P_m$. Network connections with speed $1/C$ are employed for communication and sending α load units (e.g. bytes) requires time αC. Time of processing load of size α in a system with hierarchical memory is represented by a piecewise-linear function $\tau(\alpha) = \max\{a_1\alpha, a_2\alpha + b_2\}$. Part $a_1\alpha$ corresponds with in-core computations, and part $a_2\alpha + b_2$ with the out-of core computations. Function $\tau(\alpha)$ has two properties: $\tau(0) = 0$ and $\tau(RAM) = a_1 RAM = a_2 RAM + b_2$, where RAM is the maximum size of load which can be held by the application and processed entirely in core memory (it may be smaller than hardware size). Sizes greater than RAM are partially stored in the SSDs, HDDs. A computer can be in one of the states: 1) idle – consuming power P^I, 2) starting – consuming power P^S, 3) networking – consuming power P^N, 4) running (i.e. computing). The initial state of processors $P_1, \ldots, P_m$ is idle, master P_0 starts in networking state. In the starting state a machine is progressing from idle to running or networking. Starting a machine takes time S. A machine is networking when it sends, receives load, but also when it is busy-waiting. After obtaining its share of load a machine immediately progresses to running state and after completing the computations it switches back to the idle state. The energy consumed by a computer when processing α load units in the running state is $e(\alpha) = \max\{k_1\alpha, k_2\alpha + l_2\}$. Part $k_1\alpha$ represents in-core, and $k_2\alpha + l_2$ out-of-core processing.

It is required that $e(0) = 0, e(RAM) = k_1 RAM = k_2 RAM + l_2$. For simplicity of mathematical modeling it is assumed that the time of returning results is negligible. This assumption is not restrictive and can be easily relaxed on the grounds of DLT [2,4]. The challenge is to design load distribution strategy which minimizes total consumed energy E and schedule length T. Two approaches are proposed in the following sections: with single- and with multi-installment load distribution.

2.1 Single-installment Processing

In the single-installment method processors $P_1, \ldots, P_m$ receive load once. A master machine activates processors one by one. Immediately after activating some processor $P_i, i = 1, \ldots, m$, the master sends load α_i to P_i. Only then the master activates P_{i+1}. Let t_i denote the time of computing on P_i. Time t_i must be long enough to process the whole load. Hence, $t_i = \tau(\alpha_i) = \max\{a_1\alpha_i, a_2\alpha_i + b_2\}$. In a schedule of length T processor P_i is starting in time S, networking for time $\alpha_i C$, computing for time t_i and is idle for time $T - S - \alpha_i C - t_i$. Let e_i denote energy consumed on P_i while processing load α_i. We have that $e_i = \max\{k_1\alpha_i, k_2\alpha_i + l_2\}$ Hence, the energy consumed by $P_i, i = 1, \ldots, m$, is

$$E_i = P^S S + P^N C\alpha_i + e_i + P^I(T - S - \alpha_i C - t_i).$$

The master is networking for time $\sum_{i=1}^{m}(S + \alpha_i C)$. Thus, the energy consumed in master machine is

$$E_0 = P^N \sum_{i=1}^{m}(S + \alpha_i C) + P^I(T - \sum_{i=1}^{m}(S + \alpha_i C)) = P^I T + (mS + VC)(P^N - P^I).$$

The problem of energy use minimization subject to schedule length limitation can be formulated as the following linear program:

$$\min \sum_{i=0}^{m} E_i \tag{1}$$

$$iS + \sum_{j=1}^{i} \alpha_i C + t_i \leq T \quad \text{for } i = 1, \ldots, m \tag{2}$$

$$\max\{a_1\alpha_i, a_2\alpha_i + b_2\} = t_i \quad \text{for } i = 1, \ldots, m \tag{3}$$

$$\max\{k_1\alpha_i, k_2\alpha_i + l_2\} = e_i \quad \text{for } i = 1, \ldots, m \tag{4}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{5}$$

In the above linear program the piecewise-linear dependencies of the computation time and energy in equations (3), (4) are given in a simplified form using max function directly. This is acceptable in contemporary LP solvers (e.g. CPLEX), but max can be implemented in any LP solver by using two inequalities and cost of the slack in the objective function. An analogous linear program minimizing schedule length T subject to energy limit E may be formulated. If some solution of the above formulations has $\alpha_i = 0$, then it means that the number of processors is too big for the given problem size V. Such solutions are considered infeasible in the above model.

2.2 Multi-installment Processing

In the multi-installment load distribution it is assumed that load chunks of size RAM are sent to processors $P_1, \ldots, P_m$. If load V is sufficiently big then each processor receives load many times. For simplicity of exposition we assume in the following that V is divisible by RAM. In the multi-installment distribution master starts the processors first which takes time mS in total. When all m machines are up and busy-waiting for the load, master computer starts sending them load in pieces of even size equal $\alpha_1 = \ldots = \alpha_m = RAM$. Processor P_i is idle until time $S(i-1)$, next it starts in time S, and then it is busy-waiting for the first piece of load in time $(m-i)S+C(i-1)RAM$. Thus, the energy consumed in the activation process by $P_1, \ldots, P_m$ is

$$E_A = P^I S(m-1)m/2 + P^S Sm + P^N C \cdot RAM(m-1)m/2.$$

After activation each processor receives $\lfloor V/(m \cdot RAM) \rfloor$ times load of size RAM. Each chunk of size RAM is received and processed in time $RAM(a_1 + C)$ because it fits in core memory. Energy consumed in this central phase of computations by processors $P_1, \ldots, P_m$ is

$$E_C = \lfloor V/(m \cdot RAM) \rfloor m(k_1 + CP^N)RAM$$

In the final stage of computations the load is sent once more to $m_f = (V - \lfloor V/(m \cdot RAM) \rfloor RAMm)/m$ processors. Processors $P_1, \ldots, P_{m_f}$ receive their chunks o load and finish computations in time $(m_f C + a_1)RAM$. The total length of the schedule in the multi-installment processing mode is:

$$T = mS + \lfloor V/(m \cdot RAM) \rfloor (a_1 + C)RAM + (m_f C + a_1)RAM \tag{6}$$

After the computations processor $P_i, i = 1, \ldots, m_f$ is idle for time $C(m_f - i)RAM$. Processors $P_{m_f+1}, \ldots, P_m$ remain idle in the final stage of computation. In particular, $P_i, i = m_f + 1, \ldots, m$ is idle for time $(a_1 - (i - m_f - 1)C)RAM$. The total energy consumed in the final iteration of load distribution is

$$E_f = RAM \Big[m_f(k_1 + P^N C) +$$

$$P^I \left(C(m_f - 1)m_f/2 + a_1(m - m_f) - C(m - m_f - 1)(m - m_f)/2 \right) \Big]$$

The master is communicating or busy-waiting all the time with the exception of time $a_1 RAM$ in the final stage of communication when the last processor already received its load and processes it. Hence, the energy consumed by the master is

$$E_0 = P^N(T - a_1 RAM) + P^I a_1 RAM.$$

The total energy consumption in multi-installment processing is

$$E = E_A + E_C + E_f + E_0. \tag{7}$$

Note that the current method has an analytical solution and needs no special solver. Hence, it has lower computational complexity than in the previous method. The above two modes of processing are compared in the following section.

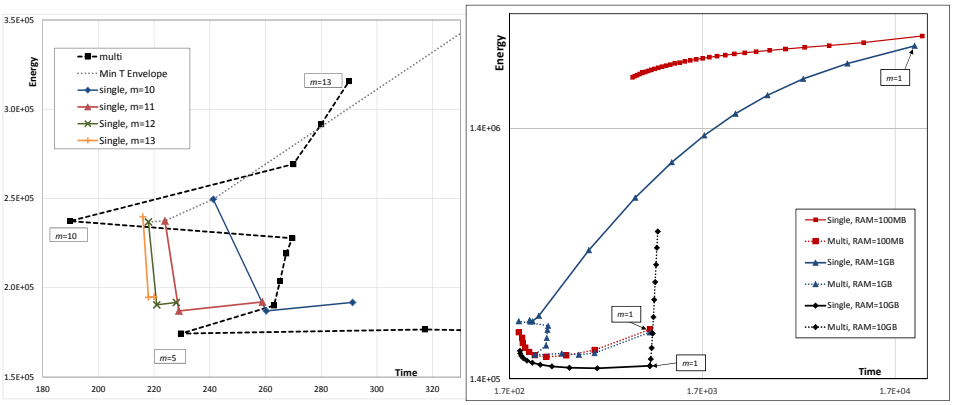


Fig. 1 Energy vs time a) for changing m, b) for changing RAM and m.

3 Performance Comparison

The goal of the simulation conducted in this section is to evaluate time and energy performance of the scheduling methods introduced above. Intuitively, it can be expected that a trade-off between energy and time exists and for shorter schedules more energy is required. Therefore, the results of the simulations will be depicted in energy vs time graphs (cf. Fig.1). The single-installment distribution may have a set of Pareto-optimum solutions in the energy and time space for each instance of the parameters. From the computations two points are derived in the single-installment model (1)-(5): minimum energy schedule and the shortest schedule with their durations and energy consumptions. The linear programs introduced in (1)-(5) were solved using CPLEX 12.6 for minimum energy subject to time limitation. The limits of time T were found by binary search. In the multi-installment model just one solution to (6)-(7) arises as the result of problem size and system parameters. Still, schedule length and energy can be controlled by changing processor number m. Unless stated to be otherwise, the parameters of the system were $C = 0.0078\,\mathrm{s/MB}$, $a_1 = 0.082\,\mathrm{s/MB}$, $a_2 = 2.366\,\mathrm{s/MB}$, $b_2 = -2284\,\mathrm{s}$, $k_1 = 13\,\mathrm{J/MB}$, $k_2 = 294\,\mathrm{J/MB}$, $l_2 = -281\,\mathrm{kJ}$, $m =$, $P^I = 14\,\mathrm{W}$, $P^N = 91\,\mathrm{W}$, $P^S = 101\,\mathrm{W}$, $RAM = 1\,\mathrm{GB}$, $S = 10\,\mathrm{s}$, $V = 10\,\mathrm{GB}$. The time and energy functions parameters were measured on a PC computer executing matrix transposition application. More results on benchmarking energy consumption in divisible applications can be found in [5]. The parameters of the idle and starting states correspond to 'suspend to RAM' state and the following startup process. The results of the simulations are collected in Fig.1, Fig.2. In all figures energy consumption is shown along the vertical axis, and schedule length along the horizontal axis.

In Fig.1a dependence of consumed energy on schedule length is depicted for changing processor numbers m. The dependence of time and energy consumed in multi-installment processing is marked as "multi". It can be observed that for $m = 6, \ldots, 9$ and $m = 11, \ldots, 13$ which do not divide (V/RAM) there are irregularities in schedule length and energy because some machines remain idle in great parts of the schedule. For the single-installment distribution three points may be observed: the point of minimum schedule length, the point of minimum energy schedule, and the rightmost point of the longest schedule for which all processors receive non-zero load. It is possible to extend the dependencies for the single-installment distribution to the longer schedules,

but it results in some machines remaining idle which we consider inconsistent with the model is Section 2.1. Minimum schedule length points are connected in Fig.1a to guide the eye (labeled as "min T envelope"). As it can be seen there is a trade-off between energy and schedule length. The range of changes between the shortest schedule and the most energy-efficient schedule is 1%-8% in schedule lenght and 19%-25% in energy. Thus, the exchange of the makespan for the energy is not equivalent (i.e. not one-to-one proportional). Increasing schedule length beyond the minimum energy length increases the energy cost because on the one hand using out-of-core memory cannot be avoided, on the other hand keeping machines idle for longer time is not costless. The results demonstrate that the multi-installment distribution may be outperformed. This calls for further analysis of both methods performance on a wider range of system parameters. Since similar patterns of the shortest and the best energy schedules in single-installment processing were obtained also in other settings, also to avoid cluttering the figures with excessive details, in the following figures we show only the shortest schedule results.

In Fig.1b energy consumption vs. schedule length is shown for various processor numbers m, and RAM sizes. Each line in Fig.1b represents time-energy trade-off for a fixed RAM size. Each point on the line represents a certain processor number. For each RAM size dependence is shown for single- and for multi-installment distribution. It can be seen in Fig.1b that in multi-installment distribution processing time and energy initially decrease and starting with some number of processors, e.g. $m = 4$ for $RAM = 100$M or $m = 5$ for $RAM = 1$G, energy consumption grows while schedule length decreases. Initially energy and schedule length decrease because by adding processors schedule gets shorter, and the overheads related to schedule length are reduced. Note that in multi-installment distribution processors receive equal load only if $(V/RAM) \bmod m = 0$. Consequently with growing processor numbers inequalities in load distribution increase. The machines which received less load idle wasting energy. Moreover, startup energy SP^S is consumed with each started machine. Thus, after exceeding certain processor number energy costs are growing. In the dependence for $RAM = 1$G an irregularity of the dependence can be seen for $m = 6, \ldots, 9$. Since $(V/RAM) \bmod m \neq 0$ here, machines $P_{m_f}, \ldots, P_m$ are idle in the last iteration constituting big part of the schedule length and the consumed energy. In the case of multi-installment processing with $RAM = 10$G both energy and schedule length grow with each new processor. In this case $V \leq RAM$ and $m = 1$ processor is sufficient to process the whole load. Hence, using each new machine only wastes energy and time. It can be concluded that small sizes of load chunks in multi-installment distribution are more effective (cf. the dependence for $RAM = 100$M vs $RAM = 10$G) because the initial and the final idle waiting periods are shorter. In the single-installment distribution schedule length and energy usage initially decrease because i) more machines have more memory and using out-of-core memory may be avoided, ii) schedule length-related overheads are decreasing. But with the further growth of m costs of startup become dominating and energy use is growing without shortening the schedule. Comparing the multi-installment and the single-installment distribution it can be concluded that multi-installment method is better when RAM is small in relation to V. Contrarily, single-installment distribution is better when RAM is big in relation to V. This is a result of different time and energy cost allocation: Multi-installment distribution loses in time and energy to single-installment in communication and idle time costs. Single-installment method loses in the costs of out-of-core computation. Thus, multi-installment distribution method can be competitive if the costs of idle waiting can be

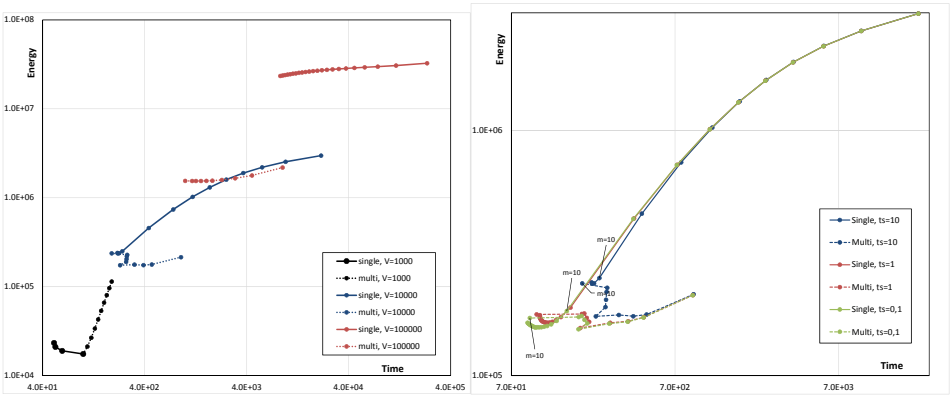


Fig. 2 Energy vs time a) for changing V and m, b) for changing S and m.

amortized in the central part of the schedule when all machines receive their loads many times. Single-installment processing can be competitive if RAM size allows to avoid out-of-core computation.

In Fig.2a the time-energy trade-off is shown for various problem sizes V. As could be expected, schedule length and energy usage grow with problem size. For $V = 100$G, multi-installment processing outperforms single-installment method in time and energy criteria. With the decreasing problem size V the difference in performance of the two method diminishes so that for $V \approx mRAM$ the single-installment method is better balancing costs of communication and idleness than the multi-installment method.

In Fig.2b the time-energy trade-off is shown for various startup times S. It can be observed that for the single-installment distribution startup duration has a two-fold impact. Firstly, the number of processors which can be exploited (i.e. receive non-zero load) increases with decreasing S. This allows for better parallelizing the computations and shorter schedules. Secondly, the schedules on a certain number of machines are less energy-demanding. In the multi-installment distribution the latter effect is similar. However, the irregular parts of the time-energy trade-off are losing more in energy and schedule length with growing startup time S.

4 Conclusions

In this study we compared single- and multi-installment distributed computation with respect to their time and energy performance. The results demonstrate that both methods have their advantages which follow from different ways of trading certain costs of processing the load. The single-installment approach effectively manages idle times but fails when out-of-core computation contributes significant part of the schedule length and energy. The multi-installment communication manages memory usage well but suffers from simplistic and wasteful managing of communications, busy-waiting and idle times.

Further studies may cover other communication strategies, heterogeneous systems and systems managing streams of load, such as server requests.

References

1. Benini, L., de Micheli, G.: System-level power optimization: techniques and tools. ACM Transactions on Design Automation of Electronic Systems **5**(2), 115–192 (2000)
2. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos, CA (1996)
3. Carter, J., Rajamani, K.: Designing energy-efficient servers and data centers. Computer **43**(7), 76–78 (2010)
4. Drozdowski, M.: Scheduling for Parallel Processing. Springer-Verlag New York Inc (2009)
5. Drozdowski, M., Marszałkowski, J.M., Marszałkowski, J.: Energy trade-offs analysis using equal-energy maps. Future Generation Computer Systems **36**, 311–321 (2014)
6. Drozdowski, M., Wolniewicz, P.: Experiments with scheduling divisible tasks in clusters of workstations. In: A. Bode, T. Ludwig, W. Karl, R. Wismuller (eds.) Proceedings of the 6th International Euro-Par Conference on Parallel Processing, LNCS 1900, pp. 311–319. Springer-Verlag (2000)
7. Kogge, P.: The tops in the flops. IEEE Spectrum **48**(2), 48–54 (2011)
8. Robertazzi, T.: Ten reasons to use divisible load theory. Computer **36**(5), 63–68 (2003)
9. Robertazzi, T.: Divisible load scheduling. [on-line] http://www.ece.sunysb.edu/ tom/dlt.html (2011)

MISTA 2015

An Efficient Simulated Annealing for the Integrated Problem of Berth Allocation and Quay Crane Assignment in Seaside Container Terminals

Abtin Nourmohmmadzadeh · Sven Hartmann

Abstract An ever increasing demand for container transshipment have caused seaside terminals to become busy more and more all over the world. Hence, efficient planning and scheduling of operations in a container terminal has gained a great importance. This paper considers two key problems arising in seaside container terminals, namely Berth Allocation and Quay Crane (QC) Assignment simultaneously, and as an integrated mixed integer mathematical formulation with the objective of minimising the weighted sum of waiting time, deviation from desired location and departing delay, for all vessels. A set of test instances of small to large size are generated according to the real condition. To solve the instances, firstly, the GAMS/BARRON software is used, which successfully solves small and medium instances to optimality. Due to the computational complexity of larger instances, a suitable Simulated Annealing (SA) algorithm with a novel solution encoding method is proposed. The comparison of the SA outputs with optimal solutions indicates its good performance in reaching near-optimal solutions in reasonable computational times.

1 Introduction

Container transshipment plays an important role in the worldwide freight transportation since its appearance in the 1950s. This is due to the fact that it provides reliable and standardised means of transportation, which leads to shorter transit times, possibility of using multiple modalities and, finally, it reduces shipping and handling costs [17]. The related statistic shows that the proportion of the global container port throughput in the worlds total dry cargo has increased from 5.1% in 1980 to 25.4% in 2008 (*UNCTAD2009*) [1] and it increased by an estimated 13.3% to 531 million 20-foot equivalent units (TEUs) in 2010 (*UNCTAD2011*) [2]. By this significant increase, container terminals are facing with larger quantities of input containers and being busy and congested more and more. This fact clearly shows that efficient planning and scheduling of operations in a seaside container terminal is very crucial and of great importance.

Department of Informatics, Clausthal University of Technology, Germany
E-mail: an14@tu-clausthal.de | sven.hartmann@tu-clausthal.de

A variety of problems emerge in a seaside container terminal to be optimised. At the beginning, arriving vessels have to be assigned to berths, then the quay cranes (QCs) are scheduled to serve them. Afterwards, the vehicles and transportation equipments are planned to execute the intra-terminal movements of containers, e.g. from the quay to the storage area and vice versa. In the next step, an appropriate storage plan of containers in special locations in the yard area is needed. Finally, transportation modes from outside such as trucks or trains are scheduled at gates of terminal to the hinterland. A comprehensive review of Operations Research problems at container terminals is presented by *Stahlbock and Voss 2008*[13], *Streenken et al 2004* [14] as well as *Rashidi and Tsang 2013* [11].

In this paper, we focus on two main significant problems arising along the wharf, namely Berth Allocation Problem (BAP) and Quay Crane Assignment Problem (QCAP). In BAP there are a set of arriving vessels to be allocated to available berthing locations across the wharf. Each vessel has a desired berthing location at which the cost of total container movements to/from the vessel is minimum. Therefore, deviation from the desired or preferred berthing location increases the related costs. The main constraint for this assignment is that no more than one vessel can berth in the same location at the same time. This makes the satisfaction of all vessels difficult because we seek to minimise their handling time, i.e the time they are at the berth. By solving this assignment problem, other than berthing locations, the exact times for berthing and departure of vessels are determined. As the arrival time of each vessel is its preferred time to moor at the berth, berthing after arrival yields extra cost according to the amount of deviation. On the other hand, leaving the terminal after a specified time must be penalised because of the imposed delay to the vessel. In the QCAP problem we make decision about the number of QCs to serve each vessel in each time period. This problem can significantly affect BAP because the number of QCs that serve a vessel influences its handling time, i.e. with more QCs the embarking and disembarking operations can be done sooner.

Due to the strong interrelation between BAP and QCAP, they are considered and formulated simultaneously as BAQCAP by an integrated mixed-integer mathematical model in our paper. Since such integrated models increase the computational complexity of the problem as its size grows, besides exact solving methods, a heuristic approach is needed to get an acceptable solution in reasonable time. Although these solutions are not necessarily optimal, their small deviation from optimality is not so important in comparison with a much shorter computational time. We propose a Simulated Annealing (SA) algorithm which is well-suited to the characteristics of the problem. For this sake, we devise a novel encoding system for solutions, so that the operators of the SA can be implemented without any trouble.

The rest of this paper is organised as follows: In section 2 a survey of the previous literature is presented. The mathematical formulation and related assumptions are covered in section 3. Section 4 proposes our SA approach. The computational experiments are performed and comparison of the results are made in section 5. Finally, in section 6 we draw a conclusion and recommend directions for future research.

2 Related Work

A variety of researches have been conducted on BAP and QCAP problems so far. They considered different version of the problem and variable assumptions. A few surveys

considered an integrated model (BAQCAP) for solving these two interrelated problems simultaneously, while others worked on them separately and sometimes in more detail. A specific review of the surveys on these two fields are presented by *Bierwirth and Meisel 2010* [3].

The BAP is classified based on some spatial and temporal distinctions by *Bierwirth and Meisel 2010* [3]. These distinctions are: (1) If the berth space is discrete or continuous, (2) If the arrival time of vessels are deterministic or stochastic, and (3) If the handling time of vessels are deterministic or stochastic.

A Particle Swarm Optimisation (PSO) algorithms is presented in *Ting et al.* [16] to solve the discrete and dynamic BAP of realistic size within reasonable time. *Dongsheng Xu et al. 2012* [18] consider the limitation by water depth and tidal condition in BAP. They model the problem as a parallel-machine scheduling with processing set restrictions and divide the time horizon into two periods, in which the processing sets are different. Both the static and dynamic cases of the problem have been taken into account and efficient heuristics are developed for them. *Zhen and Chang 2012* [19] study the development of a robust schedule for berth allocation with a degree of uncertainty (e.g. vessel's arrival time and operation time). They proposes a bi-objective optimisation model that minimises cost and maximises robustness of schedules. A heuristic is developed to solve the large-scale cases. Berth allocation with stochastic vessel handling times is formulated as a bi-objective problem by *karafa et al. 2012* [7] as well. To solve the resulting problem, an evolutionary algorithm-based heuristic and a simulation-based Pareto front pruning algorithm are proposed.

The work of *Hu et al. 2014* [6] considers vessel's fuel consumption and emissions in the integrated problem of BAP and QCAP. They apply a novel non-linear multi-objective mixed-integer programming which is, afterwards, converted to a second-order mixed-integer cone programming model to solve the problem's computational intractability. Additionally, the impact of the number of allocated QCs on port operational cost, vessel's fuel consumption and emission is analysed.

Legato et al. 2014 [10] followed a Simulation-Optimisation approach to make berth allocation decisions. A mathematical programming model at the tactical level and a simulation model at the operational level are considered. Their framework uses a beam search heuristics to obtain a weekly plan at the tactical level, and then to adjust allocation decisions at the operational level, a simulated annealing based search process is proposed. At this level, randomness in discharge/loading operations is taken into account and modelled by an event-based Monte Carlo simulator. *Buhrkal et al. 2011* [4] review and describe three main models for the discrete BAP and enhanced the performance of one of them. They do extensive numerical tests and compare all models from a computational perspective. The results indicate the superiority of the set-partitioning model over all others.

The QCAP is further extended to a version that determines which specific QCs are serving each vessel during each time period. In other words, this version schedules the QCs, and therefore, it is called Quay Crane Scheduling Problem (QCSP). *Kim and Park 2004* [8] discuss the QCSP and present a mixed-integer programming model taking various constraints related to operations of QCs into account. They used a branch and bound (B&B) method to obtain the optimal solution and a greedy randomised adaptive search procedure (GRASP) to overcome the computational difficulty. The satisfactory performance of GRASP is verified by comparison of its results with that of the B&B . *Tavakkoli-Moghaddam et al. 2009* [15] model the quay crane scheduling and assignment

problem as mixed-integer programming and proposed a genetic algorithm to cope with real-size instances.

Elwany et al. 2013 [5] propose an integrated heuristics-based solution methodology that tackles continues BAQCAP. The proposed approach is claimed to produce high quality solutions to such an NP-hard problem in a relatively short time suggesting its suitability for practical use. *Rodriguez-Molins et al. 2013* [12] apply a GRASP-based metaheuristic for BAQCAP aiming at minimisation of the total waiting time elapsed to serve all these vessels. They prove that this metaheuristic reduces the waiting time and increases both the berth utilisation and the throughput of QCs.

B. Türkoğulları et al. [17] focus on the integrated planning of BAP and QCAP. They formulate the problem as a binary integer linear program that is later extended by incorporating the quay crane scheduling problem as well, which is then named BACASP. Although the model for BAQCAP can be efficiently solved even for large instances up to 60 vessels, this is not the case for BACASP. Therefore, a necessary and sufficient condition for generating an optimal solution of BAQCSP from an optimal solution of BAQCAP using a post-processing algorithm is presented. In case this condition is not satisfied, they apply a cutting plane algorithm which solves BAQCAP repeatedly until the aforementioned condition holds.

An overview on the previous literature of BAP and QCAP indicates while numerous studies have been presented that consider one of the problems separately, the surveys on the integrated problem or BAQCAP are not so many. Bearing in mind that the result of each problem strongly affect the other, more integrated models taking more realistic condition into account are needed.

3 Mathematical Formulation

To present our mathematical model in this section, firstly, we explain the main assumptions that the model is based on. Secondly, the notations of the model are introduced, and finally, our mathematical formulation is presented and described.

Our model is based on the following assumptions: 1- The berth is considered to be continuous with a determined length and vessels can berth at any point along it if their length allows. 2- The input parameters of the model are given and deterministic 3- Each vessel has predetermined berthing time and a preferred (desired) berthing location and deviation from them are penalised. 4- Vessels are of three different sizes and the number of cranes which are assigned to each vessel is limited by a minimum and a maximum according to its size. 6- The duration of handling time of each vessel is proportional to the number of QCs assigned to it. 7- Serving vessels at the berth is continuous and without any preemption from berthing up to leaving.

The notations used in our mathematical formulation are as described below:

Sets

V	The set of arriving vessels $i, j \in V$
R_i	The set of allowable numbers of QCs to be assigned to vessel i $q \in R_i = [qmin_i, qmax_i]$, $qmin_i$ and $qmax_i$ are the minimum and maximum allowable number of quay cranes for vessel i
T	The set of time periods, $t \in T = \{1, 2, ..., H\}$, H is the duration of our planning horizon

Parameters

T_i	Target time for vessel i to berth or its arrival time
$B0_i$	The preferred location for vessel i to berth
LF_i	The latest time for vessel i to leave the berth
l_i	The length of vessel i
m_i	The number of QCs that vessel i requires
Q	The number of QCs available at berth
M	A very large integer
$c1_i$	The unit waiting cost of vessel i for berthing (deviation cost from its arrival time)
$c2_i$	The unit deviation cost of vessel i from its preferred location
$c3_i$	The unit delay cost of vessel i (deviation cost from its latest finish time)

Decision Variables

BT_i	Berthing time of vessel i
FT_i	Finish time of vessel i
b_i	Berthing location of vessel i
ΔB_i	The absolute deviation of vessel i from its preferred berthing location
r_{itq}	Binary variables =1 if q QCs are assigned to vessel i in the time period t; otherwise =0
r_{it}	Binary variables =1 if any QC is assigned to vessel i in the time period t; otherwise =0
y_{ij}	Binary variables =1 if vessel j berths physically after vessel i along the wharf; otherwise=0
z_{ij}	Binary variables =1 if vessel j berths chronologically after vessel i ; otherwise=0

Based on the above notations our proposed mathematical model is as follows:

Objective function

$$Min\ Z = \sum_{i \in V} [c1_i(BT_i - T_i) + c2_i \Delta B_i + c3_i(FT_i - LF_i)] \tag{1}$$

Constraints

$$\sum_{t \in T} \sum_{q \in R_i} (q r_{itq}) \geq m_i \qquad \forall i \in V \tag{2}$$

$$\sum_{i \in V} \sum_{q \in R_i} (q r_{itq}) \leq Q \qquad \forall t \in T \tag{3}$$

$$BT_i \geq T_i \qquad \forall i \in V \tag{4}$$

$$\sum_{q \in R_i} r_{itq} = r_{it} \qquad \forall i \in V, \forall t \in T \tag{5}$$

$$\sum_{t \in T} r_{it} = FT_i - BT_i \qquad \forall i \in V \tag{6}$$

$$(t+1) r_{it} \leq FT_i \qquad \forall i \in V, \forall t \in T \tag{7}$$

$$t r_{it} + H(1 - r_{it}) \geq BT_i \qquad \forall i \in V, \forall t \in T \tag{8}$$

$$\Delta B_i \geq b_i - B0_i \qquad \forall i \in V \tag{9}$$

$$\Delta B_i \geq B0_i - b_i \qquad \forall i \in V \tag{10}$$

$$b_j + M(1 - y_{ij}) \geq b_i + l_i \qquad \forall i, j \in V \tag{11}$$

$$BT_j + M(1 - z_{ij}) \geq FT_i \qquad \forall i, j \in V \tag{12}$$

$$y_{ij} + y_{ji} + z_{ij} + z_{ji} \geq 1 \qquad \forall i, j \in V \tag{13}$$

Equation (1) is the objective function of our model to be minimised which is the sum of three terms for all vessels, namely waiting cost for berthing, cost of deviation from the preferred location and delay cost.

Constraint (2) ensures that the sum of QCs assigned to each vessel satisfies the number required by it. Constraint (3) enforces the total number of assigned QCs in each time period not to be more than the number of available QCs. The berthing times must not be before the arrival times that is ensured by constraint (4). Adjustment of r_{it} is done by constraints (5)-(8). Constraints (9) and (10) calculate absolute deviation from the preferred location or ΔB_i. Constraint (11)-(13) ensure that each pair of vessels must not collide each other at the same time, i.e their berthing locations have to be distant respecting their lengths or one have to berth after the other.

4 The Simulated Annealing Algorithm

Considering the computational complexity and NP-hardness experienced while solving problems such as our BAQCAP, other than an exact solution method, a suitable heuristic approach have to be applied to large-scale instances of the problem. This approach should help us to find solutions of good quality in comparatively short computational time.

Simulated Annealing (SA) optimization algorithm which is inspired from annealing in metallurgy was, firstly, presented in *Kirkpatrick et al 1984* [9]. The general framework of this algorithm is as follows: It starts from an initial solution (S_0) as the current solution with an initial temperature (T_0). Then a specific number of neighbourhood searches are done. Each time fitness of the neighbouring solution is compared with the fitness of the current solution. If fitness of the neighbouring solution is better, it is considered as the current solution and otherwise this happen by a probability according to the fitness difference and current temperature which is calculated by $e^{-(Z_{new}-Z)/T}$, where Z_{new} and Z are objectives values of the new and current solution, T is the current temperature. Higher the temperature is, bigger is the probability of the neighbouring solution to be substituted for current solution. When the specific number of neighbours have been investigated, the SA starts a new iteration by reducing the temperature based on a plan. By going forward iteration by iteration the chance of moving to a new solution becomes lower. Finally, after a determined number of iterations or meeting a termination criterion, the algorithm stops.

We adapt the basic SA to be applicable to our BAQCAP. Firstly, to encode a solution we define a matrix. The rows of the matrix represent physical locations along the berth whereas the columns show time periods. If a QC is assigned to a vessel in a time period at a berth's point, the element of the related row and column is equal to the index of the vessel. An example of an encoded solution with three vessels is depicted by Fig. 1. The cells that are occupied by the length of vessels contain the value -1 which means that the assignment of another vessel to them is prohibited. The rest of

the cells where no assignment is done are 0. To penalise the assignment of more QCs than the available number of them at berth in each period, we calculate the number of elements which are not 0 or -1 in each column and the objective function is increased propositional to the violation.

The solution matrix is an encoding approach for a final solution but not the one used by the SA. We shape a vector based on another encoding scheme which determine the order or priority of assignment. Fig. 2 shows this scheme for the priority of 5 vessels to berth. In other words, the vessels are assigned to their preferred berthing time and location in the presented order and if the preferred point is occupied by another vessel which has been assigned before, they will be allocated to the nearest available point which imposes the minimal deviation cost. A neighbouring assignment is defined as a solution in which only two vessels have changed their positions in the order. A neighbouring order is also depicted in Fig. 2 and the exchanged vessels are shown in grey cells.

1	1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
-1	1	1	1	0	0	0	0	0	0
-1	-1	-1	-1	3	3	3	3	3	3
-1	-1	-1	-1	3	3	3	3	3	3
0	0	0	0	-1	3	3	3	-1	-1
0	0	0	0	-1	-1	3	-1	-1	-1
2	2	2	0	-1	-1	-1	-1	-1	-1
2	2	2	0	0	0	0	0	0	0
-1	-1	-1	0	0	0	0	0	0	0
-1	-1	-1	0	0	0	0	0	0	0
-1	-1	-1	0	0	0	0	0	0	0

Fig. 1: An encoded assignment or solution

5	3	1	4	2

5	4	1	3	2

Fig. 2: Encoding of an assignment order and a neighbouring order

All of the SA operations are implemented on these assignment orders. Each time the fitness of a new order is calculated and compared with the solutions that we have had so far. We expect to reach better solutions after passing each iteration and a near-optimal solution at the end of the algorithm. Fig. 3 illustrates the flow chart of our proposed SA algorithm.

5 Computational Results

In this section we need some test instances of different sizes to implement our solution methodologies. For this sake, we randomly generate the input data for 5 problems with 20, 30, 40, 50 and 60 vessels. The rules to generate the input data which simulate a real situation in container terminals to some extent are as follows:
- Vessels are considered to be of three different classes, namely feeder, medium and jumbo. We assume that 60%, 30% and 10% of vessels in each instance belong to the

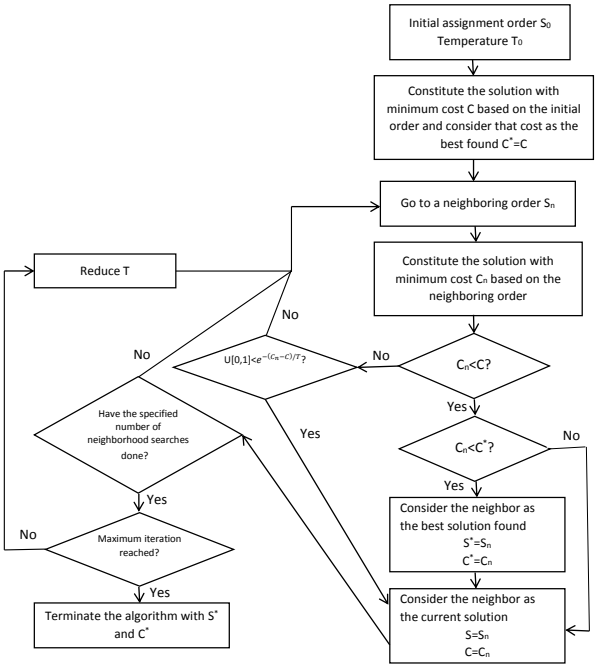


Fig. 3: Flow Chart of the Proposed SA

first, second and third class, respectively. Each of these classes have their own size and characteristics. Therefore, the data related to each vessel is generated based on its class. The technical specifications and cost rates for vessel classes are shown in Table 1

Table 1: Technical specifications and cost rates for different vessel classes

Class	l_i	m_i	$qmin_i$	$qmax_i$	$c1_i$	$c2_i$	$c3_i$
Feeder	$U[8,21]$	$U[5,15]$	1	2	1	2	3
Medium	$U[21,30]$	$U[15,50]$	2	4	2	4	6
Jumbo	$U[30,40]$	$U[50,65]$	4	6	3	6	9

- The planning horizon H is assumed to be 1 week or 168 hours.
- The arrival times of vessels T_i are uniformly distributed in the planning horizon.
- The latest finish time of vessels are calculated according to:
$LF_i = T_i + \lceil \frac{m_i}{(qmin_i + qmax_i)/2} \rceil$
- The Berth's length L is 100 units and the number of available QCs is 10 .
- The preferred berthing location of vessels are uniformly generated between 0 and $L - l_i$, $(U[0, L - l_i])$.

The solution approaches are implemented in a computer with an Intel(R) Core(TM) i7, 3.10GHz CPU and 16GB of RAM. The problems with 20, 30 and 40 vessels are

managed to be solved by the strong GAMS/BARROM program, whereas for 50 and 60 vessels the solver is unable to obtain the optimal solutions within a time limit and stops due to a very long CPU time. Therefore, for the two largest problems we have only the best solutions found until the stoppage time which is set to be 1 hour. Our SA algorithm is implemented in MATLAB environment for all instances and five times for each because the mean of objective function values for each instance can be a good measure for the performance of the SA approach. Based on the initial and final temperature also the cooling plan, 100 iterations are considered for the algorithm with 20 neighbourhood searches in each. The initial order is according to the size of vessels in which larger vessels have higher priority and vessels with the same size are randomly sorted. The computational results of both solution approaches are summarised in Table 2.

Table 2: Computational results of the two solution approaches

Problems	GAMS/BARRON		SA		Gap (%)
$\|V\|$	Optimal	time (s)	Z (mean)	time (s)	$\frac{(Z-optimal)}{optimal} \times 100$
20	504	10.30	535	52.93	6.1
30	882	488.56	926	91.60	4.9
40	1658	2832.81	1712	178.36	3.2
50	3582*	-	3608	281.24	0.07
60	6203*	-	6125	393.05	-1.2

* not optimal but best found after 1 hour

The first column contain the number of vessels or the size of problems. In the second and third columns the optimal values of GAMS/BARRON solver (except for the two large instances) and the CPU times to obtain them are tabulated. The next two similar columns contain the results of our proposed SA algorithm. These are the means of the best solutions and CPU times of 5 runs for each instance. Finally, in the last column the percentage of gap between solutions of the two approaches are shown that can be considered as a comparison criterion.

As the results in Table 2 indicate, our presented SA is able to reach near optimal solutions in all cases. The maximum gap is observed in the first instance which is only 6.1% . As the size of instances increases, this gap becomes smaller such that even in the last instance the SA solution is better than the best one reached by GAMS/BARRON until the stoppage time. It is observed that the CPU time for GAMS/BARRON rises exponentially which indicates that our BAQCAP is a NP-hard problem. The solver even exceeds the long time limit of one hour without reaching the optimal solution and the computational time for the large problems seems to be much longer than this limit. Contrarily, while the computational time of our SA do not rise so rapidly by the increase in the problem size and even for the largest instance it is only a few minutes, its performance to obtain solutions near optimality improves. The great advantage of the SA over the exact solver is a much shorter CPU time. The comparison between the performance of GAMS/BARRON and our proposed SA in terms of objective value and computational time are depicted by Fig. 4 and Fig. 5, respectively.

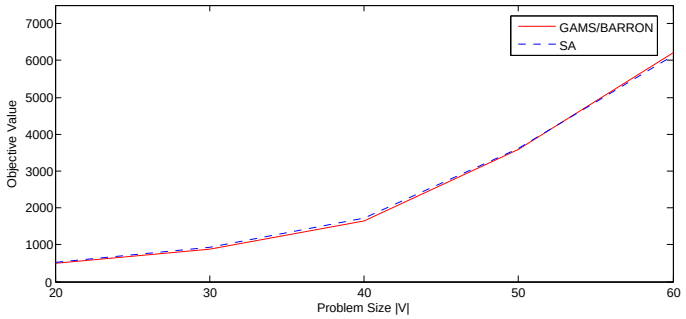


Fig. 4: Objective values of SA vs. GAMS/BARRON

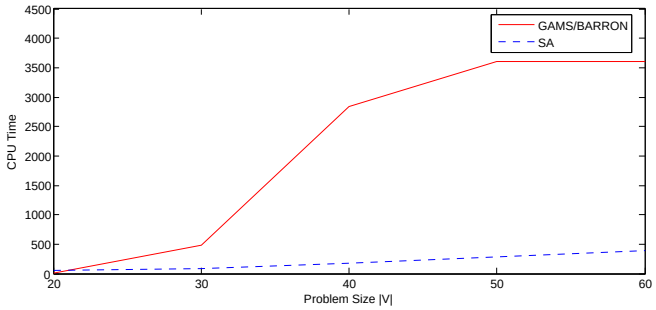


Fig. 5: CPU times of SA vs. GAMS/BARON

6 Conclusion

This paper focused on the integrated BAP and QCAP called BAQCAP by presentation of a mixed-inter zero-one mathematical formulation containing the elements of both problems. Due to the fact that the resulting problem is a NP-hard one and requires a very long computational time and effort by classical and exact methods to be solved, we proposed an efficient SA algorithm to cope with this problem in large scale. The defined encoding scheme and operators of our heuristic approach are based on specifications of the problem. We generated test instances of small to large sizes according to real situation and all the instances are solved by GAMS/BARRON solver and our proposed SA. The comparison of the results of two approaches verifies the fact that our proposed SA algorithm is capable of obtaining solutions of excellent qualities in reasonable time.

Since parameters of BAQCAP are changeable as time passes, more investigation should be done in stochastic version of this problem. Furthermore, considering the interrelation between different scheduling problems in container terminals, integrations of other problems in the model could be followed as a future direction. By the way, the performance of other heuristic optimisation methods can be evaluated.

References

1. UNCTAD, 2009. review of maritime transport, united nations conference on trade and development. http://www.unctad.org.
2. UNCTAD, 2011. review of maritime transport, united nations conference on trade and development. http://www.unctad.org.
3. Christian Bierwirth and Frank Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, May 2010.
4. Katja Buhrkal, Sara Zuglian, Stefan Ropke, Jesper Larsen, and Richard Lusby. Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E: Logistics and Transportation Review*, 47(4):461–473, July 2011.
5. Mohammad Hamdy Elwany, Islam Ali, and Yasmine Abouelseoud. A heuristics-based solution to the continuous berth allocation and crane assignment problem. *Alexandria Engineering Journal*, 52(4):671–677, December 2013.
6. Qing-Mi Hu, Zhi-Hua Hu, and Yuquan Du. Berth and quay-crane allocation problem considering fuel consumption and emissions from vessels. *Computers & Industrial Engineering*, 70:1–10, April 2014.
7. Jeffery Karafa, Mihalis M. Golias, Stephanie Ivey, Georgios K. D. Saharidis, and Nikolaos Leonardos. The berth allocation problem with stochastic vessel handling times. *The International Journal of Advanced Manufacturing Technology*, 65(1-4):473–484, May 2012.
8. Kap Hwan Kim and Young-Man Park. A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156(3):752–768, August 2004.
9. S Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34:975–986, 1984.
10. Pasquale Legato, Rina Mary Mazza, and Daniel Gullì. Integrating tactical and operational berth allocation decisions via SimulationOptimization. *Computers & Industrial Engineering*, 78:84–94, December 2014.
11. Hassan Rashidi and Edward P.K. Tsang. Novel constraints satisfaction models for optimization problems in container terminals. *Applied Mathematical Modelling*, 37(6):3601–3634, March 2013.
12. Mario Rodriguez-Molins, Miguel a. Salido, and Federico Barber. A GRASP-based metaheuristic for the Berth Allocation Problem and the Quay Crane Assignment Problem by managing vessel cargo holds. *Applied Intelligence*, 40(2):273–290, August 2013.
13. Robert Stahlbock and Stefan Voß. Operations research at container terminals : a literature update. *OR Spectrum*, pages 1–52, 2008.
14. Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research a classification and literature review. *OR Spectrum*, pages 3–49, 2004.
15. R. Tavakkoli-Moghaddam, a. Makui, S. Salahi, M. Bazzazi, and F. Taheri. An efficient algorithm for solving a new mathematical model for a quay crane scheduling problem in container ports. *Computers & Industrial Engineering*, 56(1):241–248, February 2009.
16. Ching-Jung Ting, Kun-Chih Wu, and Hao Chou. Particle swarm optimization algorithm for the berth allocation problem. *Expert Systems with Applications*, 41(4):1543–1550, March 2014.
17. Yavuz B. Türkoullar, Z. Caner Takn, Necati Aras, and . Kuban Altnel. Optimal berth allocation and time-invariant quay crane assignment in container terminals. *European Journal of Operational Research*, 235(1):88–101, May 2014.
18. Dongsheng Xu, Chung-Lun Li, and Joseph Y.-T. Leung. Berth allocation with time-dependent physical limitations on vessels. *European Journal of Operational Research*, 216(1):47–56, January 2012.
19. Lu Zhen and Dao-Fang Chang. A bi-objective model for robust berth allocation scheduling. *Computers & Industrial Engineering*, 63(1):262–273, August 2012.

MISTA 2015

A construtive heuristic to reduce costs on an integrated production-distribution environment

Roberto Fernandes Tavares-Neto · Marina Andreotti Ogawa

Abstract The integration between production and distribution functions is a important topic for both academics and practitioners. Relevant practical results have been achieved in a wide-range of industries, such as newspapers, food catering and chemical industries. This paper approaches a production system composed by a single machine that produces setup-dependent jobs that are delivered by a single vehicle with multiple routes. This problem setting combines the most simple element of a classical scheduling problem - a single machine - and the most simple description of a multi-tour vehicle routing problem - with a single capacitated vehicle. The goal is to minimize the sum of total traveling time and the total time that the orders wait to be shipped. This objective is a very relevant one, specially when one deals with perishable goods. A new constructive algorithm, named NEH-TO is proposed, and the results are compared with a non-integrated algorithm. Our experiments shown that the NEH-TO algorithm could achieve better results than the decoupled algorithm.

1 Introduction

The trade-off between setup times, inventory sizes and responsiveness of a production system is a widely accepted concept by both the academy and the industry. E.g., Pyke and Cohen (1990) and Pyke and Cohen (1994) relate the conflict between marketing and production personnel: the first requires a high-variety inventory of finished goods. The second, aims to minimize the production setup costs (to run, as stated by the authors, a "smooth production") - that could be optimality achieve by just producing one type of product. A middle term solution relies on increase the inventory size and diversity and thus solve both problems by increasing the holding costs. Actually, the study presented by Pyke and Cohen (1994) indicates that, in their view, the integration between production and distribution can be an ally to minimize the costs associated to

Roberto Tavares-Neto
Federal University of Sao Carlos
E-mail: tavares@dep.ufscar.br

Marina Andreotti Ogawa
Federal University of Sao Carlos
E-mail: marina.a.ogawa@gmail.com

this trade-off when the problem can be modeled as an extension of the lot-size problem. More extensions of the lot-size problems considering distribution issues can be found in the literatures (e.g., Seyedhosseini and Ghoreyshi (2014) and Bard and Nananukul (2009)).

When dealing with integration issues involving scheduling and distribution, some research can be found in the recent literature. E.g., Chen et al (2009) uses a nonlinear model to maximize the expected profit of the supplier of perishable food products; Chen and Vairaktarakis (2005), also motivated by the food industry, approaches 8 different problems aiming to improve both customer service level and distribution cost. Beyond the food industry, one can find relevance in integrate production and distribution of different products, such as newspaper (e.g. Hurter and Buer (1996)), chemical products (e.g. Chen and Vairaktarakis (2005)) among others. All those industries has something in common: the lifespan of the produced goals is very limited (e.g. the time between the beginning of the newspaper production and the last delivery is measured in few hours). Considering that, one can infer that is interesting to a wide group of organizations that the quality of the delivered products are related to both the delivery time and the time that the production orders waits to be shipped.

However, there's a lack of resources on the literature that deal explicitly with the scheduling issues on integrated planning of production-distribution (Amorim et al, 2013). Some similarities between the present research could be found: e.g. Armstrong et al (2008) deal with a delivery system composed by a single vehicle, but without enabling multiple tours; Amorim et al (2013) consider a set of M parallel lines that produce multiple products to multiple customers, and deliveries it by a limited fleet without considering multiple routes. In our literature review, there is no reference of an approach to an integrated production-distribution problem considering a single machine scheduling problem integrated with a multi-tour vehicle routing problem.

On this paper, a scheduling-distribution integrated problem is approached. The distribution problem is represented by a capacitated single vehicle that deliveries orders to clients. A single order belongs to a single client. Multiple routes are allowed, but only one delivery is possible for each client. The production stage is composed by a single machine, and the setup times are sequence-dependent. All the goals produced are stored into an temporary inventory of finished goods waiting to transport. The goal is to minimize two costs, well-known by the lot-sizing literature: the *holding costs* and the *distribution costs*. Extending the $\alpha/\beta/\gamma$ notation proposed by Graham et al (1979), we indicate this problem as $1 + 1/s_{ij}, \psi/I + D$: $1 + 1$ states for one machine and one vehicle; s_{ij} represents the sequence-dependent setup times; ψ represents the vehicle capacity; $I + D$ represents the inventory and distribution costs. To solve this problem, this paper proposes 7 different variations of 2 heuristics: the first one (named *Decoupled* heuristic) initially solves the distribution section of the problem, and then schedule the production order; the second one (the *NEH-TO* heuristic) deals with both the scheduling and the distribution decisions together.

This paper is presented as follows: section 2 formally defines the problem; section 4 presents the algorithm; results are presented in section 5; final remarks are presented in section 6.

2 Problem Definition

As mentioned before, this paper deals with two correlated decisions: the first one is regarding a single machine that processes a set of N orders. Each order identified by indexes i or j is defined by a process time ρ_i, a setup time κ_{ij} and a size σ_i. The second decision is related to the distribution part of the problem, performed by a single vehicle of capacity ψ. The number of the routes executed by the vehicle is given by K, and each route is referenced by k. Each order is delivered to a single client, represented by a node in a graph where the arc lengths are given by an array δ_{ij}. The completion time of a job i is represented by C_i. Route k departs at a moment given by R_k. The costs are given by $WT_i = max\{0, C_i - R_k\}$ (waiting cost of order i) and by the sum of the lengths of the arcs used by the routes. The solution is given by two sets: a set S_π that represents the production sequence and a set V_π that represents the delivery sequence.

An example of a production-distribution programming is presented in Figure 1. In this adapted version of a Gantt chart, 7 orders are produce in a sequence $J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4 \rightarrow ... \rightarrow J_7$, as presented in the P line. The D line shows three routes: $J_1 \rightarrow J_2 \rightarrow J_3$, $J_4 \rightarrow J_5$ and $J_6 \rightarrow J_7$. In this chart, W_n indicates the waiting time of order n and D_k indicates the duration of route k.

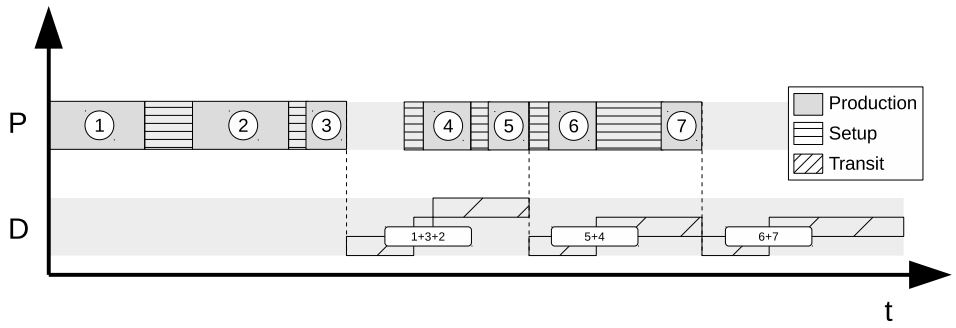


Fig. 1: An adapted Gantt chart presenting both scheduling distribution sequences

A closer examination of the scenario presented in Figure 1 allows on to draw some directives to solve this problem:

1. **Let's say a route k leaves the depot at time t. There should be an order j delivered by k whose production finished at t:** if the order finishes before t, a waiting time related to order j will be included in the fitness function.
2. **If exists, any idle time of the production facility will occur after the vehicle starts the previous route:** since our fitness function requires to minimize the waiting cost, one must backward schedule all the jobs starting by the last job delivered by the next route.
3. **If the setup-times are negligible, the procedure of the production can be obtained solving problems well-known by the literature:** if there is no dependent setup-times, there will be no advantage to produce an order from route $k+1$ before route k leaves (since it will increase the waiting time). Moreover, since part of the goal is to minimize waiting times, the jobs must be ordered by the well-known Longest Processing Time (LPT) dispatch rule. So, to solve this special case of

the problem, one must to: (i) solve the related capacitaded vehicle routing problem; (ii) divide the orders into groups according the route that each one is allocated; (iii) to order the contents of each group according the LPT rule; (iv) backward schedule each component of each group in the production site, granting the last job to complete the processing time on the departure time of the corresponding route.

The following sections present two constructive heuristics to solve this problem: Section 3 shows a 2-phase algorithm that first route and then schedule the orders; Section 4 presents an integrated insertion algorithm.

3 A decoupled approach to the $1 + 1/s_{ij}, \psi/I + D$ problem

One possible approach to a production-distribution problem is to solve it in stages. Thus, the algorithm proposed in this section first generate the routes and then the optimal schedule of the jobs within. The pseudo-code of this algorithm, named *Decoupled*, is presented in Algorithm 1. This algorithm first generates the routes using a classical insertion procedure. On a second stage, each set of orders delivered by each route are sequenced by an insertion algorithm to maximize $\sum C_i$.

Algorithm 1: The pseudo-code for the proposed algorithm

Using a standard insertion algorithm, group the orders into k routes;
foreach *Set of jobs* S_π^k *of each route* k **do**
  Generate a partial scheduling that maximize $\sum_{i \in S_\pi^k} C_i$. Disconsider the first setup cost of the subsequence.;
Schedule backwards each subsequence taking by reference the start time of the respective route;
return *The routes and the scheduling subsequences*

Although this algorithm is very simple, the benefits of the integration between production and distribution are unexplored. A different approach, that investigates the benefits of the integration between those two decisions, is presented in Section 4.

4 A construtive heuristic for the integrated problem $1 + 1/s_{ij}, \psi/I + D$

As expected, preliminary tests of the previous heuristic indicated that this class of problem claims for an integrated approach. Thus, this section presents a constructive heuristic that uses an integrated approach to the problem.

This approach, named *NEH-TO*, is presented in Algorithm 2. As the NEH algorithm, the NEH-TO is composed by an ordering phase and a insert-based construction phase. Extending the flowshop insertion performed by NEH, NEH-TO inserts a job in all possible positions of the schedule sequence and applies a similar insertion algorithm into all possible positions of the set of routes.

The next section presents the analysis of both algorithms.

Algorithm 2: The pseudo-code for the proposed NEH-TO algorithm

```
J ← set of jobs ordered by some criteria;
S_π = S*_π = V_π = V*_π = {0};
nRoutes = 0 foreach j ∈ J do
    for ps = 0...|S_π| do
        S'_π = S_π ;
        Insert j at S'_π in position ps, shifting the jobs on the set if necessary;
        Fitness* = ∞;
        foreach existing route r do
            foreach pv ←all positions of r do
                Insert j at position pv of route r, shifting the jobs on the set if
                necessary;
                Update the values of the best fitness Fitness* and the best sequences
                S*_π, V*_π.
        Create a new route with only the job j and insert it as last route, between each
        existing routes and as first route. If required, update the best values found so
        far. S_π = S*_π V_π = V*_π
return S*_π, V*_π
```

5 Results and analysis

5.1 Generation of instances

Since there is no publicly available benchmarks available to this problem, on this paper a set of random-generated instances were created. The process of generate an instance file is described as follows: the values of ρ_i, σ_i and ψ were generated according uniform distributions given in table 1. The number of jobs is given by n. The orders are then displaced into a $\theta_s \times \theta_s$ space, and the euclidean distances between them are used to find the values of κ_{ij}. A similar procedure using a space $\theta_d \times \theta_d$ is used to calculate δ_{ij}. The distances between the point of origin and the remain nodes are multiplied by θ_g. Each combination of the values of $n = \{5, 10, 20, 40, 80\}$, $\theta_s = \{10, 20, 30\}$, $\theta_d = \{10, 20, 30\}$ and $theta_g = \{1, 10, 20, 30\}$ is used to create an instance. This generates a total of 180 problem categories. Each category contains 50 files, summarizing 9,000 different problem instances.

Table 1: Values used to generate the problem instances

Parameter	Range
ρ_i	$[1; 100]$
σ_i	$[1; 10]$
ψ	$[max(\sigma_i); 5 \cdot max(\sigma_i)]$

5.2 Results

Both algorithms proposed in sections 3 and 4 were implemented in C++, compiled on a GCC 4.8.1 and run on a Linux Mint 16 on a i7 PC with 16GB of RAM. For each one of the problem instances, the 7 ordering procedures were applied on each

one of the 2 algorithms. This leads to 14 different runs. Each instance i received a value of $gap = (fitness(i) - min_j\{fitness(j)\}/min_j\{fitness(j)\})$. Figures 2 and 3 presents the values of gap found according each ordering procedure. When analyzing both graphs, one can easily note that the results of NEH-TO are significantly better than the ones from the Decoupled approach. Tables 2 and 3 presents the average values of the gaps of each algorithm.

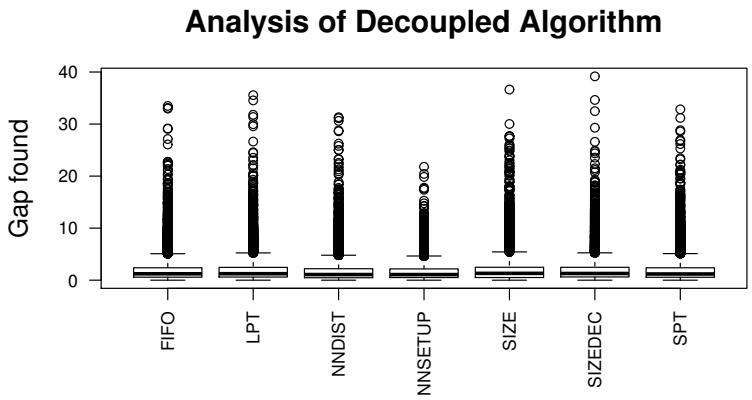


Fig. 2

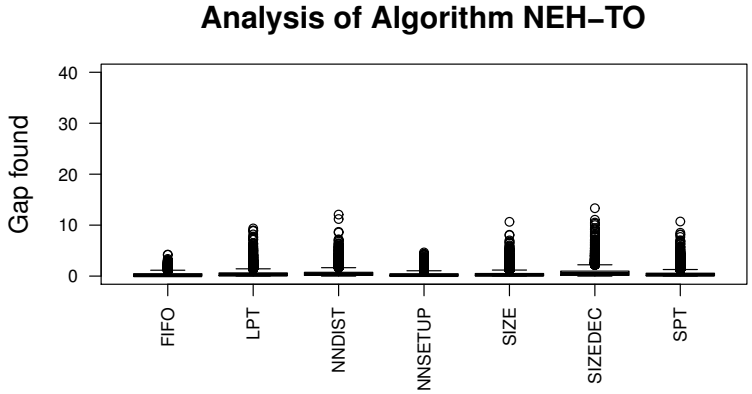


Fig. 3

By focusing the analysis on the NEH-TO algorithm, one can realize that two ordering sequences were be able to obtain better results: FIFO[1] and NNSETUP. Those

[1] since the instances are generated randomly, this is essentially a random rule

Table 2: Average gap found by Decoupled algorithm

Size	FIFO	SPT	LPT	Size	SizeDec	NNSETUP	NNDIST
N=5	0.57	0.55	0.59	0.51	0.65	0.55	0.52
N=10	1.02	1.00	1.10	1.02	1.06	0.93	0.92
N=20	1.63	1.59	1.67	1.74	1.67	1.39	1.46
N=40	2.58	2.55	2.63	2.73	2.69	2.16	2.36
N=80	4.18	4.16	4.21	4.24	4.34	3.47	3.89

Table 3: Average gap found by NEH-TO algorithm

Size	FIFO	SPT	LPT	Size	SizeDec	NNSETUP	NNDIST
N=5	0.09	0.16	0.16	0.14	0.20	0.14	0.19
N=10	0.28	0.30	0.33	0.32	0.36	0.24	0.37
N=20	0.45	0.38	0.42	0.45	0.55	0.28	0.53
N=40	0.46	0.56	0.58	0.47	0.93	0.35	0.72
N=80	0.26	0.86	0.90	0.61	1.57	0.57	1.03

results are shown in more detail on figures 4 and 5. Those figures show that the NEH-TO/FIFO algorithm shows worst results on the 20 and 40 instance sets. On the other hand, NEH-TO/NNSETUP shows a linear tendency on the increase of the value of the gap. This tendency is a expected behavior of a constructive heuristic.

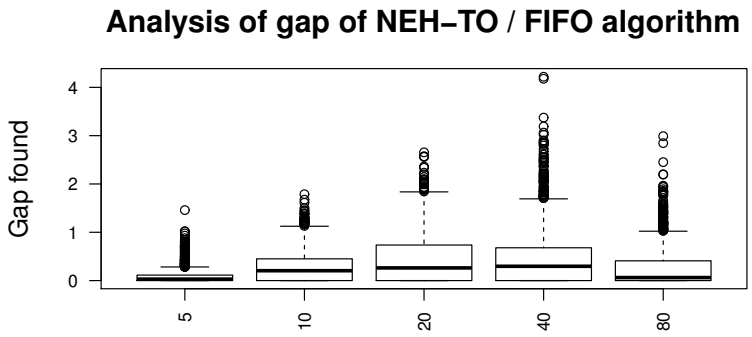


Fig. 4

Finally, the computational times required by each algorithm must be presented to verify the applicability of them. As presented in 4, although the Decoupled algorithm outperform the NEH-TO algorithm, the worst results found of NEH-TO is 2.51 seconds for 80 job instances. On our point of view, it does not disencourage the application of NEH-TO, specially observing the benefits of the integrated approach.

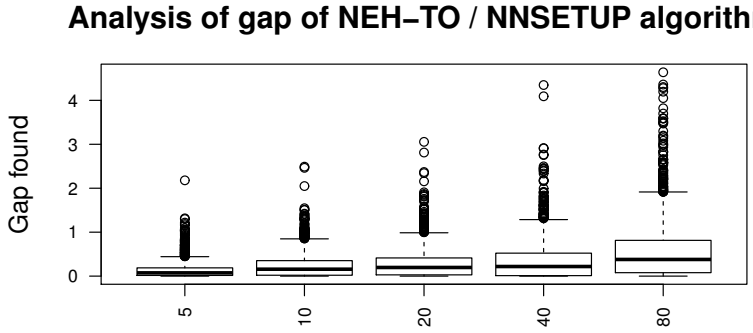


Fig. 5

Table 4: Maximum computational times required to run a single instance of each size (s)

Size	Decoupled Algorithm	NEH-TO
N=5	< 0.01	< 0.01
N=10	< 0.01	< 0.01
N=20	< 0.01	< 0.01
N=40	< 0.01	0.14
N=80	< 0.01	2.51

6 Final remarks

This paper presented an algorithm named NEH-TO that, given a production facility and a distribution system composed by a single machine and a single vehicle with multiple routes, aims to minimize the both inventory and distribution costs. To validate the integrated algorithm, a 2-stage algorithm was developed. The 2-stage algorithm first generate the routes and then the production sequence. The NEH-TO algorithm shows significantly better results, indicating the importance of development of methods that deal with those issues in a integrated manner.

When analyzing the NEH-TO algorithm, it was shown that two ordering methods could achieve better values than the remaining ones: the FIFO - that, due the procedure used to generate the instances, basically states to get the jobs using a random order - and NNSETUP - indicating that the setup times may be an important component of future algorithms that approach this problem.

Since this problem is new on the literature, one can easily enumerate some future topics for research. At first, one can analyze the ordering function, evolving the NNSETUP into a more suitable procedure. Another possibility is to aim research efforts into optimization of the insertion phase, avoiding some moves that could be proven uninteresting. Moreover, since the computational times of the NEH-TO are not prohibitive, we believe that this constructive algorithm may be used as future

basis for the development of improved methods, including local search strategies and meta-heuristics.

Acknowledgements The authors are grateful to FAPESP (process number 2013/21300-2) and CNPq (process number 475214/2013-7) for the financial support to this work.

References

Amorim P, Belo-Filho M, Toledo F, Almeder C, Almada-Lobo B (2013) Lot sizing versus batching in the production and distribution planning of perishable goods. International Journal of Production Economics 146(1):208 – 218

Armstrong R, Gao S, Lei L (2008) A zero-inventory production and distribution problem with a fixed customer sequence. Annals of Operations Research 159(1):395–414

Bard JF, Nananukul N (2009) The integrated productioninventorydistributionrouting problem. Journal of Scheduling

Chen HK, Hsueh CF, Chang MS (2009) Production scheduling and vehicle routing with time windows for perishable food products. Computers & Operations Research 36(7):2311 – 2319

Chen ZL, Vairaktarakis GL (2005) Integrated scheduling of production and distribution operations. Management Science 51(4):pp. 614–628

Graham R, Lawler E, Lenstra J, AHG RK (1979) Optimization and approximation in deterministic machine scheduling: a survey. Annals of Discrete Mathematics 5:287326

Hurter AP, Buer MGV (1996) The newspaper productiondistribution problem. Journal of Business Logistics 17(1):85–107

Pyke DF, Cohen MA (1990) Effects of flexibility through set-up time reduction and expediting on integrated production and distribution systems. IEEE Transactions on Robotics and Automation

Pyke DF, Cohen MA (1994) Multiproduct integrated production-distribution systems. European Journal of Operational Research

Seyedhosseini SM, Ghoreyshi SM (2014) An integrated model for production and distribution planning of perishable products with inventory and routing considerations. Mathematical Problems in Engineering

MISTA 2015

Integrated production and outbound distribution scheduling problem with setup times and delivery time windows

Liang-Liang Fu · Mohamed Ali Aloulou · Chefi Triki

Abstract In this paper, we study a production and outbound distribution scheduling problem in a company working in the metal packaging industry. In this problem, a set of jobs has to be processed on unrelated parallel machines with job splitting and sequence-dependent setup time (cost). Then the finished products are delivered in batches to several customers with heterogeneous vehicles subject to delivery time windows. The objective of production is to minimize the total setup cost and the objective of distribution is to minimize the transportation cost. We propose mathematical models for decentralized scheduling problems and integrated scheduling problem. We develop a two-phase iterative heuristic to solve the integrated scheduling problem. We evaluate the benefit of coordination through numerical experiments.

Keywords IPODS, setup times, delivery time windows, two-phase iterative heuristic.

1 Introduction

In recent decades, globalization expands supply chain over national boundaries and brings a fierce competition market. In order to satisfy customers' heightened expectations, the enterprises increasingly find that they must rely on effective supply chains. A non-efficient supply chain may carry a high cost. For example, according to Eurostat data 2012 (Palmer et al. 2012), about 24% of all road freight kilometers driven in Europe are empty vehicles and the average vehicle is loaded to 56% of its capacity in terms of weight.

Liang-Liang Fu
PSL, Université Paris-Dauphine, 75775 Paris Cedex 16, France
CNRS, LAMSADE UMR 7243
E-mail: fuliangliang1984@hotmail.com

Mohamed Ali Aloulou
PSL, Université Paris-Dauphine, 75775 Paris Cedex 16, France
CNRS, LAMSADE UMR 7243
E-mail: aloulou@lamsade.dauphine.fr

Chefi Triki
Dept. of Mechanical and Industrial Engineering, Sultan Qaboos University, Muscat, Oman
Dept. of Engineering for Innovation, University of Salento, Lecce, Italy
E-mail: chefi.triki@unisalento.it

As production and distribution are the main business processes in supply chain, the coordination of production and distribution issue is crucial in supply chain management. The integrated production and outbound distribution scheduling (IPODS) issue has been investigated from 1980. This issue investigates the integration of production scheduling decision-making and outbound distribution decision-making at the operational level. Chen (2010) provided an extensive review of the literature on IPODS problems. In this paper, we study an IPODS problem in a company working in the metal packaging industry. In this problem, we consider the unrelated parallel machines production context with job splitting and job sequence-dependent setup time (cost) and the vehicle routing transportation context with time windows and heterogeneous vehicles. The manufacturer decides the production and distribution schedules and outsources the transportation to transporters.

While the vehicle routing problem (VRP) has been well studied in the literature, a handful of articles investigated the IPODS problem with routing delivery. In practice, the majority of deliveries involve multiple customers and adopt the groupage transport to save transportation costs. Hence the IPODS problem involving vehicle routing deserves a further research. In the literature, Chen (2010) reviewed the papers which have studied IPODS problems involving vehicle routing. These problems are intractable in many cases because they contain the strongly NP-hard traveling salesman problem (TSP). Most of these papers (Chen and Vairaktarakis 2005, Li et al. 2005, Chen 2010)considered single machine and equal size jobs. Few papers considered identical parallel machines (Chen and Vairaktarakis 2005, Ullrich 2013).

In practice, the jobs are required by the customers to be delivered in fixed delivery time windows. The IPODS problems involving vehicle routing with time windows have been studied in the last five years. Chen (2009) investigated an IPODS problem with routing delivery and time windows for perishable food products. In this problem, the products of each delivery batch are produced continuously on a single machine and are delivered to customers within soft time windows. The demands are assumed stochastic and the deterioration of products throughout their lifetime is considered. The objective is to maximize the expected total profit of the supplier. He proposed an algorithm composed of the constrained Nelder-Mead method (Nelder and Mead 1965) and a heuristic for the vehicle routing problem with time windows. Ullrich (2013) investigated the problem where a set of jobs of general size is processed on identical parallel machines subject to the machine release time, and delivered to customers within the time windows by a fleet of heterogeneous vehicles. The objective is to minimize the sum of tardiness. They provided a genetic algorithm for the integrated problem and evaluated its performance by comparing with two classical decomposition approaches. Low et al. (2013) provided an integer nonlinear programming model and two adaptive genetic algorithms for the problem where retailers' jobs are processed in a distribution center and delivered to customers by a fleet of homogeneous vehicles within the time windows. The objective is to minimize the time required to complete producing the product, delivering it to retailers and returning to the distribution center. Low et al. (2014) provided an integer nonlinear programming model and two adaptive genetic algorithms for the same problem with the consideration of heterogeneous vehicles and soft time windows. The objective is to minimize the total cost including the transportation cost, the penalty cost of earliness and the penalty cost of tardiness.

This paper deals with a new IPODS problem in a real industrial context. The most related paper is that of Ullrich (2013). Different from his paper, we consider the unrelated parallel machines with job splitting and job sequence-dependent setup time

(cost) and the different objective functions (total setup cost and transportation cost). In our paper, we first investigate the decentralized scheduling problems, i.e. the production scheduling problem and the distribution scheduling problem, and propose two mixed integer linear programming (MILP) models for these two NP-hard problems. Then we develop a non-linear programming model and a two-phase iterative heuristic to solve the integrated scheduling problem. Finally, we evaluate the benefit of coordination through numerical experiments.

This paper is organized as follows. In section 2, we formally describe the problems and introduce notations and terminology. Section 3 is devoted to the decentralized scheduling problems, and section 4 to the integrated scheduling problem. In section 5, we evaluate the benefit of coordination through numerical experiments. Section 6 contains some conclusions and propositions for future research.

2 Problems and Notations

A set of jobs $N = \{1, \ldots, n\}$ has to be processed on a set of unrelated parallel machines $M = \{1, \ldots, m\}$. Job $j \in N$ requires the processing of q_j identical items on machines. Each job can be processed on any machine. We consider job splitting in production, in that each job can be split into parts and processed independently on several machines at a same time. There is no preemption of jobs on each machine. Let p_j^e denote the processing time of unit item of job $j \in N$ on machine $e \in M$. Let C_j denote the completion time of job $j \in N$. Let ϱ_j denote the delivery destination of job $j \in N$. Two different jobs may have the same delivery destination. Moreover machine $e \in M$ has a release time γ^e.

On one machine, a sequence-dependent setup time and a setup cost occur when production changes from one job to another. Let s_{0j} denote the setup time of job $j \in N$ which is processed as the first job on one machine. Let $s_{j_1 j_2}$ denote the setup time when production changes from job j_1 to job j_2 and $s_{j_1 j_2} = 0$ if $j_1 = j_2$, $j_1, j_2 \in N$. The setup times respect the triangle rule, i.e. $s_{j_1 j_2} + s_{j_2 j_3} \geq s_{j_1 j_3}$, $j_1, j_2, j_3 \in N$. The setup cost is proportional to the setup time. Let ρ be the cost for unit setup-time. Hence $\rho s_{j_1 j_2}$ is the setup cost when production changes from job $j_1 \in N$ to job $j_2 \in N$.

After completion of job $j \in N$, job j is delivered to its destination ϱ_j at its delivery time window $[a_j, b_j]$. If one delivery vehicle arrives before the delivery time window, it should wait until time a_j to unload. Hence the delivery time is also the beginning time of unloading. We consider batch delivery, i.e. several jobs can be delivered in one shipment. There is a set of vehicles denoted by K, consisting of several types of vehicles. For each type of vehicles there are a sufficient number of vehicles which is equal to n. Any job can be delivered by any type of vehicle. Vehicle $k \in K$ has a capacity Q^k, which is measured by the number of pallets. Let ϕ_j be the number of pallets to deliver job $j \in N$. One pallet cannot contain more than one job.

Let τ_{0j} denote the transportation time from the plant to the destination of job j. τ_{0j} includes the loading time. Let $\tau_{j_1 j_2}$ denote the transportation time from the destination of job $j_1 \in N$ to the destination of job $j_2 \in N$. Let T denote the constant unloading time of a job at its destination. $\tau_{j_1 j_2} = 0$ if $\varrho_{j_1} = \varrho_{j_2}$. The transportation times respect the triangle rule, i.e. $\tau_{j_1 j_2} + \tau_{j_2 j_3} \geq \tau_{j_1 j_3}$, $j_1, j_2, j_3 \in N$.

There are two types of transportation: direct delivery from the plant to one destination; routing delivery from the plant to several destinations in one shipment. There is

a limit of duration of any shipment, denoted by L. This value represents the maximum duration of utilization of the resources (vehicle and driver) for any shipment.

In a direct delivery, the transportation cost from the plant to the destination of job $j \in N$ with vehicle $k \in K$, is denoted by h_{0j}^k.

In a routing delivery, the transportation cost is equal to the most expensive direct delivery cost of one job among the jobs of this shipment plus the total drop costs. A drop cost occurs when we deliver more than one destination in one shipment. This is the current transportation costing system adopted by our considered company and the collaborating transporters. Let φ^k denote the drop cost per destination with vehicle $k \in K$. For example, vehicle k delivers three jobs j_1, j_2, j_3 to three different destinations, such that $h_{0j_1}^k < h_{0j_2}^k < h_{0j_3}^k$. The transportation cost of this delivery is equal to the direct delivery cost of job j_3 plus two drop costs for jobs j_1 and j_2, i.e. $h_{0j_3}^k + 2\varphi^k$. Clearly, the direct delivery is a special case of the routing delivery.

Let σ denote a *production schedule* that specifies how to assign each job on machines and when each job is processed on its assigned machine(s). Let θ denote a *delivery schedule* that specifies which vehicles are used, which jobs are in each batch, when each batch departs, and what is the traveling route for each batch. Let (σ, θ) denote an *integrated schedule* that specifies a production schedule and a delivery schedule.

The objective of production is to minimize the total setup cost, denoted by SC. The objective of distribution is to minimize the transportation cost, denoted by TC, which is the sum of transportation costs of all batches.

We consider two scenarios: (1) the production schedule and delivery schedule are determined in a consecutive order (i.e. first production, then delivery); (2) the production schedule and delivery schedule are decided concurrently. The scheduling problems are formally defined as follows.

1. **Decentralized scheduling problems.**
   (a) **Production scheduling problem.** The problem is to determine a production schedule minimizing SC subject to the production deadlines $\overline{d}_j = b_j - \tau_{0j}$ which guarantee that the jobs can be delivered in their delivery time window. We follow the three-field classification $\alpha|\beta|\gamma$ introduced by Graham et al. (1979). This is a production scheduling problem minimizing the total setup cost (SC) with unrelated parallel machines (R), machine release times (γ^e), job splitting ($split$), production deadlines ($\overline{d}_j$) and sequence-dependent setup times (s_{ij}), denoted by $R, \gamma^e | split, \overline{d}_j, s_{ij} | SC$.
   (b) **Distribution scheduling problem.** The problem is to determine a delivery schedule minimizing TC subject to the job release dates imposed by production schedule and delivery time windows. The production completion time of each job imposes a job release date for delivery. The problem is a heterogeneous vehicle routing problem with time windows and release dates (HVRPTWRD).
2. **Integrated scheduling problem.** The problem is to determine an integrated schedule minimizing $SC + TC$ subject to machine availability constraints and delivery time windows. Using the five-field notation proposed by Chen (2010), the integrated scheduling problem can be denoted by $R, \gamma^e, split | [a_j, b_j] | V(\infty, Q^k), routing | u | SC + TC$, where R means the unrelated parallel machines, $split$ means the job splitting in production, $[a_j, b_j]$ means the delivery time windows, $V(\infty, Q^k)$ and $routing$ mean the routing delivery with sufficient heterogeneous vehicles, and $u \leq n$ represents the number of customers.

Example: To illustrate the decentralized and integrated scheduling problems, we consider the following example.

- Number of jobs $n = 2$, number of machines $m = 2$.
- Quantity of items of job $j \in N$: $q_j = 25$.
- Processing time of unit item of job $j \in N$ on machines: $p_j^1 = 1$ and $p_j^2 = 2$.
- Setup times: $s_{01} = 5$, $s_{02} = 4$, $s_{12} = s_{21} = 3$.
- Cost of unit setup time: $\rho = 100$.
- Machines release times are zero.
- There are 2 identical vehicles with capacity of 20 pallets.
- Number of pallets to deliver job $j \in N$: $\phi_j = 10$.
- Unloading time $T = 1$, and limit of duration of shipment $L = 30$.
- Transportation times: $\tau_{01} = 10$, $\tau_{02} = 15$, $\tau_{12} = \tau_{21} = 12$.
- Delivery time windows: $[a_1, b_1] = [50, 60]$ and $[a_2, b_2] = [60, 70]$.
- Direct transportation costs with vehicle $k \in K$: $h_{01}^k = 750$ and $h_{02}^k = 1000$.
- Drop costs of vehicle $k \in K$: $\psi^k = 100$.

1. **Decentralized scheduling problems.** Figure 1 illustrates an optimal production schedule for the production scheduling problem in the decentralized model. With $C_1 = 30$ and $C_2 = 54$, two jobs cannot be delivered in one shipment because of the deadline of job 1. Hence we have $SC = 100*(5+4) = 900$, $TC = 750+1000 = 1750$, and the total cost is equal to $SC + TC = 2650$.

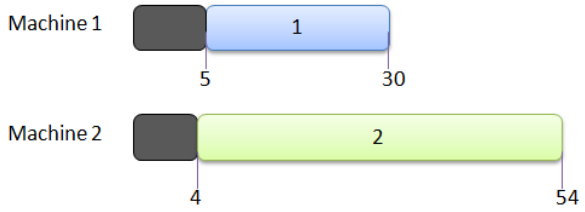


Fig. 1 Optimal production schedule in the decentralized model

2. **Integrated scheduling problem.** Figure 2 illustrates a production schedule for the integrated scheduling problem. In this schedule, job 2 is split: 10 items are processed on machine 1 and 15 items are processed on machine 2. The setup cost increases to $SC = 100 * (5 + 4 + 3) = 1200$. With $C_1 = 30$ and $C_2 = 43$, the two jobs can be delivered in one shipment: the shipment departs at time 43, drops job 1 at time 53 and reaches the destination of job 2 at time 66. Hence we have $TC = 1000 + 100 = 1100$, and the total cost is equal to $SC + TC = 2300$. The benefit of coordination is 13.2% with respect to the total cost.

3 Decentralized Scheduling Problems

In the decentralized scenario, the production schedule and delivery schedule are determined consecutively. We provide a mixed integer linear programming (MILP) model for each decentralized scheduling problem.

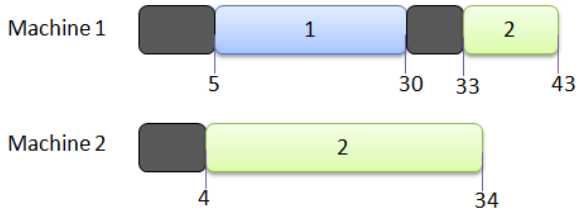


Fig. 2 Optimal production schedule in the integrated model

3.1 Production Scheduling Problem

We recall that the single machine scheduling problem $1|s_{ij}|C_{\max}$ is NP-hard (Bruno and Downey 1978). Since this single machine scheduling problem is a special case of our production scheduling problem, the production scheduling problem is also NP-hard. Recall that the deadline $\bar{d}_j$ of job $j \in N$ is equal to $b_j - \tau_{0j}$.

We provide a MILP model, which is similar to the model of Zhu and Heady (2000) proposed for a similar problem without job splitting, without machine release times and with a different objective function (job earliness and tardiness). We introduce two fictive jobs 0 and $n+1$. The decision variables are defined as follows.

$$X_{ij}^{e} = \begin{cases} 1, \text{ if job } i \text{ is the direct predecessor of job } j \text{ on machine } e, i = 0, \ldots, n, \\ \quad j = 1, \ldots, n+1, i \neq j, e \in M \\ 0, \text{ otherwise} \end{cases}$$

Y_j^e = number of items of job j processed on machine e, $j \in N$, $e \in M$.

C_j = completion time of job j, $j \in N$.

MILP1:

$$\min \rho \sum_{e=1}^{m} \sum_{i=0}^{n} \sum_{j \in N, j \neq i} s_{ij} X_{ij}^{e} \tag{1}$$

$$\text{s.t.} \sum_{i=0,\ldots,n, i \neq j} X_{ij}^{e} \leq 1, \;\; j = 1, \ldots, n+1, e \in M \tag{2}$$

$$\sum_{i=0,\ldots,n, i \neq j} X_{ij}^{e} - \sum_{g=1,\ldots,n+1, g \neq j} X_{jg}^{e} = 0, \;\; j \in N, e \in M \tag{3}$$

$$\sum_{i=0,\ldots,n, i \neq j} X_{ij}^{e} q_j \geq Y_j^e, \;\; j \in N, e \in M \tag{4}$$

$$\sum_{i=0,\ldots,n, i \neq j} X_{ij}^{e} \leq Y_j^e, \;\; j \in N, e \in M \tag{5}$$

$$\sum_{e=1}^{m} Y_j^e = q_j, \;\; j \in N \tag{6}$$

$$C_j \leq \overline{d}_j, \;\; j \in N \tag{7}$$

$$C_j - C_i \geq p_j^e Y_j^e + X_{ij}^e s_{ij} + (X_{ij}^e - 1)(p_j^e q_j + \max\{b_i, b_j\}), \;\; i, j \in N, i \neq j, e \in M \tag{8}$$

$$C_j \geq X_{0j}^e (\gamma^e + s_{0j}) + p_j^e Y_j^e, \;\; j \in N, e \in M \tag{9}$$

$$X_{ij}^e \in \{0, 1\}, \;\; i = 0, \ldots, n, j = 1, \ldots, n+1, i \neq j, e \in M \tag{10}$$

$$Y_j^e \in \mathbb{N}, \;\; j \in N, e \in M \tag{11}$$

The objective function (1) minimizes the total setup cost. Constraints (2) ensure that one job is processed on each machine once at most. Constraints (3) impose that for each job $j \in N$, the number of its direct predecessors is equal to the number of its direct successors on each machine. Constraints (4)-(5) impose the relation between variables X_{ij}^e and Y_j^e: if $Y_j^e > 0$, then $\sum_{i=0,\ldots,n,i\neq j} X_{ij}^e > 0$, otherwise $\sum_{i=0,\ldots,n,i\neq j} X_{ij}^e = 0$. Constraints (6) ensure that all jobs are processed. Constraints (7) enforce the job deadline restriction. In constraints (8), if job i precedes job j on machine e, i.e., $X_{ij}^e = 1$, we ensure that the completion time of job j is far enough after that of job i to include the processing time of processed parts of job j and setup time for job j on machine e. Otherwise, i.e., $X_{ij}^e = 0$, we have $C_j - C_i \geq -\max\{b_i, b_j\} \geq p_j^e Y_j^e - p_j^e q_j - \max\{b_i, b_j\}$, hence constraints (8) are always valid in this case. Constraints (9) enforce the machine release time restriction. Constraints (10)-(11) give the domain of definition of each variable.

With the MILP1, we find an optimal production schedule minimizing SC. Because of constraints (8), there may exist some unnecessary idle times between jobs. Hence, we remove all idle times in this obtained production schedule and update C_j for $j \in N$.

3.2 Distribution Scheduling Problem

In the distribution scheduling problem, if the delivery destinations are given for one shipment, the transportation cost of this shipment is fixed. Because of this difference

from the classical vehicle routing problem with time windows (VRPTW), we need to prove the complexity of the distribution scheduling problem. We consider the following special case of the distribution scheduling problem:

- each customer has one job only,
- the delivery time window $[a_j, b_j] = [0, \infty]$ for $j \in N$,
- the production completion time $C_j = 0$ for $j \in N$,
- the limit of length of a trip $L = \infty$,
- the vehicles are identical,
- the transportation costs $h^k_{0j_1} = h^k_{0j_2} = h^k$ for $j_1, j_2 \in N$ and $k \in K$.

Let B denote the number of delivery batches. In this case, we have the following equation for the overall transportation cost TC of a delivery schedule with B batches.

$$TC = Bh^k + (n - B)\varphi^k \tag{12}$$

Hence the objective of minimizing TC is equivalent to the objective of minimizing the number of delivery batches B. This special case is the bin packing problem which is NP-hard in the strong sense (Garey and Johnson 1979). In the bin packing problem, objects of different volumes must be packed into a finite number of bins of equal size in a way that minimizes the number of bins used. With the similar argument, we observe that several special cases of this problem are NP-hard in the strong sense. So the distribution scheduling problem is NP-hard in the strong sense.

Then, we provide a multicommodity network flow MILP model similar to that of Desrochers et al. (1988) proposed for a classical VRPTW.

This problem can be defined on a direct graph $G = (V, A)$, where $V = \{0, 1, \ldots, n+1\}$. The vertex $j \in \{1, \ldots, n\}$ represents the destination ϱ_j of job j. The vertexes 0 and $n + 1$ represent the manufacturing plant and one fictive ending point, denoted by ϱ_0 and ϱ_{n+1} respectively. The arcs represent the paths between two places. Feasible vehicle routes correspond to paths starting at vertex 0 and ending at vertex $n + 1$. We set service time $\psi_i = T$ for vertex $i \in N$ and $\psi_0 = \psi_{n+1} = 0$ for vertexes 0 and $n + 1$. τ_{ij} is the travel time from vertex i to vertex j. We set $\tau_{i,n+1} = 0$, for $i \in \{0, \ldots, n\}$. Each vertex i is associated to a time window $[a_i, b_i]$. Moreover we set $a_0 = \min_{i \in N} C_i$, $b_0 = \max_{i \in N}\{b_i - \tau_{oi}\}$, $a_{n+1} = \min_{i \in N}\{a_i + \psi_i\}$ and $b_{n+1} = \max_{i \in N}\{b_i + \psi_i\}$. Here, completion time C_j of job $j \in N$ is given by the manufacturer. Arc (i, j), for $i, j \in N$ and $i \neq j$, exists only if

(a) job i can be delivered before job j on respecting their delivery time window, i.e. $a_i + \psi_i + \tau_{ij} \leq b_j$,
(b) the shipment including jobs i and j respects the limit of length of a shipment, i.e. $\tau_{0i} + \psi_i + \max\{\tau_{ij}, a_j - b_i - \psi_i\} \leq L$,
(c) the completion time of job j is no later than the latest possible departure date of the shipment including jobs i and j, i.e. $b_i - \tau_{0i} \geq C_j$.

There exists an arc from vertex 0 to each other vertex, and from each other state to vertex $n + 1$. Let $\delta^+(i) = \{j : (i, j) \in A\}$ and $\delta^-(j) = \{i : (i, j) \in A\}$. We define the decision variables as follows.

$$x^k_{ij} = \begin{cases} 1, \text{ if arc } (i,j) \text{ is used by vehicle } k, (i,j) \in A, k \in K \\ 0, \text{ otherwise} \end{cases}$$

w^k_i = starting time of unloading of vehicle k at vertex i, $i \in V$,$k \in K$.

H^k = transportation cost of the trip accomplished by vehicle k, $k \in K$. Remark that since the number of vehicles of each type is sufficient, we can suppose that each vehicle is assigned at most to one trip.

MILP2:

$$\min \sum_{k \in K} H^k \tag{13}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{j \in \delta^+(i)} x_{ij}^k = 1, \ \ i \in N \tag{14}$$

$$\sum_{j \in \delta^+(0)} x_{0j}^k = 1, \ \ k \in K \tag{15}$$

$$\sum_{i \in \delta^-(j)} x_{ij}^k - \sum_{i \in \delta^+(j)} x_{ji}^k = 0, \ \ k \in K, j \in N \tag{16}$$

$$w_j^k \geq w_i^k + x_{ij}^k(\psi_i + \tau_{ij}) - (1 - x_{ij}^k)b_i, \ \ k \in K, (i,j) \in A \tag{17}$$

$$a_i \leq w_i^k \leq b_i, \ \ k \in K, i \in V \tag{18}$$

$$\sum_{i \in N} \sum_{j \in \delta^+(i)} x_{ij}^k \phi_i \leq Q^k, \ \ k \in K \tag{19}$$

$$w_0^k \geq C_j \sum_{i \in \delta^-(j)} x_{ij}^k, \ \ j \in N, k \in K \tag{20}$$

$$w_j^k - w_0^k \leq L + (1 - \sum_{i \in \delta^-(j)} x_{ij}^k)b_j, \ \ j \in N, k \in K \tag{21}$$

$$H^k \geq h_{0j}^k \sum_{i \in \delta^-(j)} x_{ij}^k + \varphi^k(\sum_{(u,v) \in A, \varrho_u \neq \varrho_v} x_{uv}^k - 2) \ \ j \in N, k \in K \tag{22}$$

$$x_{ij}^k \in \{0,1\}, \ \ k \in K, (i,j) \in A \tag{23}$$

$$H^k \geq 0, \ \ k \in K \tag{24}$$

The objective function (13) minimizes the transportation cost. Constraints (14) ensure that one job is delivered once. Constraints (15) ensure that one vehicle is used once. Constraints (16) state that the solution satisfy the flow conservation at each vertex. Constraints (17)-(18) ensures that each job is delivered at its destination in the delivery time windows. Constraints (19)-(20) enforce the vehicle capacity restriction and the job availability restriction. Constraints (21) enforce the delivery length restriction. Constraints (22) calculate the transportation cost. Constraints (23)- (24) give the domain of definition of each variable.

4 Integrated Scheduling Problem

The integrated scheduling problem is to minimize $SC+TC$ subject to machine availability constraints and delivery time windows. The objective is to optimize the performance of the global supply chain.

In what follows, we propose a nonlinear programming model and a two-phase iterative heuristic to solve the integrated scheduling problem. Since the individual scheduling problems are NP-hard, the integrated scheduling problem is also NP-hard.

4.1 Nonlinear programming model

We combine MILP1 and MILP2 to construct a nonlinear programming model for the integrated scheduling problem. We ignore the rule (c) of existence of arc in MILP2, which makes that the existence of arc in MILP2 is independent on solution of MILP1. Since now C_j and x_{ij}^k are both decision variables, this model is nonlinear.

NLP6.1:

$$\min \rho \sum_{e=1}^{m} \sum_{i=0}^{n} \sum_{j \in N, j \neq i} s_{ij} X_{ij}^e + \sum_{k \in K} H^k \tag{25}$$

$$\text{s.t.} \quad (2) - (11)$$

$$(14) - (24)$$

4.2 Two-phase iterative heuristic

Absi et al. (2014) proposed a two-phase iterative heuristic to solve an integrated problem considering the integration of production planning and vehicle routing decisions. They considered the production planning instead of the production scheduling, and the vehicle routing problem such that there are no delivery time windows. We propose a similar two-phase iterative heuristic (see algorithm 1) to solve our integrated scheduling problem.

Algorithm 1: Two-phase iterative heuristic

```
Initialize F_v^j = 0 and η_j = 1, for j ∈ N and v ∈ I_j;
while ending criterion do
    Solve the production scheduling problem;
    Remove all idles times and update C_j for j ∈ N;
    Solve the distribution scheduling problem with fixed C_j for j ∈ N;
    Update the best solution so far;
    Update F_v^j, for j ∈ N and v ∈ I_j;
    Update η_j for j ∈ N;
```

In the first phase, we solve a production scheduling problem in which an approximate of the transportation cost is integrated. We first determine the discrete possible delivery times in each delivery time windows, like $\{a_j, a_j + 1, \ldots, b_j\}$ for the delivery time window $[a_j, b_j]$ and $j \in N$. Let I_j denote a set of indexes of possible delivery times of job $j \in N$ and t_v^j denote the v^{th} possible delivery time for $v \in I_j$. In order to evaluate the transportation cost, we introduce F_v^j to represent an estimation of the transportation cost of job j if job j is delivered at time t_v^j, $j \in N$ and $v \in I_j$. We introduce a decision variable λ_v^j which is equal to 1 if t_v^j is chosen, and 0 otherwise. Moreover

we introduce a parameter η_j for $j \in N$ to modify the completion time constraints (see constraints 27 of MILP3). In the obtained production schedule of the first phase, we remove all idle times on each machine and update C_j for $j \in N$. The anticipation of completion times offers a better input for the second phase.

In the second phase, we solve the distribution scheduling problem with fixed C_j. According to the solution of the second phase, we update F_v^j and η_j for next iteration. The procedure stops when a fixed number of iterations is reached or the solution is not improved for a fixed number of iterations.

In the following, we give the detail of the production phase and the distribution phase.

4.2.1 The production phase

In this phase, we propose a MILP to solve the production scheduling problem in which an approximate of transportation cost is integrated. The decision variable λ_v^j is equal to 1 if t_v^j is chosen for $j \in N$ and $v \in I_j$, and 0 otherwise. The other decision variables are as introduced in MILP1.

MILP3:

$$\min \rho \sum_{e=1}^{m} \sum_{i=0}^{n} \sum_{j \in N, j \neq i} s_{ij} X_{ij}^e + \sum_{j=1}^{n} \sum_{v \in I_j} F_v^j \lambda_v^j \tag{26}$$

$$\text{s.t.} \quad C_j \leq \eta_j (\lambda_v^j t_v^j + (1 - \lambda_v^j) b_j - \tau_{0j}), \;\; j \in N, v \in I_j \tag{27}$$

$$\sum_{v \in I_j} \lambda_v^j = 1, \;\; j \in N \tag{28}$$

$$\lambda_v^j \in \{0, 1\}, \;\; j \in N, v \in I_j \tag{29}$$

$$(2) - (6)$$

$$(8) - (11)$$

The objective function (26) minimizes the sum of the total setup cost and the approximate transportation cost. Constraints (27) ensure that if $\lambda_v^j = 1$, the completion time $C_j \leq \eta_j(t_v^j - \tau_{0j})$, otherwise $C_j \leq \eta_j(b_j - \tau_{0j})$. And η_j is a parameter to control the degree to force the reduction of completion time. Initially, η_j is equal to 1 and is reduced at the end of each iteration. Constraints (28) ensure that each job is delivered exactly once.

At the end of the first phase, in the obtained production schedule, we remove all idle times and update C_j for $j \in N$.

4.2.2 The distribution phase

In this phase, we use MILP2 to solve the HVRPTWRD with fixed C_j for $j \in N$. According to the solution of the second phase, we update F_v^j and η_j (see algorithm 2).

In algorithm 2, we update F_v^j with the consideration of two cases:

1. If job j is visited by vehicle k at time $t_{v'}^j$ where $v' \in I_j$, replacing delivery time $t_{v'}^j$ by t_v^j in the trip is allowed if (lines 5-8):
   - The transportation times from its direct predecessor to job j, and from job j to its direct successor are respected.

Algorithm 2: Procedure of updating F_v^j and η_j

```
1  for j ∈ N do
2      for v ∈ I_j do
3          F_v^j = ∞;
4          for k ∈ K do
5              if job j is visited by vehicle k at time t_{v'}^j, where v' ∈ I_j, and the delivery
               time of job j in vehicle k can be replaced by t_v^j then
6                  B^k = set of jobs delivered by vehicle k;
7                  H_1 = cheapest cost to deliver the jobs of B^k \ {j};
8                  F_v^j = min{F_v^j, H^k − H_1};
9              if job j is not visited by vehicle k and can be inserted in the trip of vehicle
               k and visited at time t_v^j then
10                 B^k = set of jobs delivered by vehicle k;
11                 H_2 = cheapest cost to deliver the jobs of B^k ∪ {j};
12                 F_v^j = min{F_v^j, H_2 − H^k};
13 for j ∈ N do
14     if 0.8η_j(b_j − τ_{0j}) ≥ min_{e∈M}(γ^e + p_j^e q_j) + s_{0j} and job j is delivered by a vehicle of
       which the number of delivered pallets is less than or equal to
       max_{k∈K} Q^k − min_{i∈N} φ_i then
15         η_j = 0.8η_j;
```

- The new trip does not violate the limit of length of a trip.

If the conditions are satisfied, we update F_v^j by $\min\{F_v^j, H^k - H_1\}$, where $H^k - H_1$ represents the transportation cost to deliver job j. Remark that if $v = v'$, the above conditions are satisfied.

2. If job j is not visited by vehicle k, the insertion of job j in vehicle k at time t_v^j is allowed if (lines 9-12):
   - The largest capacity among all vehicles, i.e. $\max_{g\in K} Q^g$, allows.
   - There exists two successively visited vertexes which allow the insertion of job j with delivery time t_v^j, i.e., the transportations times are respected.
   - The new trip does not violate the limit of length of a trip.

   If the conditions are satisfied, we update F_v^j by $\min\{F_v^j, H_2 - H^k\}$, where $H_2 - H^k$ represents the transportation cost to deliver job j.

Moreover, we explain how to find the cheapest cost to deliver a set of jobs B (lines 7 and 11). For given B and vehicle $k \in K$, the corresponding transportation cost is fixed. Hence we choose the cheapest vehicle to deliver this set of jobs.

After the consideration of all vehicles, F_v^j represents the cheapest transportation cost to deliver job j at time t_v^j for $j \in N$ and $v \in I_j$ with the consideration of the obtained delivery schedule.

Concerning the parameter η_j for $j \in N$, if η_j is small enough or job j is delivered by a vehicle in which the size of delivered jobs is close to the largest vehicle capacity, we do not change η_j, otherwise we reduce η_j by 20%.

In the distribution schedule obtained in the second phase, F_v^j approximates the transportation cost if job j is delivered at time t_v^j, $j \in N$ and $v \in I_j$. In next iteration, for each job $j \in N$, the algorithm may choose another delivery time with smaller transportation cost. This modification of delivery time of each job can influence the

production completion time of each job in the first phase (see constraints 27 of MILP3). The objective of the reduction of η_j is to force the algorithm to reduce the production completion time of job j and increase the opportunity to find a better transportation cost in the second phase. Finally, we note that there is no guarantee that the heuristic produces an optimal solution.

5 Computational Results

In this section, we evaluate the feasibility of the two-phase iterative heuristic and the potential benefit of coordination. We propose an approach of generation of instances inspired by the company data. We analyze the results for small and medium instances.

The benefit of coordination is measured by comparing the integrated schedule generated by the heuristic with the decentralized schedules generated by MILP1 and MILP2. The algorithms are implemented in C++ and Cplex V12.5.1. The experiments are carried out on a DELL 2.50GHz personal computer with 8GB RAM.

We consider $n \in \{5, 10, 15, 20\}$ and $m \in \{2, 3, 5\}$. The integers q_j and ϕ_j, for $j \in N$, are generated from the uniform distributions [50,200] and $[q_j/10, q_j/5]$ respectively. The processing times of unit item p_j^e, for $j \in N$ and $e \in M$, are generated from the uniform distribution [0.01, 0.1]. The machines release times γ^e, for $e \in M$, are generated from the uniform distribution [0,4]. The integers s_{ij}, for $i = 0, \ldots, n$, $j \in N$, are generated from the uniform distribution $[0.1 \min\{\max_{e\in M} p_i^e q_i, \max_{e\in M} p_j^e q_j\}, 0.5 \min\{\max_{e\in M} p_i^e q_i, \max_{e\in M} p_j^e q_j\}]$. We set $s_{ij} = 0$ for $i = j$. In order to guarantee the triangle property, after generation, if $s_{ij} \leq S/2$, where S is the maximum generated setup time, we regenerate another $s_{ij} \in]S/2, S]$. The cost per unit setup time ρ is equal to 100.

We suppose that each customer has only one job. The customers are divided into two groups, $N_1 = \{1, \ldots, n/2\}$ and $N_2 = \{n/2+1, \ldots, n\}$. The integers τ_{0j}, for $j \in N_1$ are generated from the uniform distribution $[15, 19]$ and for $j \in N_2$ from $[25, 29]$. If i, j are in the same group, the integers τ_{ij} are generated from the uniform distribution $[4, 6]$, otherwise from $[15, 19]$. We set $\tau_{ij} = \tau_{ji}$ and set $\tau_{ij} = 0$ if $\varrho_i = \varrho_j$. The limit length of a trip $L \in \{45, 60\}$. The unloading time $T = 1$. The integer lower bounds of time windows a_j, for $j \in N$, are generated from the uniform distribution $[B/2, B]$, where $B = \max_{e\in M} \gamma^e + \max_{e\in M} \sum_{j\in N} p_j^e q_j/m + L + 0.75nS$ represents an estimated delivery time if all jobs begin their processing at the latest machine release time, each job is split in m parts and processed on m machines, and the transportation time of the trip reaches the limit L. The integer upper bounds of time windows b_j, for $j \in N$, are generated from the uniform distribution $[a_j + \epsilon - 5, a_j + \epsilon]$, where $\epsilon_2 = \{10, 15\}$. There are two types of vehicles and $2n$ vehicles totally, i.e., $K = \{1, \ldots, 2n\}$. For $k \leq n$ and $j \in N$, $Q^k = 30$, $\varphi^k = 50$ and $h_{0j}^k = 50\tau_{oj}$. For $k > n$ and $j \in N$, $Q^k = 60$, $\varphi^k = 80$ and $h_{0j}^k = 80\tau_{oj}$. 10 instances are generated for each combination of parameters n, m, L and ϵ. Totally 480 instances are generated.

We impose 3 minutes as a limit of execution time of a single MILP. We generate the decentralized schedules in three steps: first apply MILP1 to create a production schedule, then remove the idle times in the obtained production schedule, and finally apply MILP2 to create a delivery schedule. We apply the two-phase iterative heuristic to generate an integrated schedule. Concerning the ending criterion, we set that the total number of iterations cannot exceed 6 and the number of iterations without improvement cannot exceed 3.

Table 1 Average execution times of heuristic

n	5	10	15	20
Time	3.34	238.28	952.30	929.29

Table 2 Benefit of coordination

	Improved			Average Benefit			Max Benefit		
n	$m=2$	$m=3$	$m=5$	$m=2$	$m=3$	$m=5$	$m=2$	$m=3$	$m=5$
5	25.00%	20.00%	30.00%	2.83%	2.40%	3.15%	21.43%	29.70%	24.28%
10	32.50%	20.00%	32.50%	1.16%	0.68%	1.00%	14.69%	6.47%	6.13%
15	57.50%	57.50%	67.50%	1.52%	1.41%	1.47%	6.61%	6.51%	5.29%
20	42.50%	77.50%	55.00%	1.09%	2.55%	1.63%	5.72%	7.90%	7.98%

Table 1 and Table 2 illustrate the execution times of our heuristic and the benefit of coordination. The measures are described as follows.

Time: the average CPU time in seconds to execute the heuristic.
Improved: the percentage of instances which has a positive benefit.
Benefit: the benefit of coordination measured by

$$\frac{SC_1 + TC_1 - SC_2 - TC_2}{SC_1 + TC_1} \tag{30}$$

where SC_1 and TC_1 are the values of objective functions of the decentralized schedules, and SC_2 and TC_2 are the values of objective functions of the integrated schedule.

From Table 1, one can observe that the average execution time of heuristic grows rapidly. When $n = 5$, all MILPs can be solved optimally in the given time. When $n \geq 15$, we observe a difficulty for MILP2 which cannot find an optimal solution in the given time.

From Table 2, when $n = 5$ and 10, we find that there exists the instance with a significant benefit which can reaches 29.7% when $n = 5$ and 14.69% when $n = 10$. At the same time, we find more than 67.5% of instances which cannot be improved. Since the MILPs can find an optimal solution for the instances with $n = 5$ and 10 in the given time, that means the decentralized schedules are the same as the integrated schedule for more than 67.5% of instances. That's why the average benefit of coordination is not significant. When $n = 15$ and 20, more than 42.5% of instances can be improved. However the efficiency of MILP2 impedes the improvement of the transportation cost in the two-phase iterative heuristic, which imposes a poor average benefit of coordination. The significant maximum benefit of coordination verifies the feasibility of the heuristic and the potential benefit of coordination. Moreover, in order to improve the efficiency of MILP2 in the distribution phase of the heuristic, we tested another time-expanded network flow-based model and found that the new MILP is less efficient than MILP2. For future research, it is interesting to develop an efficient exact algorithm or an efficient heuristic for the distribution phase of the two-phase iterative heuristic.

6 Conclusions

In this paper, we investigated a production and outbound distribution scheduling problem in an enterprise working in the packaging industry. We first proposed two MILP models for the decentralized scheduling problems. Then we provided a nonlinear programming model and a two-phase iterative heuristic for the integrated scheduling problem.

We also proposed an approach of generation of instances and evaluated the benefit of coordination through numerical experiments for small and medium instances. The significant maximum benefit of coordination verified the feasibility of the heuristic and the potential benefit of coordination.

We pointed out the need to improve the efficiency of the algorithm for the distribution phase of the two-phase iterative heuristic. In order to evaluate the performance of the two-phase iterative heuristic, one might develop a meta-heuristic and compare their performance.

References

1. Absi, N., Archetti, C., Dauzère-Pérès, S., Feillet, D.: A two-phase iterative heuristic approach for the production routing problem. Transportation Science (Published online July 11, 2014)
2. Bruno, J., Downey, P.: Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM Journal on Computing **7**(4), 393–404 (1978)
3. Chen, H.K., Hsueh, C.F., Chang, M.S.: Production scheduling and vehicle routing with time windows for perishable food products. Computers & Operations Research **36**(7), 2311 – 2319 (2009)
4. Chen, Z.L.: Integrated production and outbound distribution scheduling: Review and extensions. Operations Research **58**(1), 130–148 (2010)
5. Chen, Z.L., Vairaktarakis, G.L.: Integrated scheduling of production and distribution operations. Management Science **51**(4), 614–628 (2005)
6. Desrochers, M., Lenstra, J., Savelsbergh, M., Soumis, F.: Vehicle routing with time windows: Optimization and approximation. In: B. Golden, A. Assad (eds.) Vehicle Routing: Methods and Studies, Studies in Management Science and Systems, pp. 65–84. North-Holland (1988)
7. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
8. Li, K.P., Ganesan, V.K., Sivakumar, A.I.: Synchronized scheduling of assembly and multi-destination air-transportation in a consumer electronics supply chain. International Journal of Production Research **43**(13), 2671–2685 (2005)
9. Low, C., Chang, C.M., Li, R.K., Huang, C.L.: Coordination of production scheduling and delivery problems with heterogeneous fleet. International Journal of Production Economics **153**(0), 139 – 148 (2014)
10. Low, C., Li, R.K., Chang, C.M.: Integrated scheduling of production and delivery with time windows. International Journal of Production Research **51**(3), 897 – 909 (2013)
11. Nelder, J.A., Mead, R.: A simplex method for function minimization. Computer Journal **7**, 308–313 (1965)
12. Palmer, A., Saenz, M.J., Woensel, T.V., Ballot, E.: Characteristics of collaborative business models. CO3 position paper (2012)
13. Ullrich, C.A.: Integrated machine scheduling and vehicle routing with time windows. European Journal of Operational Research **227**(1), 152 – 165 (2013)
14. Zhu, Z., Heady, R.B.: Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. Computers & Industrial Engineering **38**(2), 297 – 305 (2000)

MISTA 2015

Stability of probabilistic scheduling problems with precedence constraints and weighted completion time objective

Zied Bouyahia[†]

Abstract We address the probabilistic problem of scheduling n tasks on m parallel identical machines under precedence constraints to minimize the probabilistic total weighted finishing time. Probabilities are associated explicitly with the data in the formulation of the problem to provide a natural model. Two strategies are proposed to solve the probabilistic problem $P_m|q_i,\ prec|\sum w_iC_i$: The first, called re-optimization, consists in solving separately each potential instance, the second called *a priori* strategy consists in updating a scheduling for the initial problem by means of a modification method. The aim of this paper is to define and to study the behavior of the *a priori* strategy relying on natural modification method. Special care is devoted to evaluate the performance of the reoptimization and the *a priori* strategies.

Keywords Probabilistic scheduling problem, precedence constraints, a priori strategy, reoptimization strategy.

1 Introduction

In classical formulation of a scheduling problem, we assume that the number of tasks to be scheduled is fixed. However, in several practical situations, this assumption does not modelize the real-world problems since it does not take into consideration the random aspect of data. Generally, classical deterministic combinatorial optimization problems (COPes) formulation fails to provide a realistic modelization of several challenging practical problems. Up to the 1990's, randomness used to be considered in COPes as a feature of the datum itself or in the relationship between data. For instance, probabilistic and stochastic studies dealing with scheduling problems has relied on deterministic data assumption and randomness has been associated with tasks characteristics (processing times, release time, etc...) or in the relationship between tasks (probability on an edge in a precedence graph).
In the practice, after solving a COP, we have to solve an exponential number of COPes which are simple variations of the initial problem due to the absence of a subset of data.

SOIE Laboratory - National School for Computer Studies , Tunis, Tunisia
E-mail: bouyahiazied@gmail.com
[†] Corresponding author

Then if the COP of size n is NP-Hard, we have to deal with 2^n NP-Hard problems. Attempting to overcome this difficulty, we use instead Probabilistic Combinatorial Optimization Problem (PCOP).
The approach of PCOP was initiated in [12] for the Traveling Salesman Problem (see also [5, 1, 2]). Since then, several PCOPes have been studied providing a generalization of classical COPes [4]. These studies have been essentially motivated, firstly, by the attempt to provide a model formulation more appropriate for several challenging real-world problems where randomness is present and, secondly, the interest in studying the impact of a perturbation of an initial problem which consists generally in the absence of data. In scheduling problems, after solving a particular instance, some tasks may not need to be processed because they are postponed or simply aborted. The initial problem data is slightly perturbed and we have a new instance of a scheduling problem. Intuitively, we think about "reusing" the previous solution trying to update it to solve the new instance. This situation can be perfectly modeled by a probabilities system on the data set.
In this work, we consider particularly the problem $P_m|prec|\sum w_iC_i$ in which we have to schedule n tasks subject to precedence constraints on m identical parallel machines minimizing the so-called "weighted mean finishing time". The relaxed counterpart of the probabilistic problem $P_m|q_i\ |\sum w_iC_i$ has been studied in [7] and [6]. The problem subject to precedence constraints is of a great interest in practice and, known as NP-Hard, it has been studied attempting to provide efficient and rapid solution by polynomial-time heuristics [18].
In this work, we propose a generalization of the classical $P_m|q_i,\ prec|\sum w_iC_i$ problem by emphasizing the impact of randomness in problem formulation. The proposed model consists in considering the number of tasks as a random variable varying between 0 and n instead of a deterministic value n.
Under this assumption, if some tasks are absent from the initial instance of n tasks, two strategies are possible to solve such a problem : The first, called reoptimization strategy, consists simply in treating separately each subset of the initial whole set of tasks, the second called a priori strategy updates, by means of a modification method, an a priori solution found for the full list of tasks to get a schedule for the present tasks. We define, hereafter, the two strategies and we study experimentally their behaviors. Special care is also devoted to the performance evaluation of the a priori strategy on synthetic benchmarks.

2 Problem definition

We address the problem of scheduling a set of n jobs (or tasks), $L_n = \{T_1, \ldots, T_n\}$, on a set of m identical parallel machines under precedence constraints. We assume that a machine can work on only one job at a time and that each job can be processed by at most one machine at a time. Each job T_i has a positive weight w_i and requires a processing time p_i on any machine. The start time of a scheduled task T_i is denoted by S_i and the completion time is denoted by C_i. An acyclic graph $G(L_n, U)$ can describe the precedence constraints in which an edge from T_i to T_j implies that $C_i \leq S_j$. A feasible schedule in which the precedence constraints are satisfied is denoted by ξ and can be defined by a set of m "sequencing" mappings $\xi \equiv \{\sigma_1, \ldots, \sigma_m\}$, where $\sigma_k(i)$ is the i^{th} job to be processed on k^{th} machine. If B_k is the set of jobs assigned to k^{th}

machine, the total weighted mean finishing time $wmft(\xi)$ of a schedule ξ is given by:

$$wmft(\xi) = \sum_{k=1}^{m} \sum_{i=1}^{|B_k|} w_{\sigma_k(i)} C_{\sigma_k(i)} \tag{1}$$

In the standard classification scheme of Graham et al. [10], the basic problem is denoted by $P_m|prec|\sum w_i C_i$.
In the probabilistic version of the problem, the number of tasks is a discrete random variable N varying in $[0, n]$. On a given instance of the problem, only a subset $I \subseteq L_n$ is present with a probability $P(I)$ of occurrence. A probabilistic modelization of the problem consists in attributing a probability of presence q_i for each task $T_i \in L_n$ independently from the others. In this paper, we focus on the particular case where every task has the same probability of presence q independently from the others. Then N is a binomial random variable with parameters (n, q). Under these assumptions, we propose in this paper to study the re-optimization and the a priori strategies to solve the probabilistic version of the problem $P_m|prec|\sum w_i C_i$

2.1 Reoptimization strategy

Reoptimization strategy consists in finding an optimum scheduling for every potential instance of the probabilistic problem. We are considering, for each reduced instance, a deterministic scheduling problem defined for the subset I of present tasks among the initial set L_n. The optimum value of $wmft$ for the subset I is denoted by $wmft(I, Opt)$. The $wmft$ for the probabilistic problem is a random variable which is denoted by $wmft(Opt)$ such that its expected value is:

$$\mathbb{E}\left[wmft(Opt)\right] = \sum_{I \subseteq L_n} P(I) wmft(I, Opt) \tag{2}$$

As a matter of fact, reoptimization strategy provides an optimal solution. However, even for easy particular cases of the problem, this strategy is impossible to carry out since the number of potential instances is exponential.

2.2 A priori strategy

In the a priori strategy, we give an "a priori solution" denoted by ξ which is a scheduling of the n tasks of the entire set L_n. For every potential subset I of tasks in L_n (i.e. $I \subseteq L_n$) we attribute a probability $P(I)$ of occurrence. The a priori strategy relies on a modification method denoted by $\mathcal{U}$ providing a solution for each subset I by updating "quickly" the a priori solution ξ. Hence, the $wmft$ is a random variable denoted by $wmft(\xi)$. Let $wmft(I, \xi_{\mathcal{U}})$ be the $wmft$ for a scheduling for a subset $I \subseteq L_n$, obtained from ξ by the modification method $\mathcal{U}$, then we have :

$$\mathbb{E}_{\mathcal{U}}\big[wmft(\xi)\big] = \sum_{I \subseteq L_n} P(I) wmft(I, \xi_{\mathcal{U}}). \tag{3}$$

Then, an optimal scheduling is ξ^* such as :

$$\xi^* = \arg\min_{\xi} \mathbb{E}_{\mathcal{U}}\left[wmft(\xi)\right] \tag{4}$$

We define the probabilistic weighted mean finishing time subject to precedence constraints problem which we denote by $P_m|prec, q_i|\mathbb{E}_{\mathcal{U}}\left[\sum w_i C_i\right]$, where $\mathbb{E}_{\mathcal{U}}\left[\sum w_i C_i\right]$ is the expected value of $wmft$. The choice of the modification method $\mathcal{U}$ is very important in the a priori strategy. Indeed, $\mathcal{U}$ must provide the solution in a reasonable time. $\mathcal{U}$ might be inspired from real-world situations and ideally it provides efficiently for every subset I schedules whose values are close to or at least within a constant factor $\epsilon > 1$

3 Reoptimization strategy and approximation algorithms

The deterministic $P_m|prec|\sum_i w_i C_i$ is NP-Hard for $m \geq 2$ and for an empty precedence graph [18]. For $m = 1$ and empty precedence graph, the problem can be efficiently solved in polynomial time algorithm relying on the Smith's rule [17] which consists in creating a priority list according to the non-increasing order of $\frac{w_i}{p_i}$. Generalizations of this algorithm, on single machine, to non-empty graph of precedence find optimal schedules for chains, trees and series-parallel precedence graph in pseudo polynomial time [3, 14, 11].
Even for the easy particular cases, the reoptimization strategy is impossible to carry out: Consider a problem with n tasks (L_n), the number of potential reduced instances (i.e. $I \subseteq L_n$) is 2^n and we have to solve optimally an exponential number of different problems. Then, we should better adopt a near optimal strategy which consists in approximating the reoptimization strategy by means of heuristics. Such approach is called redistribution strategy relying on an approximation algorithm $\mathcal{A}$. The expected value of $wmft$ is denoted by $\mathbb{E}\left[wmft(\mathcal{A})\right]$.

3.1 The Largest Ratio First algorithm (LRF)

The LRF algorithm relies on Smith's rule and finds an optimum schedule when the precedence graph is empty for a single machine problem. For $m > 1$ and/or non-empty precedence, the LRF algorithm has a performance guarantee within $\frac{1+\sqrt{2}}{2}$ (see [13]). Algorithm 1 describes the generalized LRF algorithm for $m \geq 1$ and an empty precedence graph:

Algorithm 1 The LRF algorithm for empty precedence graph.

1: **repeat**
2: Assign the m tasks of longest processing times to the m different machines.
3: Remove assigned tasks from L_n
4: **until** All tasks are assigned to different machines
5: For each machine process the tasks assigned in the Shortest Weighted Processing Time order.

However, for non-empty precedence graph, the LRF algorithm shows poor performance [3] even for the single machine case. We present in the following algorithms for the case of chains and tree classes of precedence.

3.2 The ρ-max algorithm

We consider, in this work, that the precedence graph is a set of r chains which implies that for every task there exists at most one immediate predecessor and at most one immediate successor. In the following, we denote by $\{H_i, 1 \leq i \leq r\}$ the set of chains describing the precedence graph. We denote by $i : j$ the task member of H_i and figuring in position j in the chain. We denote also by $Pred(i, j)$ the task (i, j) and all of its predecessors in H_i. Furthermore, we generalize Smith's rule for a set of tasks I by the ratio $\rho(I)$ such that:

$$\rho(I) = \frac{\sum_{T_i \in I} w_i}{\sum_{T_i \in I} p_i} \tag{5}$$

Algorithm 2 presents the variant of ρ-max algorithm for chains precedence graph.

Algorithm 2 The ρ-max algorithm for chains precedence.

1: Initialize an empty priority list β.
2: **repeat**
3: **for** every nonempty chain H_i **do**
4: Let $\delta_i = \max_j \rho\big(Pred(i, j)\big)$
5: **end for**
6: Let $\delta = \max_i \delta_i$ and let i^* and j^* be the largest values such that $\rho\big(Pred(i^*, j^*)\big) = \delta$
7: Add the tasks in $Pred(i^*, j^*)$ in order to the end of β, and remove these tasks from H_{i^*}
8: **until** the chains are all empty
9: Apply list scheduling, using β as the priority list.

In the case of single machine, the priority list β and the final schedule are equivalent and the cost is optimum. Besides, for $m > 1$ the algorithm does not guarantee optimum solution [8].

4 The natural a priori strategy

The a priori strategy relies on a modification method which updates an a priori solution for the entire set of tasks to get a feasible schedule for every potential instance I. A natural modification method which we denote by $\mathcal{U}_A$ consists first in deleting the absent tasks from the a priori schedule then to get a feasible schedule for the reduced instance, every task remaining in the schedule is shifted respecting the precedence constraints such that for every task in I figuring in the a priori schedule ξ, $\sigma_k(i)$:

$$S_{\sigma_k(i)} = \max\left(C_{\sigma_k(i-1)}, C_{Pred(\sigma_k(i))}\right), \tag{6}$$

where $Pred(\sigma_k(i))$ is the predecessor of the task $\sigma_k(i)$ and $\sigma_k(i-1)$ is the task previously processed on the machine k.
An explicit formula of the objective function can be found by considering the problem $\mathcal{P}$, defined as:

- $\forall i$, the weight of the task T_i is $W_i = q_i w_i$

- $\forall i$, the processing time of the task T_i is $P_i = q_i p_i$
- A task $\sigma_k(i)$ in a feasible schedule ξ starts at $S_{\sigma_k(i)}$ such that:

$$\max\left(q_{\sigma_k(j-1)}C_{\sigma_k(j-1)}, q_{Pred(\sigma_k(i))}C_{Pred(\sigma_k(i))}\right),$$

where $Pred(\sigma_k(i))$ is the predecessor of the task $\sigma_k(i)$.

The objective function of the problem $\mathcal{P}$ is :

$$\mathbb{E}_{\mathcal{U}_A}\left[wmft(\xi)\right] = \sum_{k=1}^{m}\sum_{i=1}^{|B_k|} w_{\sigma_k(i)}q_{\sigma_k(i)}\left(S_{\sigma_k(i)} + q_{\sigma_k(i)}p_{\sigma_k(i)}\right) \tag{7}$$

For the particular case of empty precedence graph, the objective function has a simpler expression :

$$\mathbb{E}_{\mathcal{U}_A}\left[wmft(\xi)\right] = \sum_{k=1}^{m}\sum_{i=1}^{B_k} w_{\sigma_{k(i)}}q_{\sigma_k(i)}\left[q_{\sigma_k(i)}p_{\sigma_k(i)} + \sum_{j<i} q_{\sigma_k(j)}p_{\sigma_k(j)}\right] \tag{8}$$

In the following we study the stability of probabilistic flowtime problem on a single machine subject to precedence constraints.

Lemma 1 *Let $\sigma = (\sigma(I), \sigma(\overline{I}))$ be a minimum cost schedule of n tasks on a single machine where $I \subseteq L_n$. $\sigma(I)$ (resp. $\sigma(\overline{I})$) is a minimum cost schedule for $\mathcal{G}(I)$ (resp. $\mathcal{G}(\overline{I})$).*

Proof
From the expression of weighted flowtime eq. (7), we have:

$$\begin{aligned}\mathbb{E}_{\mathcal{U}_s}\left[Fw(\sigma)\right] &= \sum_{i=1}^{n} w_{\sigma(i)}q_{\sigma(i)}\left(S_{\sigma(i)} + q_{\sigma(i)}p_{\sigma(i)}\right)\\ &= \sum_{i\in I} w_{\sigma(i)}q_{\sigma(i)}\left(S_{\sigma(i)} + q_{\sigma(i)}p_{\sigma(i)}\right) + \sum_{i\notin I} w_{\sigma(i)}q_{\sigma(i)}\left(S_{\sigma(i)} + q_{\sigma(i)}p_{\sigma(i)}\right)\end{aligned}$$

Then to minimize the total weighted flowtime, one should minimize the restricted flowtime of the subsets I and $\overline{I}$. ■

Lemma 2 *Let $I_1, I_2, ..., I_r$, $r > 1$ be a partition of L_n, then:*

$$\rho(L_n) = \sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)}\rho(I_i) \tag{9}$$

Proof

$$\rho(L_n) = \frac{w(L_n)}{p(L_n)} = \sum_{i=1}^{r}\frac{w(I_i)}{p(L_n)} = \sum_{i=1}^{r}\frac{w(I_i)p(I_i)}{p(L_n)p(I_i)} = \sum_{i=1}^{r}\frac{p(I_i)}{p(L_n)}\rho(I_i)$$

■

Lemma 3 *Let $I_1, I_2, ..., I_r$, $r > 1$ be a partition of L_n, then:*

$$\max\{\rho(I_i),\ 1 \leq i \leq n\} \geq \rho(L_n)$$

Moreover $\max\{\rho(I_i),\ 1 \leq i \leq n\} = \rho(L_n)$ *if and only if* $\rho(I_1) = \rho(I_2) = ... = \rho(I_r)$

Proof
This lemma is a consequence of lemma 2. $\rho(L_n)$ is a convex combination of $\rho(I_i),\ 1 \leq i \leq r$. Let $i^* = \arg_i \max \rho(I_i)$ then $\forall 1 \leq i \leq r$, we have $\rho(I_i) \leq \rho(I_{i^*})$. Hence:

$$\sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)} \rho(I_i) \leq \sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)} \rho(I_{i^*})$$

Besides:

$$\sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)} \rho(I_{i^*}) = \rho(I_{i^*})$$

and since $\sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)} = 1$, then:

$$\sum_{i=1}^{r} \frac{p(I_i)}{p(L_n)} \rho(I_i) = \rho(L_n).$$

Which concludes the proof. ■

Lemma 4 *Let $r > 1$ and $I_1, I_2, ..., I_r$ be a partition of L_n and let $\sigma = (\sigma(I_1), ..., \sigma(I_j), \sigma(I_{j+1}), ..., \sigma(I_r))$ and $\sigma' = (\sigma(I_1), ..., \sigma(I_{j+1}), \sigma(I_j), ..., \sigma(I_r))$ two feasible schedules, then we have:*

$$\mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma)\big] \leq \mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma')\big] \text{ if and only if } \rho(I_j) \geq \rho(I_{j+1})$$

Proof
We compute $\mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma)\big] - \mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma')\big]$. In both schedules σ and σ', the tasks of the subsets $I_i,\ i \neq j$ and $i \neq j+1$ appear in the same order and are not involved in the difference between the two expectancies. Then:

$$\begin{aligned}
\mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma)\big] - \mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma')\big] &= w(I_{j+1})p(I_j) - w(I_j)p(I_{j+1}) \\
&= \rho(I_{j+1})p(I_{j+1})p(I_j) - \rho(I_j)p(I_j)p(I_{j+1}) \\
&= \big[\rho(I_{j+1}) - \rho(I_j)\big]p(I_j)p(I_{j+1})
\end{aligned}$$

The sign of $\mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma)\big] - \mathbb{E}_{\mathcal{U}_S}\big[Fw(\sigma')\big]$ is the same of $\big[\rho(I_{j+1}) - \rho(I_j)\big]p(I_j)p(I_{j+1})$.
■

Theorem 5 *If I is an initial ρ maximum set, then there exists a minimum cost schedule σ such that $\sigma = (\sigma(I), \sigma(\bar{I}))$ and I is a continuous sequence in σ*

Proof
In the precedence graph $\mathcal{G}' = (L_n, U')$, $U' = U \backslash \{(i,j)$ such that $i \in I, j \in \overline{I}\}$. From a given schedule defined on $\mathcal{G}$, we build a feasible solution for $\mathcal{G}'$ such that the jobs of the subset I are scheduled first.

Assume that a minimum cost schedule defined over $\mathcal{G}'$ is $\sigma \equiv (\sigma(J_1), \sigma(I_1), \sigma(J_2), ..., \sigma(I_r), \sigma(J_r))$, with $r > 1$, $\cup_{i=1}^{r} I_i = I$, $\cup_{i=1}^{r} J_i = J = \overline{I}$ and $J_r \neq \emptyset \forall r$.
Let k be the smallest integer such that $\rho(I_k) \geq \rho(I_i)$, $\forall i \in [1, r]$. From lemma 3, we have $k > 1$. Indeed, if $k = 1$, I_1 would be ρ-maximum. However, I is ρ-maximum and $I_1 \subset I$. Moreover, we have : $\rho(J_k) \geq \rho(I_k)$ since σ is optimal for $\mathcal{G}'$. Hence:

$$\rho(I_{k-1}) \leq \rho(I_k) \leq \rho(J_k).$$

From lemma 4, if we swap J_k and I_{k-1} in σ, the flowtime improves. Hence σ is not optimal. Then, if σ is an optimal scheduling for $\mathcal{G}'$ then $\sigma = (\sigma(J_1), \sigma(I), \sigma(J_2))$.

Now, let us build an optimal schedule for $\mathcal{G}$ from an optimal schedule over $\mathcal{G}'$.
Let σ be a minimum-cost schedule defined over the precedence graph $\mathcal{G}'$ such that:

$$\sigma = (\sigma(J_1, \sigma(I), \sigma(J_2)), \text{ with } J_1 \neq \emptyset.$$

Let σ' be a schedule obtained from σ by swapping J_1 and I. σ' is a feasible solution for $\mathcal{G}$ and $\mathcal{G}'$. Assume that σ' is not optimal for $\mathcal{G}'$. From lemma 4, we have: $\rho(J_1) \geq \rho(I)$.
Besides: $J_1 \cup I$ is an initial set in $\mathcal{G}$ and

$$\rho(J_1 \cup I) = \frac{p(J_1)}{p(J_1 \cup I)}\rho(J_1) + \frac{p(I)}{p(J_1 \cup I)}\rho(I).$$

Furthermore, we have $\rho(J_1) > \rho(I)$. Then $\rho(J_1 \cup I) > \rho(I)$. However I is ρ-maximum, which contradicts the assumption.
Then, σ' is optimal for $\mathcal{G}$.
If I is an initial set and I is ρ-maximum in $\mathcal{G}$, then there exists an optimal solution for $\mathcal{G}$ such that $\sigma = (\sigma(I), \sigma(\overline{I}))$.

■

Theorem 5 depicts a particular case of stability of the probabilistic weighted flowtime problem. The restriction of an a priori solution for a given subset I of present tasks yields a minimum cost solution when I is a collection of initial and ρ-maximum sets. In this case, the overall sequencing remains unchanged if some tasks are deleted.

The following lemma states a general description of the stability of probabilistic weighted flowtime problem subject to precedence constraints.

Lemma 6 *Let σ be an optimal schedule for a set of n tasks L_n on a single machine subject to precedence constraints. There exists an initial ρ-maximum set I such that $\sigma = (\sigma(I), \sigma(\overline{I}))$.*

Proof
Assume that the first sequence of the tasks of a ρ-maximum set I is such that $\sigma = (\sigma(J_1), \sigma(I), \sigma(J_2))$ and that $J_1 \neq \emptyset$.
σ is optimal, then, from lemma 3, we have:

$\rho(J_1) \geq \rho(I)$

I is ρ-maximum, then:

$$\rho(J_1) \leq \rho(I)$$

Hence:

$$\rho(J_1) = \rho(I)$$

Then, J_1 is either ρ-maximum or J_1 contains a ρ-maximum set. Hence, the contradiction.

■

In the case of single machine problem, when the precedence graph is empty, the a priori strategy finds a schedule equivalent to the reoptimization strategy since an optimal schedule consists in a list which is not perturbed by the absence of some tasks. When the precedence graph is a set of r chains, the a priori strategy does not find exactly the same solution as reoptimization unless the absent tasks constitute a set of modules [16] which is a subset of sub-chains with maximum ρ factor.
An exact performance evaluation of the a priori strategy when the absence of tasks is arbitrary is difficult to perform. Then we propose in the following section to study experimentally the redistribution strategy according to an approximation algorithm $\mathcal{A}$ and the a priori strategy relying on the modification method $\mathcal{U}_A$.

5 Experimental Study

In this section we evaluate and compare experimentally the two strategies. An exact performance evaluation of these strategies appears to be difficult, then we propose an analysis through simulations. The experience consists in considering the values of m in $\{1, 2, 3, 4\}$ and generating for each value a set of n tasks where the processing times p_i are discrete samples from a uniform distribution between 1 and 50 and the weights w_i are discrete samples from a uniform distribution between 0 and 10 independent from processing times. Furthermore, for every task is attributed the same probability of presence q. Precedence constraints are randomly generated for each instance such that for chains precedence, we select a partition of the set L_n of n tasks. The random partition is detailed in [15].
To compute a mean value of the expectation of $wmft$ we generate, for each initial problem of size n, $k = 10000$ reduced problems where some tasks are absent and removed from the initial problem. Runs were conducted for n ranging between 100 and 1400 with step of 100 and for q in $\{0.1, 0.5, 0.9\}$.

5.1 Asymptotic behavior of the reoptimization strategy

First, we attempt to show experimentally that the redistribution strategy by an approximation algorithm $\mathcal{A}$ is asymptotic to the reoptimization strategy. Given the lower bound B [19, 9], we evaluate the redistribution strategy deviation from optimum scheduling. Let $R_{\mathcal{A}}^{Redist}$ denote the following ratio:

$$R_{\mathcal{A}}^{Redist} = \frac{\mathbb{E}\big[wmft(\mathcal{A})\big]}{B}, \tag{10}$$

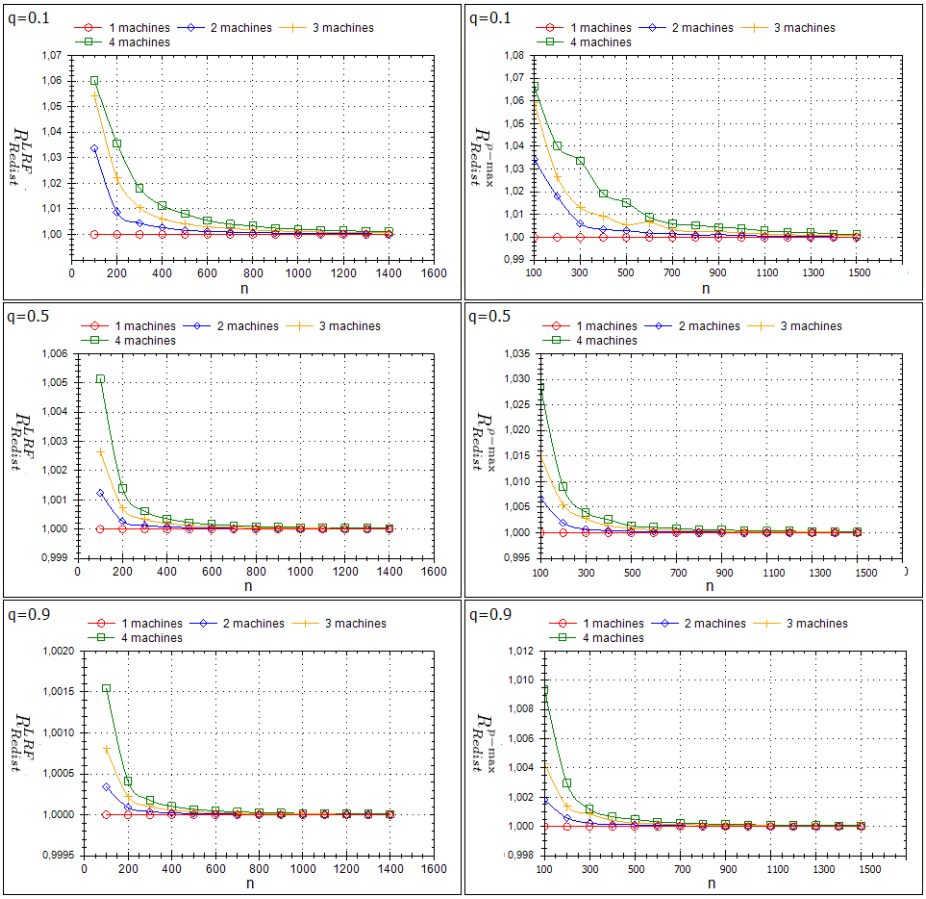


Fig. 1 Variations of the ratios R^{LRF}_{Redist} and $R^{\rho-\max \text{ Chains}}_{Redist}$.

where $\mathbb{E}\big[wmft(\mathcal{A})\big]$ is the expected $wmft$ for the present tasks of the set L_n by the approximation algorithm $\mathcal{A}$. Figure 1 shows the variation of $R^{Redist}_{\mathcal{A}}$.
We notice that the ratios R^{LRF}_{Redist} and $R^{\rho-\max \text{ Chains}}_{Redist}$ converge toward 1 from an $n_0 \approx 800$ for $q = 0.1$, $n_0 \approx 600$ for $q = 0.5$ and $n_0 \approx 400$ for $q = 0.9$.
The deviation is a decreasing function of q and of m and is at most 7% for $q < 0.5$ and 1% for $q > 0.5$.
This implies that the redistribution strategy by LRF heuristic for empty precedence graph and $\rho-\max$ for chains precedence is asymptotically equivalent to reoptimization strategy.

5.2 Asymptotic behavior of the a priori strategy

Now, we focus on the a priori strategy attempting to study its behavior. The problem consists in finding, under a priori strategy assumptions, a scheduling with optimum value according to the expression (7). In this experiment, we compute the following

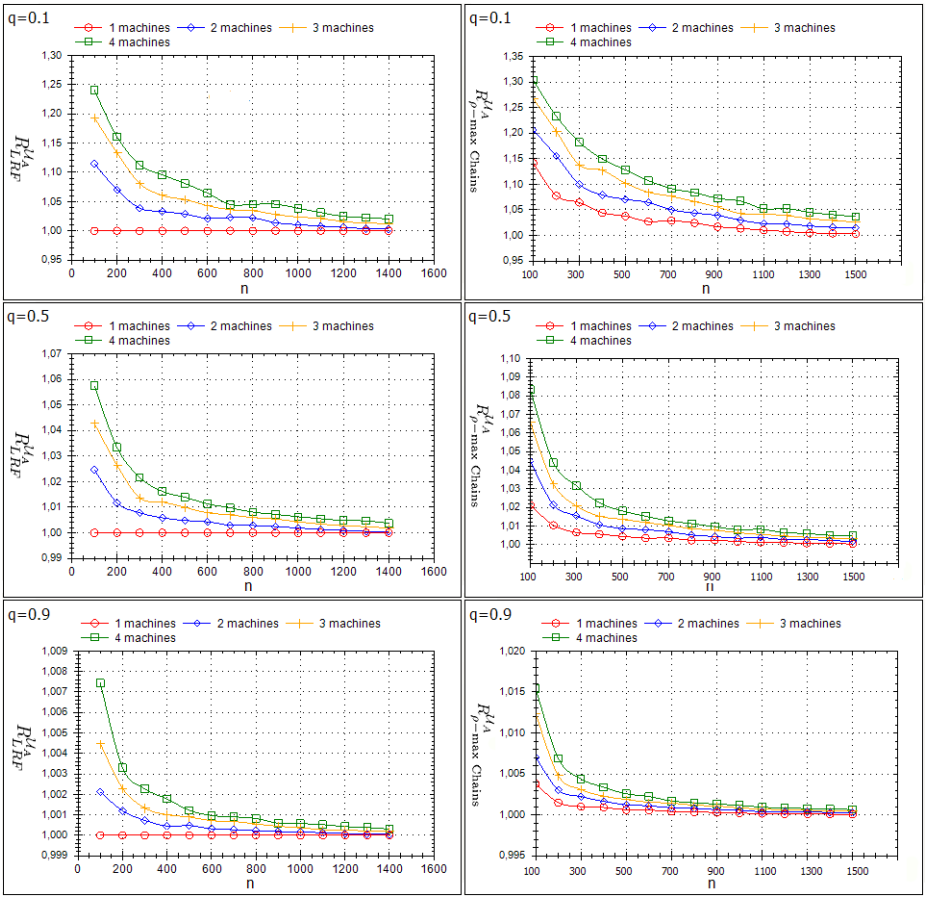


Fig. 2 Variations of the ratios $R^{\mathcal{U}_A}_{LRF}$ and $R^{\mathcal{U}_A}_{\rho-\max}$.

ratio for $m \in \{1, 2, 3, 4\}$:

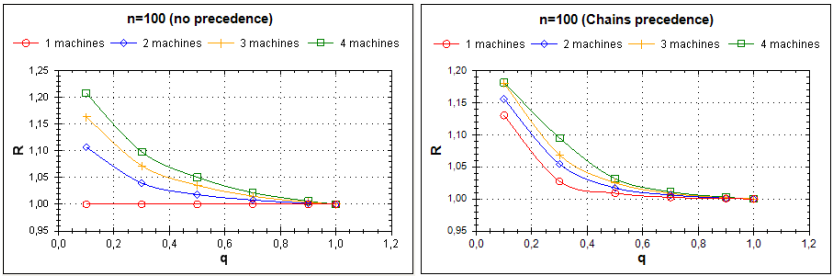
$$R^{\mathcal{U}_A}_{\mathcal{A}}(m) = \frac{\mathbb{E}_{\mathcal{U}_A}\left[wmft(\xi)\right]}{B}, \tag{11}$$

This ratio measures the deviation of a priori strategy from an optimal scheduling. Figure 2 shows the variation of the ratio for different problem sizes (n varying between 100 and 1400 with step of 100) and different values of q in $\{0.1, 0.5, 0.9\}$.
In these experiments, we notice that the deviation for non empty precedence graph are more important than the empty precedence graph case

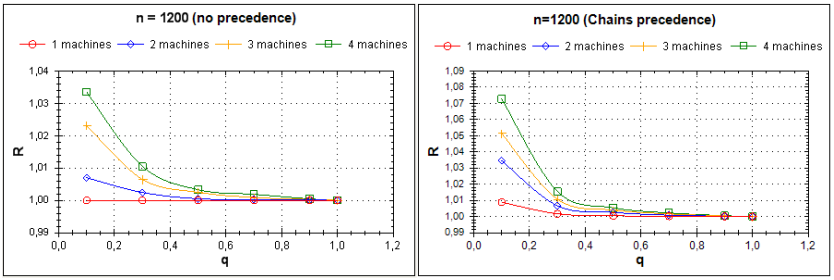
- The plots show that the value of the ratio $R^{\mathcal{U}}_{LRF}$ is a decreasing function of n and that the deviation increases as the number of machines increases. We notice also that the a priori strategy finds an optimum scheduling for $m = 1$ and that for $m \geq 2$, $R^{\mathcal{U}}_{LRF}(4) \geq R^{\mathcal{U}}_{LRF}(3) \geq R^{\mathcal{U}}_{LRF}(2)$. For $q = 0.1$ which implies a potential important perturbation of the initial problem, we have : $0 \leq R^{\mathcal{U}}_{LRF}(4) - 1 \leq 25\%$ and for $q = 0.9$ which can be interpreted as a tiny perturbation we have $0 \leq R^{\mathcal{U}}_{LRF}(4) - 1 \leq 0.7\%$. As for an average case, $(q = 0.5)$ we have $0 \leq R^{\mathcal{U}}_{LRF}(4) - 1 \leq 6\%$.

– In the case of chains precedence graph, we notice that the ratio $R^{\mathcal{U}}_{\rho-\max}$ decreases according to the size of the initial problem n and that it increases in function of m: $R^{\mathcal{U}}_{\rho-\max}(4) \geq R^{\mathcal{U}}_{\rho-\max}(3) \geq R^{\mathcal{U}}_{\rho-\max}(2)$. The ratio $R^{\mathcal{U}}_{\rho-\max}$ rises from 0.5% to 30% for $q < 0.5$ and from 0.1% to 6.4% for $q \geq 0.5$.

5.3 Pairwise comparison



(a) $n = 100$



(b) $n = 1200$

Fig. 3 Variation of the ratio R according to the probability q.

In this section we perform a pairwise comparison of the performances of reoptimization and a priori strategies. We compute the ratio:

$$R_m(q) = \frac{\mathbb{E}_{\mathcal{U}_A}\big[wmft(\xi)\big]}{\mathbb{E}\big[wmft(Opt)\big]}, \tag{12}$$

where $\mathbb{E}_{\mathcal{U}_A}\big[wmft(\xi)\big]$ (resp. $\mathbb{E}\big[wmft(Opt)\big]$) denotes the minimum expectation of $wmft$ for present tasks of L_n by a priori (resp. reoptimization) strategy . We compute this ratio for different values of q ranging in $\{0.1, 0.3, 0.5, 0.7, 0.9, 0.9999\}$ and for $n = 100$ and $n = 1200$ and $m \in \{1, 2, 3, 4\}$.
The variation of the ratio $R_m(q)$ according to q on Figure 3 shows that the ratio is a decreasing function of m. For empty precedence graph, the deviation is bounded by 0% and 20% for $n = 100$ and varies from 0% and 3.5% for $n = 1200$. For chains precedence, the ratio $R_m(q)$ rises from 0% to 7% for $n = 1200$ and ranges between 0% and 17% for $n = 100$. This leads to conclude that, asymptotically, the a priori and reoptimization strategies are close. The deviation is a decreasing function of q and the

strategies are completely equivalent independently from q for single machine case on empty precedence graph. This result is foreseeable since an optimal scheduling for the single machine problem is not altered by the absence of some tasks. This can be seen as a stability feature of the probabilistic problem. Besides, when the precedence graph is a set of chains, the strategies are equivalent only if the set of absent tasks is an initial ρ-max set.

6 Conclusion

In this paper, we have introduced the probabilistic problem $P_m|q_i, prec|\mathbb{E}\left[\sum w_i C_i\right]$ attempting to provide a realistic model for several real-world problems where we have to face the uncertainty of data. We have also defined two possible strategies to solve the problem and we have introduced an a priori strategy based on a natural modification method attempting to give a fast and robust solution. The a priori strategy is completely equivalent to the reoptimization for the particular case of single machine and empty precedence graph. In the case of non empty precedence graph and particularly when the precedence consists in a set of chains, the absence of some tasks from the initial set of data does not perturb the a priori schedule unless the absent tasks consist in a set of modules.
Through experiments on randomly generated instances, the a priori strategy exhibits encouraging results. However the deviation of the a priori strategy relying on $\mathcal{U}_A$ from reoptimization strategy shows that the divergence of the two strategies is due either to the modification method which is rigid and does not take into consideration the specificities of the problem definition or to the probabilistic formulation itself. Indeed, the probabilistic model proposed in this work is quite simple and the probabilities are associated independently with different tasks. As further works, we envisage to consider a more appropriate model which combines random aspect of data to the precedence constraints by associating probabilities to the modules instead of separated tasks.

References

1. Dimitris J. Bertsimas. Probabilistic combinatorial optimization problems. Technical report, Operations Research Center, MIT , Cambridge Mass, 1988.
2. Dimitris J. Bertsimas and L Howell. Further results on probabilistic travelling salesman problem. In *MIT Sloan School of Management Working*, pages 2066–2088, September 1988.
3. Ivan D. Baev, Waleed M. Meleis, and Alexandre Eichenberger. An experimental study of algorithms for weighted completion time scheduling. *Algorithmica*, 33:34–51, 2002.
4. Monia Bellalouna, Cecile Murat, and Vangelis Th. Paschos. Probabilistic combinatorial optimization problems on graphs: A new domain in operational research. *European Journal of Operational Research*, 87(3):693–706, December 1995.
5. Oded Berman and David Simchi-Levi. Finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers. *Transportation Science*, 22(2):148–154, 1988.
6. Z. Bouyahia, M. Bellalouna, and K. Ghédira. Load balancing a priori strategy for the probabilistic weighted flowtime problem. *Computers & Industrial Engineering*, 64(1):1 – 10, 2013.
7. Z. Bouyahia, M. Bellalouna, P. Jaillet, and K. Ghedira. A priori parallel machines scheduling. *Comput. Ind. Eng.*, 58(3):488–500, 2010.

8. J. Du, J.Y.-T. Leung, and G.H. Young. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Inform. and Comput.*, 92(2):219–236, 1991.
9. Eastman, Even, and Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 1964.
10. R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.*, 5:287–326, 1979.
11. W. A. Horn. Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal on Applied Mathematics*, 23(2):189–202, 1972.
12. Patrick Jaillet. Probabilistic traveling salesman problem. Technical report, Operations Research Center, MIT , Cambridge Mass, 1985.
13. Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an lrf schedule for the mean weighted flow-time problem. *SIAM J. Comput.*, 15(4):1119–1129, 1986.
14. E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math.*, 2:75–90, 1978.
15. A. Nijenhuis and H. Wilf. Combinatorial algorithms. *Academic Press*, 1975.
16. J. B. Sidney. Decomposition algorithms for single-machine sequencing with precedence rela- tions and deferral costs. *Operations Research*, 23:283–298, 1975.
17. W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
18. Martin Skutella and Gerhard J. Woeginger. A ptas for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research*, 25(1):63–75, 2000.
19. S. Webster. New bounds for the identical parallel processor weighted flow time problem. *Management Science*, 38:124–136, 1992.

MISTA 2015

An optimization-based approach for delivering radio-pharmaceuticals to medical imaging centers

Ioannis Akrotirianakis · Amit Chakraborty

Abstract It is widely recognized that early diagnosis of most types of cancers can increase the chances of full recovery or substantially prolong the life of patients. Positron Emission Tomography (PET) has become the standard way to diagnose many types of cancers by generating high quality images of the affected organs. In order to create an accurate image a small amount of a radio-active agent needs to be injected in the patient's body. These agents are produced in specially equipped pharmacies and then distributed to medical imaging centers which are located in metropolitan and rural areas. Due to the relatively fast decay process of the radio-activity levels it is very important that they arrive at the imaging centers well before the time that the patient enters the room where PET scanner is located. In this paper we discuss the distribution process of radio-pharmaceuticals and develop a flexible and efficient mathematical model. Our objective is to serve a number of customers within a pre-specified time interval at minimum transportation cost. At the same time the model ensures that all orders arrive at the imaging centers well before the patients enter the PET scanners. In addition the model takes into consideration the availability and capacity of the transportation vehicles. To demonstrate the effectiveness and efficiency of our optimization model we present preliminary computational results in a variety of test cases which show that it can achieve substantial savings in transportation costs.

1 Introduction

The transportation of products is a fundamental aspect in every efficient supply chain. With the fast development of economic globalization a variety of products routinely need to be transported to an ever increasing number of geographically dispersed customers. In recent years the distance among customers and production sites have increased dramatically, resulting in larger transportation costs. The most common transportation means are vehicles for ground transportation (e.g., pick up tracks and vans) which are mostly used for small quantities and/or light products. On the other hand, cargo planes are used for feaster long-distance deliveries and boats for larger quantities

I. Akrotirianakis, A. Chakraborty
Business Analytics & Monitoring, Siemens Corporate Technology, Princeton, NJ 08540, USA
E-mail: (ioannis.akrotirianakis , amit.chakraborty)@siemens.com

and heavier products. Each transportation means has a specific capacity and traveling speed. In most cases the products have to arrive at the customer locations within a pre-specified time interval which may make their transportation time-critical and complex.

A daily activity in today's pharmaceutical, medical, chemical and food industries involves the long-distance transportation of perishable products. The quality of such products may decay at rapid rates immediately after their production and during transportation. If a product does not meet certain quality criteria, the customer may discard it and refuse payment, in which case the manufacturing company may incur substantial loss of profit as well as customer dissatisfaction. In order to make sure their products arrive on time, in good quality and at minimum cost, companies have to carefully choose their transportation routes.

This paper develops an efficient and flexible optimization model that is able to find the most cost effective transportation routes of products manufactured in the radio-pharmaceutical industry. Nuclear medical imaging is routinely used for many diagnostic tests by physicians. In order to create an image for diagnostic purposes a small amount of a radio active agent is injected to the body of the patient and travels to the organ of interest. The emitted radiation is then detected and high accuracy images of the organ can be generated by Positron Emission Tomography (PET) scanners. Fludeoxyglucose (FDG) is the most commonly used radio-pharmaceutical PET. After FDG is injected into a patient's body, a PET scanner can form two or three dimensional images of the distribution of the FDG around the organ that needs to be examined. FDG has been used extensively for diagnosis, staging and monitoring treatment of cancers, particularly Hodgkin's disease, non-Hodgkin's lymphoma, colorectal cancer, breast cancer, melanoma and lung cancer. It has also been approved for use in diagnosis of Alzheimer's disease. When searching for tumors in the human body a dose of FDG is typically between 5 and 15 millicurie (denoted by mCi). The dose is injected rapidly into a saline drip running into a vein. The patient then waits for one hour for the sugar to distribute and be taken by organs that use glucose. To avoid consumption of the radioactive sugar by muscles (which use sugar) the patient must be in minimal physical activity. After one hour the patient is placed in a PET scanner for a series of scans, a process that may last from 20 minutes to one hour.

Sales of FDG have been growing since 2010 and are expected to exceed $880 million by 2017, while the market for PET radio-pharmaceuticals will increase to $3.5 billion by 2017 [3]. The production of FDG takes place in manufacturing facilities that contain special purpose equipments called cyclotrons [12]. Production is structured in batches and delivery takes place in doses. A dose contains the radio-active agent that will be injected to a patient before he/she enters the PET scanner. A fundamental activity in every manufacturing facility is the creation of a daily plan for the production of batches and distribution of doses to imaging centers and hospitals. An individual batch may provide sufficient product for up to forty or more individual doses.

A delivery schedule is a collection of doses that are assigned to transportation vehicles and routed to customer locations which are typically medical imaging centers or hospitals. Delivery schedules are site specific based on geography and local customer demand requirements. Typically, delivery schedules vary depending on the day of the week and/or seasonal ordering patterns. It is crucial that an order arrives at an imaging center at a certain time prior to the time that is going to be injected to the patient. If it arrives later, it may be discarded and the customer is not obliged to pay the cost of the doses. Other constraints associated with delivery schedules include the limited

number of delivery vehicles available at the manufacturing site and the capacity of each vehicle.

Every day, it is necessary to create delivery schedules. Current practice is the delivery schedules to be generated manually by experienced personnel in the radio-pharmacies and as a result they are not optimized. This may result in two major issues: (a) the delivery of some orders may arrive at a customer location later than the actual injection time specified in the contract, resulting in extra cost for the pharmaceutical company as it needs to replace the order free of charge in a later time or day, and (b) since the delivery routes are not optimal they may cost more to the pharmaceutical company simply because the vehicles travel longer distances. Apart from the increased monetary cost, longer delivery routes result in the release of more greenhouse gas emissions which negatively affect the climate and human health. The main contribution of our work is to address the above limitations by defining an efficient mathematical optimization model that determines the routes that are the most cost effective and guarantee that all doses arrive at a customer location before their injection times. To the best of our knowledge this is the first paper that deals with the distribution of radiopharamceuticals.

The paper is structured as follows. In section 2 we present the work that has been done in the area of transporting short-lived products that is relevant to our problem. In section 3 we discuss the details of the problem and define the notation we will use in the rest of the paper. In section 4 we present the mathematical optimization model and we discuss its functionality and in particular the purpose that each constraint serves. In section 5 we present a number of numerical results obtained by our model and the savings in transportation costs achieved. Finally, in section 6 we conclude the paper and give directions of future work.

2 Literature review

The problem we study in this paper falls in the general category of the vehicle routing problem (VRP) which is well studied in the area of Operations Research [10]. There are many papers dealing with several variations/extensions of the VRP, mainly because of its wide applicability in real-world applications. However, there is very limited research in the distribution of radio-pharmaceuticals and for this reason we focus on the application of VRP in the distribution of short-lived products which must arrive at the customer site between a specified time window. Emphasis is given in applications in the medical field. The original VRP formulation was introduced by Dantzig and Ramser [4] back in 1959. A detailed review of the classical VRP has been written by Toth and Vigo [20] and Parragh et al [17]. Laporte [13] provides an overview of major VRP definitions as well as efficient exact and approximate algorithms for solving it. Savelbergh [18] focused on the complexity of the VRP and proved that it is an NP-hard problem.

In the classical VRP we are given a set of trucks with a limited capacity and a set of customers each having a known demand of a product that is manufactured in a specific location. The distance and traveling times from the manufacturing site to each customer location are known. The aim is to determine the minimum cost or distance routes such that (a) every customer is served by only one vehicle, (b) all routes start and end at the depot and (c) the transportation vehicles do not carry weight more than their capacity. A very popular extension of the VRP is the case where the customers

request the delivery of their orders to arrive during a pre-specified time interval. This extension is known as the VRP with Time Windows (VRPTW) and it is often used in the transportation of perishable products. Many transportation problems in the radiopharmaceutical industry can be modeled as VRPTW.

In general, there two types of approaches used to solve VRPTWs: (i) exact (e.g., branch-and-cut, branch-and-price) and (ii) heuristics (e.g., tabu search, genetic algorithms). The literature is very large and we will only present a few papers that study the modeling and solution techniques for problems similar to ours. In [1] the authors describe an exact algorithm for solving the VRPTW where a single vehicle can participate in more than one routes. Their application area comes from the distribution of perishable goods where the routes are small and can be combined. In [2] an branch-and-cut algorithm is proposed in order to find the minimum number of vehicles required to visit a set of customers subject to time window constraints and capacity limitations. The authors introduce a wide variety of cuts and use then to tighten the relaxation of the MILP problem. Tarantilis and Kiranoudis [19] developed an efficient and robust meta-heuristic algorithm for solving the problem of distributing fresh milk using a heterogeneous fleet of vehicles. Hsu *et* al [11] proposed a model for the stochastic VRP with time windows and obtained optimal delivery routes, loads, fleet dispatching and departure times for delivering perishable products. Zanoni and Zavanella [21] developed an MILP model and a heuristic algorithm for solving the shipping of a set of perishable products from a single vendor to a common buyer with the objective of minimizing the sum of inventory and transportation costs. Osvald and Stirn [16] study the problem of distributing fresh produce and emphasize in the perishability of the transported products. They formulate the problem as VRP with time windows and time-dependent travel times. Their aim is to find transportation routes that minimize the distance and time traveled, the delay costs for servicing late a customer and the costs related with perishability. Doerner *et* al [6] developed a model and several heuristics for solving a novel type of a vehicle routing problem where time windows for the pickup of perishable goods depend on the dispatching policy used in the solution process. The application area is motivated by a project carried out with the Austrian Red Cross blood program to assist their logistics department. Dessouky *et* al [5] study the coordinated solution of the a facility location problem together with a VRP in order to ensure quick distribution of medical supplies in response to an emergency situation. Migahlaes and de Souza [15] present a model and an algorithm for solving VRP for the classical pharmaceutical industry, where the customer orders may change dynamically. More recently, Luo *et* al [14] propose a mathematical model for a VRP with stochastic demands and real-time vehicle control for distributing medical supplies in large-scale emergencies.

3 Problem description

It is common that pharmaceutical companies outsource the delivery of the doses to logistics companies, which are responsible for providing the transportation vehicles together with the drivers and the load and unload equipment. A driver may be allowed to work up to a certain number of hours, denoted by T (e.g., 8 hours), or drive a distance of a maximum number of miles per day (e.g., 250 miles). Also there is a maximum number of vehicles (denoted by N) that the logistics company makes available to the pharmaceutical company.

Mathematically the transportation of the doses to the medical imaging centers can be expressed as a general vehicle routing problem with time widows. Let $G = (V, E)$ be a complete undirected graph where $V = \{0, 1, 2, \ldots, n\}$ represents the nodes of the graph and $E = \{(i, j) : i, j \in V\}$ is the set of edges connecting the nodes. The set of nodes consists of the radio-pharmacy, denoted by 0, and the imaging centers. For convenience we will denote the set of the imaging centers as $V_c = \{1, 2, \ldots, n\}$. Also, throughout the paper we may refer to imaging centers as *customers*. Every imaging center places an order which consists of a number of doses. Let D_i represent the number of doses ordered by the i-th imaging center. In addition, the j-th dose ordered by the i-th imaging center has an injection time, T_{ij}^{INJ}, associated with it. All doses ordered by the a specific imaging center are delivered by the same vehicle. This means that the delivery vehicle must arrive at the imaging center before the earliest injection time, that is

$$T_i^{INJ} = \min\{T_{ij}^{INJ} : j = 1, \ldots, D_i\}, \forall i \in V_c \tag{1}$$

Furthermore, every imaging center may require the delivery to arrive during a certain time window, $[e_i, \ell_i]$, where e_i represents the earliest and ℓ_i the latest arrival times. The latest arrival time may be a certain number of minutes, p_i, prior to T_i^{INJ} defined by (1). Therefore the vehicle must arrive at the i-th imaging center no later than

$$\ell_i = T_i^{INJ} - p_i. \tag{2}$$

It is also possible that some imaging centers do not allow deliveries prior to a certain time. For example, an imaging center may not accept doses to be dropped off before the center opens for business. In this case e_i will be set to the opening time of the imaging center. In the case where doses can be dropped off any time in the day (even when the imaging center is closed) we set $e_i = 0$.

The distance and the time it takes to drive between node i and node j are denoted by d_{ij} and t_{ij}, respectively. We obtain distances and drive times by using the *geo-coding* services offered by Google [9]. The service can provide the distance and duration matrices of a network of any number of nodes. The distance and duration measures are not symmetric. This means that in general we have $d_{ij} \neq d_{ji}$ and $t_{ij} \neq t_{ji}$. The cost, c_{ij}, of driving from a location i to a location j is defined as

$$c_{ij} = (m + f)d_{ij} + g \tag{3}$$

where the m is the cost of traveling one mile, f is the fuel surcharge that the logistics company asks for every mile traveled, and g is a flat amount charged by the drivers for every customer site they visit (typical value ranges of m, f and g are \$1.1-\$1.5, \$0.055-\$0.065, and \$10-\$15). Also there is a fixed cost, F, associated with every vehicle used.

The fleet of vehicles is homogeneous, meaning that all vehicles have the same weight capacity, denoted by W_{VEH}. During transportation to customer locations the doses are placed and sealed in lead or tungsten containers in order to minimize the radiation exposure. The weight of each container is denoted by W_{CON} (usually a container may weigh 32.5 lbs). Hence the total weight of the order of the i-th customer is defined by $W_i = D_i W_{CON}$. We assume that one vehicle will deliver all doses ordered by a customer, that is, we do not consider split orders. This means that all orders weigh less than the vehicles' capacity, i.e., $W_i \leq W_{VEH}$. If the order of the i-th customer weighs more than the vehicle's capacity, then this customer can be split into the appropriate

number of *dummy* customers, $(i_1, \ldots, i_d)$ so that each of them is assigned orders whose total weight is less than the vehicle capacity.

Once a vehicle arrives at a customer location the drivers need to spend some time unloading the containers, signing certain documents and picking up empty boxes. This is called *service time* of customer i and is denoted by s_i. The service time may be fixed for all customers (e.g., 30 minutes) or may be a function of the number of doses ordered by the corresponding imaging center (e.g., $s_i = D_i s$, where s is the nominal time allocated for servicing one dose, typically 3 minutes).

Besides the parameters, described above, we need to introduce a number of variables whose optimal values will be determined by the solution of the mathematical model, described in the next section. More specifically, we use the binary variables y_{ij} to represent the order by which the network nodes are visited by the transportation vehicles, that is, $y_{ij} = 1$, if node i precedes node j, and $y_{ij} = 0$ otherwise.

Since every vehicle has a maximum weight capacity, we are also interested in the total weight carried by it during the complete route. Hence we define the variable w_i representing the total weight a vehicle has delivered until it has reached customer i. We also define the variable x_i representing the arrival time of a vehicle to customer site i (note that, in the implementation of the model, x_i is measured in minutes after midnight of the day of the delivery). Since the doses have to arrive at the customer before the dose with the earliest injection time we always have $x_i \leq \ell_i$, where ℓ_i is the latest time that a vehicle must arrive at a customer site and is defined in (2).

The objective of the pharmaceutical company is to determine delivery routes which will minimize the total transportation cost and guarantee that all orders reach the imaging centers before the specified injection time of each dose. In the remaining of this section we summarize the variables and parameters used in the mathematical description of the optimization model presented in the following section.

Parameters and sets

n:	number of medical imaging centers placing orders
V:	the set of all nodes in the network, $V = \{0, 1, \ldots, n\}$
V_c:	the set of all customer nodes in the network, $V_c = \{1, \ldots, n\}$
E:	the set of arcs in the network, $E = \{(i, j) : \forall i, j \in V\}$
T:	maximum time a driver is allowed to drive during a day
N:	maximum number of available vehicles
D_i:	the total number of doses ordered by imaging center i
F:	fixed cost for dispatching a vehicle
c_{ij}:	the cost of traveling from node i to node j
d_{ij}:	the distance of traveling from node i to node j
t_{ij}:	the time of traveling from node i to node j
p_i:	the time a vehicle must arrive at a customer prior to the injection time
s_i:	the service time needed by a driver to spend in an imaging center
W_{VEH}:	the vehicle weight capacity of each available vehicle
W_{CON}:	the weight of each container that seals a dose
W_i:	the weight of the total number of doses ordered by customer i
$[e_i, \ell_i]$:	the time window a dose must arrive in an imaging center

Variables

w_i: measures the weight a vehicle has delivered until it reaches customer i
x_i: the arrival time of a vehicle to customer site i
y_{ij}: 1, if node i is visited immediately before node j; 0 otherwise

4 Mathematical description of the optimization model

In this section we formally define the optimization model (equations (4) through (17)) and discuss in detail its major variables and constraints and the purpose they serve.

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} y_{ij} + F \sum_{j \in V_c} y_{0j} \tag{4}$$

$$\text{s.t.} \sum_{j=0, i \neq j}^{n} y_{ji} = 1, \ \forall i \in V_c \tag{5}$$

$$\sum_{j=0, i \neq j}^{n} y_{ij} = 1, \ \forall i \in V_c \tag{6}$$

$$\sum_{j \in V_c} y_{0j} \leq N \tag{7}$$

$$W_i \leq w_i \leq W_{VEH}, \ \forall i \in V_c \tag{8}$$

$$w_i \leq W_{VEH} + y_{0i}(W_i - W_{VEH}), \ \forall i \in V_c \tag{9}$$

$$w_j \geq w_i + W_j - W_{VEH} + y_{ij} W_{VEH} + y_{ji}(W_{VEH} - W_j - W_i), \ \forall i, j \in V_c, i \neq j, \tag{10}$$

$$e_i \leq x_i \leq \ell_i, \ \forall i \in V_c \tag{11}$$

$$x_i \geq e_i + y_{0i}(t_{0i} - e_i), \ \forall i \in V_c, \tag{12}$$

$$x_i \leq y_{i0}(T - t_{i0} - s_i - \ell_i) + \ell_i, \ \forall i \in V_c \tag{13}$$

$$x_j \geq x_i - \ell_i + y_{ij}(\ell_i + t_{ij} + s_i), \ \forall i, j \in V_c \tag{14}$$

$$x_i \geq 0, \ \forall i \in V_c \tag{15}$$

$$w_i \geq 0, \ \forall i \in \{1, 2, \ldots, N\} \tag{16}$$

$$y_{ij} \in \{0, 1\}, \ \forall i, j \in V \tag{17}$$

The objective function is defined in (4). In terms of the transportation costs we define it as the distance traveled by each vehicle multiplied by the cost per mile plus the fuel surcharge (see (3)). We have also added the fixed cost for using a vehicle.

To ensure the meaningfulness of our routes and avoid cycling, each customer should be visited once. After visiting that customer, we can only go for one customer next. Each truck can only deliver customers one by one and each customer can be only delivered once. These requirements are enforced by constraints (5) and (6). The maximum number of vehicles that are available every day in a radiopharmacy is enforced by constraint (7).

As stated by constraint (8) the total weight, w_i, delivered up to customer i should always be less than or equal to the capacity of the vehicle and greater than or equal to the weight of the order placed by customer i. Constraint (9) takes care of the case when the i-th imaging center is the first one to be visited by a vehicle. Indeed, when $y_{0i} = 1$ constraint (9) becomes $w_i \leq W_i$, which in conjunction with (8) gives us the stronger

constraint $w_i = W_i$. When the i-th imaging center is not the first to be visited, then $y_{1i} = 0$ and constraint (9) becomes $w_i \leq W_{VEH}$ which is redundant. By combining the two constraints (8) and (9) we are able to strengthen the feasible region of our problem resulting in a faster location of the integer optimal solution.

The case where imaging center i is not the first one to be visited deserves special attention. This case is taken care of by constraint (10). In this case the value of the variable w_i is equal to the weight of the orders of all the imaging centers that were visited between the pharmacy and the i-the center itself. For example, if center j is immediately after center i, then $y_{ij} = 1$ and $y_{ji} = 0$. As a result, constraint (10) becomes $w_j \geq w_i + W_j$ which means that the weight delivered to center j is at least equal to that delivered in center i plus the weight of the order of center i. If, on the other hand, center j is visited immediately before center i then we have $y_{ij} = 0$ and $y_{ji} = 1$, and constraint (10) becomes $w_j \geq w_i - W_i$. This constraint states that the weight delivered between the pharmacy and the j-th imaging center is not less than the weight delivered between the pharmacy and the i-th imaging center. In addition if center j is visited immediately before center i, we can deduce that $w_i \geq w_j + W_i$. Combining the last two inequalities we obtain the equation $w_i = w_j + W_i$. If centers i and j are not visited successively, then constraint (10) becomes $w_j \geq w_i + W_j - W_{VEH}$. By noting that the right hand side of the above constraint is always less than zero and by using the fact that $W_i \geq 0$ and the constraint (8), we can deduce that (10) becomes redundant.

In addition, a very important requirement in the delivery of radio-pharmaceuticals is the time window that a dose must arrive at an imaging center. We use the variable x_i in order to measure the time when a vehicle arrives at customer i. Constraint (11) defines the time window that a vehicle is allowed to arrive at customer i. The lower bound e_i defines the time after which the vehicle must arrive at the customer. If the driver of the vehicle can drop the orders any time at the customer or imaging center then $e_i = 0$, otherwise a value must be specified. The upper bound ℓ_i defines the latest time that the vehicle must arrive at the customer. Usually the customers and imaging centers request a dose to arrive a certain number of minutes before its injection time to the patient. Therefore the upper bound ℓ_i is initialized according to equation 2.

Constraints (13) and (14) define tighter upper and lower bounds on the arrival time at a customer location taking into consideration the customer sites that precede or follow site i. We analyze first constraint (13), which sets an explicit upper bound on the arrival time at the last customer site visited in a route. That is, if the i-th imaging center is the last one visited in a route then $y_{i0} = 1$ and (13) becomes $x_i \leq T - t_{i0} - s_i$. On the other hand, constraint (14) connects the arrival time between two consecutive locations. For example, if customer i precedes customer j, then $y_{ij} = 1$ and (14) becomes $x_j \geq x_i + t_{ij} + s_i$. Otherwise (i.e., when $y_{ij} = 0$), constraint (14) becomes $x_j \geq x_i - \ell_i$ which is redundant due to the constraint (11).

Finally, all the variables are continuous except y_{ij} which are binary. These requirements are described by constraints (15) to (17) in the optimization model.

5 Computational results

To illustrate the efficiency and practicality of the proposed mathematical model we initially present a case study describing the distribution of orders during a typical day in a radio-pharmacy. Due to confidentiality of company and patient data we do not

present the names or the locations of the medical imaging centers or hospitals that placed orders. We solve the model by using the FICO-Xpress optimization package, which includes a powerful modeling language (Mosel) [7] and efficient solvers [8] that can solve problems with integer and continuous variables as well as linear and nonlinear constraints.

The current practice in the radio-pharmacy of interest is to have employees (typically the pharmacists) to produce the distribution schedules and routes for the delivery vehicles. Although the pharmacists have great domain knowledge and experience, oftentimes they come up with sub-optimal delivery schedules, resulting in routes that cost more to the company and may not guarantee the on-time arrival at an imaging center. In addition, having pharmacists determining the delivery routes takes valuable time away from their main tasks and decreases their productive time by at least 30 minutes per day. We expect the optimization model we have developed to be a valuable decision support tool for every radio-pharmacist, since it will save them a lot of time and at the same time produce cost effective delivery routes saving a large amount of money to the pharmaceutical company.

The data for the model's parameters are obtained from two main sources. The first source is the pharmacy's Enterprise Resource Management (ERM) system, which stores information related to the doses ordered by the customers (e.g., the number of doses ordered by each imaging center, the injection times of the doses, the time windows when the orders must arrive at the customer, the addresses of the customers, etc). The second source is the Google Geocoding API [9] which can provide the distance and duration matrices of a set of customer locations provided that their addresses are available. Note that both matrices are not symmetric. We used the addresses of the radio-pharmacy and all the customers that have placed orders and created the corresponding distances, d_{ij}, and traveling times, t_{ij}.

For our case study, we selected a typical week day which consists of 16 imaging centers, denoted by C1 to C16. Those centers were requesting 67 doses in total. Table 1 presents more details about the orders placed by each imaging center. The doses were produced at the manufacturing site (radio-pharmacy) which is denoted by C0. The orders were ready for pickup by the drivers at 04:00 in the morning. All doses in an order have to arrive at the corresponding imaging center 30 minutes prior to the dose with the earliest injection time, described in the third column of Table 1. Orders can be delivered any time at all customer locations (even when they have not opened yet). This means that the early time is set to zero (i.e., e_i=00:00 or midnight). The service time at each customer location was set to 10 minutes. In the fourth column of Table 1 we have recorded the arrival time obtained by the optimal solution of our mathematical model. As can be seen all orders arrived well before the earliest injection time minus 30 minutes (the specified common early time).

The total transportation cost was \$1,258.36 and the total distance traveled by all drivers was 950.96 miles. The optimal routes are shown in Figure 1.A. On the left side of every connecting arc there are two values. The values in parentheses denote the drive time from one location to the next, whereas the other values denote the distance. As can be seen a total of 6 vehicles (equivalently, six drivers) were used to deliver the orders to all imaging centers.

In contrast, the routes determined by the pharmacist (actual routes) cost \$1,493.83 and the total number of miles covered was 1,154.83. Figure 1.B shows the graph of the actual routes and Table 2 compares the total cost and distance for the optimal and actual routes. In addition, it summarizes the improvements we get when the optimal

Customer ID	Doses ordered	Earliest dose injection time	Arrival time
C1	5	07:15	04:05
C2	4	08:00	04:32
C3	5	07:15	04:23
C4	3	08:45	06:39
C5	2	09:00	06:23
C6	6	08:45	06:33
C7	6	07:30	06:37
C8	2	09:00	07:15
C9	3	09:15	05:50
C10	4	09:30	06:27
C11	1	10:30	09:36
C12	3	08:45	04:37
C13	4	08:30	07:28
C14	13	12:30	04:38
C15	3	09:45	05:38
C16	3	08:00	04:02

Table 1 Details for the orders placed by imaging centers.

routes are used. As can be seen, there was a total of 15.76% reduction in transportation cost and 17.65% reduction in traveled distance. In addition the actual routes needed 8 vehicles, which represents an increase of two vehicles more than those needed by our model. This is quite important since it demonstrates better utilization of the vehicle capacity which is very useful when the logistics company does not have availability of the extra drivers or may charge more for offering additional vehicles.

	Delivery cost	Travel distance	Number of vehicles
Optimal routes	1,258.36	950.96	6
Actual routes	1,493.83	1,154.83	8
Improvements	15.76%	17.65%	33.33%

Table 2 Summary of the improvements in total cost, distance traveled and vehicles used between the optimal and actual routes.

Examining Figures 1.A and and 1.B closer we can see that our optimization model determined a much better way of delivering the orders to customers C10 and C11. More specifically, our model used one vehicle to travel to C10 and then to C11, whereas the pharmacist decided to use two vehicles to travel separately to C10 and C11. The total distance traveled by the vehicle of our model was 165+148=313 miles. On the other hand, the two vehicles sent by the pharmacist, traveled a total of 293+165=458 miles. It is these type of route consolidation that can provide substantial savings in traveling distance (in this case 458-313=145 miles), monetary cost and number of vehicles needed. An exactly similar situation arises with customers C9 and C13. Our model used one vehicle and traveled a total of 114+96=210 miles, whereas the pharmacist used two vehicles which traveled a total of 114+207=321 miles.

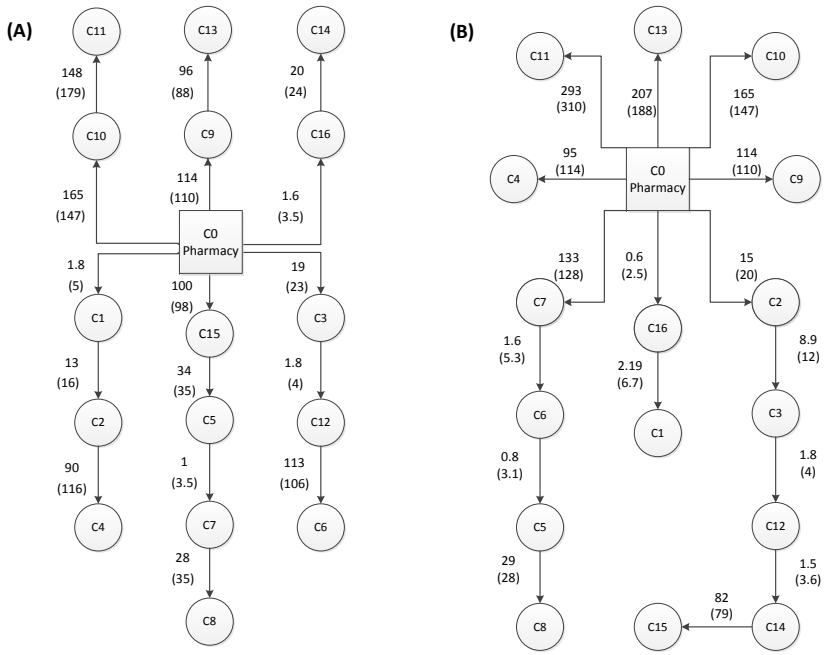


Fig. 1 The optimal routes are shown in graph (A) and the actual routes in graph (B). The distances are measured in miles and the drive times (in parentheses) are measured in minutes. The doses of all orders are available for pickup by drivers on 04:00. The actual routes are determined by experienced pharmacists or other personnel working in the radio-pharmacy.

We also used our model to compute optimal routes and compare them with those determined by pharmacists for a period of one week during the month of May 2014. That week contains some customers that are located far from the radio-pharmacy and some others that are located in close proximity. Also the total number of imaging centers that place orders in each day may vary. In Table 3 we include the number of customers (i.e., imaging centers) assigned to each batch for all the days. As can be seen the first batches in each day contain more customers. This is because the radiopharmacies try to satisfy as much demand as possible early in the day. The size of the MILP problems corresponding to batches assigned early in the day is larger and will therefore require more time to be solved than the later batches. Table 4 summarizes the improvements we get when we use the routes computed by our optimization model. It should be mentioned, however, that the size of all MILP problems is relatively small and the Xpress solver finds the optimal solution in few seconds. For this reason we do not report the CPU time needed to solve these problems. Our main focus is on the savings we obtain in the delivery cost and transportation distance compared to those obtained by pharmacists or other experiences personnel.

In all cases the optimization model produced routes that are more cost effective and need fewer vehicles than the actual ones. The highest improvements are obtained in batch 1 of day 1 where we have more than 20% improvements in both the cost and miles traveled. This is because the imaging centers fulfilled by that batch are far away

Day♯.Batch♯	Number of customers	Day♯.Batch♯	Number of customers
Day1.Batch1	16	Day3.Batch1	14
Day1.Batch2	13	Day4.Batch1	16
Day1.Batch3	10	Day4.Batch2	14
Day2.Batch1	14	Day5.Batch1	17
Day2.Batch2	13	Day5.Batch2	16
Day2.Batch3	11	Day5.Batch3	12

Table 3 Summary of the total number of imaging centers in each batch and day.

from each other and from the radio-pharmacy. Therefore, determining a cost effective set of routes for delivering the orders becomes difficult for human experts even when they have extensive experience in the subject.

On the other hand there are batches where the routes determined by the optimization model and the human expert do not differ much in terms of cost and distance traveled (see for example batch 3 in day 2 and batches 2 and 3 in day 1). This is also expected because most of the imaging centers in those batches are located close to each other and to the radio-pharmacy. As a result there is not much loss if a sub-optimal route is selected.

Table 5 summarizes the total improvements we obtain for all batches when we use our optimization model. We obtain savings in both the total delivery cost and traveling distance. The cost savings represent great news for the pharmaceutical company as they can invest them in other activities such as research and development of new drugs. The savings in the traveling distance has the important benefit of reducing the emissions released to the environment by the delivery vehicles and the positive impact to the quality of air and people's lives.

Finally we tested our optimization model on few days that contain a larger number of customers in order to see how computation time grows in terms of the size of the MILP problem. We could only obtained data for three days, which contained more than 25 customers placing orders. These represent large cases in the radiopharmaceutical industry. The details are shown in Table 6 and the results are summarized in Table 7. We can see that the optimization model is solved in relatively short time. We believe that the running time can be reduced further if valid inequalities and heuristics were introduced during the solution process. We plan to investigate this in a follow up paper. We should also mention that it was not possible to obtain the actual routes (i.e., the routes determined by the experienced personnel) and for this reason we do not report any improvements. We expect, however, the improvements to be larger than those reported in Table 2 as it is impossible for any human expert to determine the optimal routes as the number of customers becomes larger.

6 Conclusion

We have presented a new way of determining routes for delivering radio-pharmaceuticals to medical imaging centers that are geographically dispersed. The mixed integer optimization model we have developed can provide the most cost effective transportation routes and guarantees that all doses will reach the imaging centers before the time they

	DAY 1	Delivery cost	Distance traveled	Vehicles used
Batch 1	Optimal routes	1058.22	846.94	4
	Actual routes	1339.73	1090.67	6
	Improvements	21.01%	22.35%	
Batch 2	Optimal routes	517.19	343.89	5
	Actual routes	535.42	359.67	7
	Improvements	3.4%	4.39%	
Batch 3	Optimal routes	367.22	222.7	4
	Actual routes	380.03	233.79	5
	Improvements	3.37%	4.74%	
	DAY 2	Total cost	Total distance	Vehicles used
Batch 1	Optimal routes	1035.21	827.02	4
	Actual routes	1125.25	904.98	5
	Improvements	8.0%	8.61%	
Batch 2	Optimal routes	635.23	446.09	6
	Actual routes	693.93	496.92	7
	Improvements	8.46%	10.23%	
Batch 3	Optimal routes	238.41	154.47	3
	Actual routes	240.88	156.61	3
	Improvements	1.026%	1.366%	
	DAY 3	Total cost	Total distance	Vehicles used
Batch 1	Optimal routes	313.86	202.48	3
	Actual routes	337.84	223.26	4
	Improvements	7.11%	9.31%	
	DAY 4	Total cost	Total distance	Vehicles used
Batch 1	Optimal routes	1258.36	950.96	7
	Actual routes	1493.83	1154.83	8
	Improvements	15.76%	17.65%	
Batch 2	Optimal routes	539.94	380.9	3
	Actual routes	596.58	421.28	3
	Improvements	9.5%	9.58%	
	DAY 5	Total cost	Total distance	Vehicles used
Batch 1	Optimal routes	786.21	611.44	4
	Actual routes	807.65	630.0	5
	Improvements	2.65%	2.95%	
Batch 2	Optimal routes	398.52	267.11	5
	Actual routes	445.30	307.62	5
	Improvements	10.50%	13.17%	
Batch 3	Optimal routes	555.68	420.50	4
	Actual routes	582.17	443.44	5
	Improvements	4.55%	5.17%	

Table 4 Comparisons of optimal and actual routes and the improvements we obtain. The costs are in USD and the distances in miles.

	Total cost	Total distance	♯ of vehicles
Optimal routes	7,704.05	5674.50	52
Actual routes	8,578.61	6423.07	63
Actual − Optimal	874.56	748.57	11
Improvements	10.19%	11.65%	17.46%

Table 5 Summary of the total cost, distance and number of vehicles used for a week.

Day♯.Batch♯	Number of customers
Day6.Batch1	27
Day7.Batch1	32
Day8.Batch1	35

Table 6 Number of customers in larger batches.

	DAY 6	Delivery cost	Distance traveled	Vehicles used	CPU time (s)
Batch 1	Optimal routes	1876.32	1383.87	12	209
	DAY 7	Total cost	Total distance	Vehicles used	CPU time
Batch 1	Optimal routes	2338.51	1748.93	14	241
	DAY 8	Total cost	Total distance	Vehicles used	CPU time
Batch 1	Optimal routes	2524.89	1883.34	15	265

Table 7 Computational results for batches containing a larger number of customers. The CPU time is measured in seconds.

need to be injected to the patients. The optimization model has been applied on an illustrative example which demonstrates the monetary and mileage savings as well as the better utilization of the transportation vehicles. We have also tested the model on a typical week where each day has a different demand. From that computational study we were able to deduce that higher savings in delivery costs and distance are obtained when the number of the imaging centers increases and the distance between them and the production facility is large.

We plan to extend the model to cover the cases where (i) the fleet of transportation vehicles is not homogeneous, that is there are vehicles with different capacities and diving speed, (ii) determine which customers to serve first when there are not enough vehicles to fulfill all orders, and (iii) determine the best strategy by which a vehicle would collect empty containers (used in previous days) from the imaging centers it visits.

References

1. Azi N., Gendreau M. and Potvin J.-Y.: An exact algorithm for a vehicle routing problem with time windows and multiple use vehicles, European Journal of Operational Research, 202(3), 756-763 (2010)

2. Bard F. J., Kontoravdis G. and Yu G.: A branch-and-cut procedure for the vehicle routing problem with time windows, Transportation Science, 36(2), 250-269 (2002)
3. Burns, M.: Market For PET Radiopharmaceuticals and PET Imaging, Report 320, Bio-Tech Systems Inc., 4167 Pinecrest Circle West, Las Vegas, Nevada 89121 (2010)
4. Dantzig, G. B., Ramser, J. H.: The truck dispatching problem, Management Science, 6 (1), 80-91, (1959)
5. Dessouky, M. M., Ordez, F., Jia, H., and Shen, Z.: Rapid Distribution of Medical Supplies, In: Patient Flow: Reducing Delay in Healthcare Delivery , R. Hall (ed), Springer, (2006)
6. Doernera, K. F., Gronaltb, M., Hartl, R. F., Kiechlec, G., and Reimannd, M.: Exact and heuristic algorithms for the vehicle routing problem with multiple interdependent time windows, Computers and Operations Research, 35, 3034-3048 (2008)
7. FICO-Mosel.: Xpress-Mosel Reference Manual, Release 3.6, www.fico.com (2014)
8. FICO-Xpress.: Xpress Optimization Suite: getting started with Xpress, Release 7.3, www.fico.com (2012)
9. Google Geocoding API.: https://developers.google.com/maps/documentation/geocoding/
10. Hillier, F. S., and Liebernam, G. J.: Introduction to Operations Research, McGraw-Hill, New York, NY, (2009)
11. Hsu, C. I., Hung, S. F., Li, H. C.: Vehicle routing problem with time-windows for perishable food delivery, Journal of Food Engineering, 80 (2), 465-475 (2007)
12. Jacobson, M.S.: R. A. Steichen, and P. J. Peller, PET Radiochemistry and Radiopharmacy, *In: PET-CT and PET-MRI in Oncology, Peller et al. (Eds)*, Springer-Verlag Berlin Heidelberg (2012)
13. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms, European Journal of Operational Research, 59,pp.345-358, (1992).
14. Luo, J. Y., Wang, J. Y., and Yu, H.: A Dynamic Vehicle Routing Problem for Medical Supplies in Large-scale Emergencies, In Proceedings of the 6^{th} Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, August 20-22, 2011
15. Migahlaes, J. M., and Souza, J. P.: Dynamic VRP in pharmaceutical distributiona case study, Central European Journal of Operations Research, 14 (2), pp 177-192, (2006)
16. Osvald A., Stirn L.: A vehicle routing algorithm for the distribution of fresh vegetables and similar perishable food. Journal of Food Engineering 85 (2), 285295, (2008)
17. Parragh, S. N., Doerner, K. F., Hartl, R. F.: A survey on pickup and delivery problems, Journal fur Betriebswirtschaft, 58 (1), 21-51 (2008)
18. Savelsbergh, M.: Local search in routing problems with time windows. Annals of Operations Research 4(1):285305 (1985)
19. Tarantilis, C. D., Kiranoudis, C. T.: A meta-heuristic algorithm for the efficient distribution of perishable foods, Journal of Food Engineering, 50, 1-9, (2001)
20. Toth, P. and Vigo, D.: The vehicle routing problem, SIAM, Philadelphia (2002)
21. Zanoni, S., Zavanella, L.: Single-vendor single-buyer with integrated transport-inventory system: models and heuristics in the case of perishable goods, Computers and Industrial Engineering, 52 (1), 107-123 (2007)

MISTA 2015

Reliability Oriented DARP Models Involving Autonomous Vehicles

Victor PIMENTA • Alain QUILLIOT • Hélène TOUSSAINT • Daniele VIGO

Abstract We deal here with a static decisional model related to the monitoring of a DARP (Dial and Ride) system which involves, on a closed industrial site, small electrical autonomous vehicles. Because of technological issues, we focus on reliability, and propose a model which aims at assigning requests to vehicles in a way which minimizes the number of load/unload transactions. We first propose an ILP formulation of this Stop-Number model and compare the respective behaviors of several of its variants. Next we study it from a theoretical point of view and perform an experimental analysis of the behavior of a set covering oriented reformulation of the model. Finally we choose to handle our Stop-Number problem in a heuristic way, through a GRASP scheme which implements insertion mechanisms, well fitted to realistic dynamic contexts. Those methods are implemented and tested.

1. Introduction

Current trends in mobility management involve the emergence of flexible reactive systems, which meet mobility demands in a dynamic way while implementing some vehicle sharing. Those emerging systems strive to find their room between full individual mobility and traditional collective transportation systems. They also aim at bridging the gap between good and people mobility, and involve advanced technologies [16]: Internet, web services, mobile communications, remote tracking and monitoring… Depending on the context, they may work as closed systems, which are systems whose access is restricted to users who accept rules related to mobility tracking, pricing and responsibility, or open systems, which work on the basis of a free access/free market principle. Among such systems, one may mention Dial and Ride systems, Car-Sharing, Car-Pooling systems (AUTOLIB…), and Ride Sharing systems.

Recent advances in artificial perception and remote control make now arise new generations of autonomous (without any driver) individual or collective electrical vehicles: Cycab, VIPA (Individual Autonomous Vehicles of LIGIER S.A),…, which are involved into the design of new mobility services [10]. In case of VIPA electrical cars, experiments are currently undertaken on MICHELIN industrial site in CLERMONT-FERRAND, FRANCE, in order to assist internal mobility of both people and objects (internal mail, small packages…) inside the industrial site. Other applications are considered for the future, involving hospitals and some pedestrian downtown areas.

Such an experimental service works like a specific Dial and Ride (DARP) system. Users call from smart phones or from ad hoc communication devices and wait for the vehicles which

Victor Pimenta, Alain Quilliot, Hélène Toussaint
LIMOS CNRS 6158, Labex IMOBS3,Blaise Pascal University,
63000 CLERMONT-FERRAND, FRANCE
E-mail: {pimenta, toussain, quilliot}@isima.fr

Daniele Vigo
BOLOGNE University, ITALY
E-mail: daniele.vigo@unibo.it

are going to service them. But, since related moves are performed inside a closed restricted area (no more than a few square kilometers), user's demands must be handled in a very reactive way. Also, since autonomous vehicles are involved, there are very few routing options, and vehicles most often move along predetermined routes with fixed lengths. Thus, routing, as well as minimizing individual riding times and total riding times, are not anymore a key issue. What matters is reliability, related to the steady flow of the traffic induced by the involved vehicle fleet, and which can be improved through lean scheduling: scheduling has to smooth the trajectories of the vehicles and minimize complex interactions between the vehicles.

The model which we handle here, and which may be viewed as an extension of interval graph coloring models, is typical of this new kind of problems. It derives from a case study about the management of a specific Dial and Ride system, which involves VIPA automated vehicles and works in real time as a "horizontal elevator". Constraints are the classical DARP constraints, but performance criterion focuses on Stop Minimization: we do in such ways that, as often as possible, vehicles follow their way while avoiding any break (deviation toward a parking place, load/unload transactions…) in their trajectory. Though in practice, the related decision problem has to be handled on line, with performance evaluated through discrete event simulation, we set here, in order to get both benchmarks and a better understanding of the problem, a specific static (off line) ILP Stop-Number decision model.

So the paper is organized as follows: in section 2, we set our Stop-Number problem in a formal way, and provide it with an ILP formulation. Also, we discuss variants of this problem, and propose a set covering oriented reformulation of the Stop-Number model, which avoids considering the number of vehicles as a parameter of the model. Next, in section 3, we discuss some theoretical features of our model, and perform an experimental analysis of the behavior of its different ILP formulations. Finally, in section 4, we propose and test a GRASP heuristic algorithm (see [19]), based on insertion mechanisms, specially well-fitted to on line contexts.

2. A Static ILP model

Automated VIPA Vehicles [10], run here along a closed network which has the shape of a circuit Γ, while meeting *Dial and Ride* demands (see [1, 3, 4, 5]). It would be possible to propose other topologies for such a network, like tree or star topologies, but, in any case, a key point here is that there will be only one elementary path connecting some origin o to some destination d. So routing is not going to be here a part of the problem. The nodes of Γ are denoted by $\{0..n = 0\}$, and the vehicles always run in the same direction: in case a demand is about the transportation of some load L from an origin o to some destination d, the related trajectory is $\{o, o+1 \text{ Mod } n, o+2 \text{ Mod } n,.., d\}$. Circuit Γ is made of a *common track* and of load/unload areas, associated with nodes $\{0..n-1\}$ according to figure 1 below:

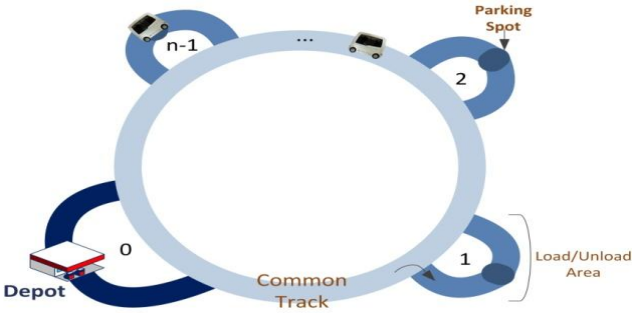


Figure 1: **VIPA Track**

Node 0 is a *Depot* node, where vehicles may charge batteries, and the speed of the vehicles on the main track is constant (about 15 km/h): thus overtaking is forbidden on the main track. When a vehicle gets out some *load/unload* area, it has no priority on the other vehicles. Vehicles meet users on *load/unload* areas. The profile of the speed of a vehicle in the neighborhood of a load/unload area comes as in figure 2:

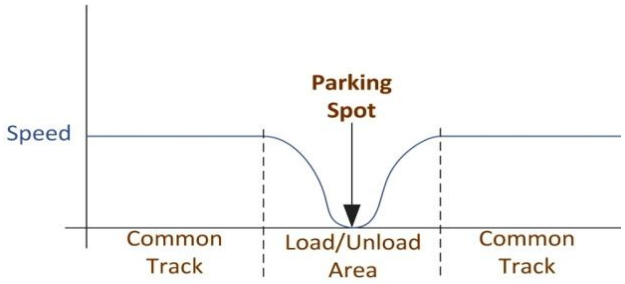


Figure 2: ***VIPA Speed Profile***

Running along the whole circuit without stopping anywhere only takes a few minutes. So, an important feature of the problem is that a vehicle may run several times around Γ (*waiting laps*) before effectively servicing a user. Still we forbid such a user to stay a full tour inside the vehicle. It comes that managing the vehicle fleet means, for every demand *j*, accepting it or rejecting it, and, in case it is accepted, assigning it both a vehicle *k* and a *waiting lap h*, that means the number of times the vehicle is going to run along Γ before servicing this demand.

We adopt here a *static* point of view, and suppose that we are provided with a fixed number *K* of identical vehicles, all located at time 0 in the *Depot* node of Γ, all those vehicles being endowed with a same capacity *CAP*, which represents a weight or a number of passengers. Thus, a *demand* (or request) $j = (o(j), d(j), L(j))$ is defined by an *origin* $o(j)$ and a *destination* $d(j)$, both in $\{0..n\text{-}1\}$, together with a *load* $L(j)$. We first suppose here that users ask for the system only when they are ready to move, and, so, that a demand does not involve any kind of temporal requirement (time-window…). We denote by *H* the largest number of tours a vehicle is allowed to perform before starting servicing a given demand (*waiting lap*). In order to avoid dealing with *Rejected Demand* minimization, we suppose that the *Vehicle Number K* is large enough to meet all demands according to the *waiting lap* restriction induced by *H*.

Since **reliability**, which is essentially correlated to *load/unload* transactions, is at stake,we assign vehicles and *waiting laps* to users in such a way that vehicles minimize their *Stop Number*, that means stay as often as possible on the main track without deviating onto the *load/unload* areas. This defines the *Stop-Number Minimization* Problem. If we refer to the standard *Dial and Ride* criteria (see [3, 5, 7]), we see that individual *riding times* are not part of our problem. Also, we intuitively feel that, because of Figure 2, minimizing the *Stop Number* will also tend to minimize the *Global Riding Time* of the fleet as well as the *Vehicle Number*, which means the number of vehicles which are needed in order to meet the demands.

2.1 The ILP Stop Number Minimization Model

In order to cast our problem into a formal framework, we unfold the circuit Γ as a linear ordered set $I(\Gamma) = \{0..(H + 2).n\}$, which we call the *Stop Node Set*.

Let D be the set of all demands $j = (o(j), d(j), L(j))$, $j = 1..m$. In case $d(j) < o(j)$, we replace $d(j)$ by $d(j) + n$. So we associate, with every demand $j = 1..m$, a collection of $H + 1$ discrete intervals $I_{j,h} = \{o(j) + h.n, \ldots, d(j) + h.n\}$, $h = 0..H$, of the *Stop Node Set*.

Clearly, we may suppose that every node $i = 1..n\text{-}1$ is *active* for our problem, that means such that there exists at least one index value j such that $i = o(j)$ or $i = d(j)$. If it is not the case, we remove all *non-active* nodes from the set $\{1..n\text{-}1\}$.

For any node i in the *Stop Node Set*, $i \neq (H+2).n$, we set:

- $Cross(i) = \{(j,h), j = 1..m, h = 0..H$, such that $o(j) + h.n \leq i < i+1 \leq d(j) + h.n\}$: so, $j \in Cross(i)$ means that if a vehicle services j after running h times around Γ, (that means according to *waiting lap h*), then it must be carrying load $L(j)$ when moving from node i to its successor $i+1$;
- $Stop(i) = \{(j, h), j = 1..m, h = 0..H$, such that $(d(j)+ h.n = i)$ OR $(o(j)+ h.n = i)\}$: so, $j \in Stop(i)$ means that if a vehicle services j according to *waiting lap h*, then it stops at i.

Then we get the following simple ILP model for the *Stop Number Minimization* problem, in case the number K of available vehicles is fixed:

Stop-Number(*K*) ILP Model:

{***Unknown vectors***:

- $Z = (Z_{j,k,h}, j = 1..m, k = 1..K, h = 0..H)$ with {0, 1} values : $Z_{j,k,h} = 1$ if vehicle k services demand j according to *waiting lap h*;
- $T = (T_{k,i}, k = 1..K, i = 1..n.(H+2)\text{-}1$, with {0, 1} values : $T_{k,i} = 1$ if i is a *stop node* for vehicle k.

Constraints:

- For any $j = 1..m$, $\Sigma_{k,h} Z_{j,k,h} = 1$; (**Demand j is serviced once**)
- For k, i, $\Sigma_{(j,h) \in Cross(i)} L(j).\ Z_{j,k,h} \leq CAP$, (**Capacity Constraints**)
- For any k, i , any $(j, h) \in Stop(i)$: $Z_{j,k,h} \leq T_{k,i}$. (**Coupling Constraints**).

Minimize : $\Sigma_{k,\,i}\ T_{k,i}\}$

Let us denote by *V-Stop-Number*(*K*) the optimal value of the *Stop-Number*(*K*) program. Then the *Stop Number Minimization* problem comes as follows: {Compute $K = K_{Sto}$ such that *V-Stop-Number*(*K*) be the smallest possible}. In case all loads $L(j)$, $j = 1..m$, are equal to 1, we talk about the *Unit Stop-Number* Problem.

2.2 Extensions and Variants

Time Windows: Introducing *time windows* in the *Stop-Number* model means imposing lower and upper bounds $Min(j)$, $Max(j)$, $j = 1..m$ on the *waiting laps* $h(j)$, $j = 1..m$: which means setting: $Z_{j,k,h} = 0$ for any $h \notin \{Min(j), .., Max(j)\}$ and for any $k = 1..K$.

Let us denote now by *Load-Vehicle*(*K*) the feasibility problem which derives from restricting the *Stop-Number*(*K*) ILP Model to vector *Z*.

Feasibility *Load-Vehicle*(*K*) ILP Model:

{*Compute* $Z = (Z_{j,k,h}, j = 1..m, k = 1..K, h = 0..H)$ with {0, 1} values such that:

- For any $j = 1..m$, $\Sigma_{k,h} Z_{j,k,h} = 1$; (**Demand j is serviced once**)
- For any vehicle k, any stop node $i = 0..(H+2).n - 1$,
  $\Sigma_{(j,h) \in Cross(i)} L(j).\ Z_{j,k,h} \leq CAP$; (**Capacity Constraints**)}

This allows us to set the following variants of the *Stop-Number* problem:

- ***Vehicle Number* Minimization**: {Compute the smallest value $K = K_{Veh}$ such that *Load-Vehicle*(*K*) has a feasible solution}

- ***Global Riding Time* Minimization**: {Compute $K = K_{Rid}$ such that *Load*-Vehicle(*K*) has a feasible solution and such that the sum $2K$ + (Optimal value of the following program *Global-Riding*(*K*)) is the smallest possible}:

***Global-Riding*(*K*) ILP Model**:
{*Compute* $Z = (Z_{j,k,h}, j = 1..m, k = 1..K, h = 0..H)$, with {0, 1} values, and $Ride_k$, $k = 1..K$, with integral values, such that:
- For any $j = 1..m$, $\Sigma_{k,h} Z_{j,k,h} = 1$; (**Demand j is serviced once**)
- For any k, any i, $\Sigma_{(j,h) \in Cross(i)} L(j).\ Z_{j,k,h} \leq CAP$; (**Capacity Constraints**)
- For any k, and any h,j, $h.Z_{j,k,h} \leq Ride$k;
- $\Sigma_k Ride_k$ is the smallest possible}

2.3 A Set Covering Oriented Generic Reformulation.

As it also happens for the graph coloring problem (see [6, 8, 12]), our previous formulation of the *Stop-Number* problem is not a true ILP one, since it does not manage parameter *K* according to the ILP formalism. Modifying it in such a way it induces a true ILP model increases its size and leads to a rather clumsy formulation, which makes appear a large number of useless vehicles. But it is possible, following [1, 9], to reformulate the *Stop-Number* model as a *set covering* oriented ILP model. In order to do it, we define the notion of *feasible service*

- A *feasible service* is any pair $s = (J, h)$, where J is some subset of the set $\{1..m\}$, and h some function from $\{1..m\}$ to $\{0..H\}$, such that, for any stop node $i = 0..(H+2).n - 1$:
  - $\Sigma_{j \in J \text{ such that } (j,h(j)) \in Cross(i)} L(j) \leq CAP$.

Clearly, a feasible service identifies a set J of demands which might be serviced by a same vehicle together with, for any such a demand $j \in J$, its related *waiting lap* $h(j)$. According to this, we use the notation $j \in s$ in order to say that demand j is met by service s.

Then we denote by S the set of all feasible services. For any $s = (J, h) \in S$, we set: *Stop-Node*(s) = $\{i \in 1..(2+H).n - 1$, such that $Stop(i) \cap J$ is not empty$\}$.We denote *N-Stop*(s) the cardinality of *Stop-Node*(s). Doing it leads to following *Stop-Number* reformulation:

***Stop-Number Set Covering Oriented* Reformulation**:
{*Unknown vector*: $(X_s, s \in S)$ with {0, 1} values: $X_s = 1$ if some vehicle performs the *feasible service s*.
Constraints : For any $j \in 1..m$, $\Sigma_{s \text{ such that } j \in s} X_s = 1$; (**Every demand is serviced once**)
Minimize : Σ_s *N-Stop*(s).X_s}

We notice that this *Set Covering oriented* reformulation does not involve the parameter *K*.

3. A First Theoretical and Experimental Analysis

Before proposing and testing algorithms, we first try to identify the features which make it be a really difficult problem. Also, we compare to each other in an experimental way the different variants which were introduced in previous Section 2.

3.1 Linking the *Stop-Number* Problem with Interval Graph Coloring: Discussing Complexity.

The first question which arises in a natural way is about worst case complexity. We may state:

Theorem 1: *In the case when H = 0, and no hypothesis is done about CAP and the loads L(j), j = 1..m, Stop-Number is NP-Hard*.

Proof: We may consider a collection of demands $j = 1..m$ such that:

- $o(j) = 1$ and $d(j) = n-1$ for every $j = 1..m$; $2.CAP = \Sigma_j L(j)$.

Then the optimal value of the related *Stop-Number* instance is equal to 4 iff the *2-Partition* instance which is defined by *CAP* and the loads $L(j)$ has a solution. *End-Proof*.

In case $L(j)$ are all equal to 1 (*Unit Stop-Number Problem*), we can only conjecture:

Conjecture: *If H = 0, the Unit Stop-Number Problem is NP-Hard*.

Still, in case $H = 0$, the *Unit Stop-Number* Problem is close to time-polynomiality. If $H = 0$, then the matrix associated with the *Load-Vehicle*(K) feasibility model is totally unimodular, since, for every $k = 1..K$, the matrices M_k defined by the constraints:

- For any stop node i, $\Sigma_{(j,h) \in Cross(i)} L(j).\ Z_{j,k,h} \leq CAP$, (**Capacity Constraints**)

are a same interval matrix. As a matter of fact, if $H = 0$, the *Unit Vehicle-Number* problem can be solved through a *Min Flow* algorithm (see for instance [8]).

Also, in the case when K and CAP are fixed and $H = 0$, it happens that the *Stop-Number* problem is time-polynomial, even if no restriction is imposed on the load values $L(j)$, $j = 1..m$.

Theorem 2: *If H = 0 and if CAP, K are fixed, then the Stop-Number Problem can be solved in polynomial time*.

Proof: We only have to check that, in such a case, the *Stop-Number* problem may be solved through dynamic programming, while involving a *state space* of polynomial size. We scan the set *TS* of pairs (i, j), $i = 1..n$, $j = 1..m$, linearly ordered as follows: $(i, j) << (i', j')$ iff $(i < i')$ or $((i = i')$ and $(j < j'))$. This set plays the role of the *time space*. We consider that the current state of the process at such an *time* (i_0, j_0) in *TS* is defined as the current load functions $LOAD_k$, $k = 1..K$, which to any load $1 \leq L \leq CAP$ makes correspond the largest node i_1, $i_0 \leq i_1 < i_0 + n$, such that the load of k is already scheduled to be larger or equal to L, based upon the subset $\{j$ such that $(o(j) < i_0)$ or $((o(j) = i_0)$ and $(j < j_0))\}$ of the set $\{1..m\}$ induced by all demands which have already been assigned to vehicles $k = 1..K$. Then the decision consists in assigning the smallest $j_1 \geq j_0$ such that $o(j_1) = i_0$ to some vehicle k, in such a way the capacity constraint is not broken and some Bellman equations be satisfied. *End-Proof*.

3.2 Evaluating the Practical Difficulty of Stop-Number: the Lower Bound Issue

In order to get an idea of the practical difficulty of the *Stop-Number* problem, we run the *Stop-Number*(K) ILP program with the CPLEX library, while observing the way running times vary as a function of n, m and K, together with the gap induced by relaxation of the integrality constraints on both vector Z and T. Since no test-bed exists for the *Stop-Number* problem, we randomly generate instances *Gr_n-m-H-L-CAP*, whose main characteristics are:

- $n =$ number of active stop nodes of the circuit; $m =$ number of demands;
- $H =$ maximal *waiting lap*; $L =$ mean load value; $CAP =$ common vehicle capacity.

For every instance, we apply the CPLEX12 library on both the *Stop-Number*(K) program and its rational relaxation. We denote by:

- *OPT*: optimal value of *Stop-Number*(*K*); *OPT**: optimal value of therelated rational relaxation; *GAP* = (*OPT* – *OPT**)/*OPT*; K_{Sto} = the related vehicle number;
- K_{Veh} = optimal value of the *Vehicle-Number* model;
- *CPU*: CPU running time related to the resolution of *Stop-Number*(K_{Veh}), in seconds.

Tests are performed on a serveur linux CentOS release 6.6, processeur Intel(R) Xeon(R) CPU E7- 8870 @ 2.40GHz, with the help of the CPLEX12.5 LP Library, and we get:

Instance	K_{Veh}	K_{Sto}	*OPT*	*OPT**	*GAP (%)*	*CPU*
Gr_10-28-0-5-10	6	6	29	15	48.3	1.6
Gr_10-28-1-5-10	3	4	24	12.5	47.9	25.6
Gr_10-28-2-5-10	2	3	22	11.6	47.0	47.5
Gr_10-28-3-5-10	2	2	22	11.25	48.9	30.3
Gr_10-28-4-5-10	2	2	22	11	50.0	433.4
Gr_10-33-0-5-10	10	10	37	18	51.3	13.4
Gr_10-33-1-5-10	5	5	31	14	54.8	604.8
Gr_10-33-2-5-10	4	4	29	12.6	56.3	1636.5
Gr_10-33-3-5-10	3	3	29	12	58.6	4933.9
Gr_10-33-4-5-10	2	2	29	11.6	60.0	1512.0
Gr_10-38-0-5-10	8	8	33	16	51.5	122.7
Gr_10-38-1-5-10	4	4	29	13	55.1	617.0
Gr_10-38-2-5-10	3	3	28	12	57.1	3926.8
Gr_10-38-3-5-10	2	3	27	11.5	56.9	2473.5
Gr_10-38-4-5-10	2	2	27	11.1	58.8	5540.4
Gr_15-42-0-5-10	11	12	50	23	54.0	11244.7

Table 1: ***No Hypothesis about the loads L(j), j = 1..m.***

Instance	K_{Sto}	*OPT*	*CPU*	*OPT**	*GAP*
Gr_10-28-0-1-5	6	23	3.1	15.00	34.78
Gr_10-28-1-1-5	3	20	16.9	12.50	37.50
Gr_10-28-2-1-5	2	19	31.3	11.67	38.60
Gr_10-28-3-1-5	2	18	37.7	11.25	37.50
Gr_10-28-4-1-5	2	18	200.6	11.00	38.89
Gr_10-33-0-1-5	10	26	17.3	18.00	30.77
Gr_10-33-1-1-5	5	22	239.6	14.00	36.36
Gr_10-33-2-1-5	4	21	830.2	12.67	39.68
Gr_10-33-3-1-5	3	20	1255.8	12.00	40.00
Gr_10-33-4-1-5	2	20	1227.2	11.60	42.00
Gr_10-38-0-1-5	8	25	61.7	16.00	36.00
Gr_10-38-1-1-5	4	22	665.1	13.00	40.91
Gr_10-38-2-1-5	3	21	1938.4	12.00	42.86
Gr_10-38-3-1-5	2	21	2390.1	11.50	45.24
Gr_10-38-4-1-5	2	21	11278.1	11.13	47.02
Gr_15-42-0-1-5	11	37	22493.8	23.00	37.84

Table 1.bis: ***The Unit Case.***

Analysis:

1. Not surprisingly, we fail solving the general *Stop-Number*(*K*) model, as soon as *n* and *m* become larger than respectively 10 and 30. What can be seen here is that the *GAP* value is very large: in most cases, the lower bound which is obtained through relaxation of the integrality constraint of *Stop-Number*(*K*) is less than 50% of the optimal value *OPT*. This can be easily explained because of the *coupling* constraint: relaxing the integrality constraint on *Z* allows to split demands $j = 1..m$ into several small fractional pieces, and distribute them among the vehicles $k = 1..K$ and the waiting laps 0..*H*. It follows that related values $T_{i,k}$ may turn out to be very close to 0.

2. The situation is improved when we limit ourselves to the case $H = 0$ and so keep demands $j = 1..m$ from being distributed along the temporal dimension.

3. As could be expected from previous subsection ***3.1***, the case when H = 0 and loads L(j) are equal to 1 can be handled in a more efficient way. In such a case, we also see that relaxing the integrality constraint of *Stop-Number* yields lower bounds with a GAP value around 35% , that means with better quality.

The Lower Bound Issue: **Using the *Set Covering Oriented* Reformulation.**

If one wants to design exact methods for the *Stop-Number* problem, which will outperform CPLEX12 and solve larger size instances, then one needs to be able to compute a better lower bound that the standard one, obtained through relaxation of the integrality constraint. One may think into getting a better lower bound by relaxing the integrality constraint of the *Set Covering Oriented Stop-Number* Reformulation, and solving it through column generation. This would lead to an exact resolution of the *Stop-Number* Problem through *Branch and Price* algorithm.

Let us suppose that we are provided with an active service subset S_0 of the feasible service set *S*, and that we just solved the restriction of the *Stop-Number Set Covering* Reformulation. Then we are provided with a related dual solution ($\lambda = (\lambda_j, j = 1..m)$), and generating a new service $s = (J, h)$ means solving the following ILP model:

ILP *Stop-Number-Price* Model:
{***Unknown vectors***:

- $Z = (Z_{j,h}, j = 1..m, h = 0..H)$ with {0, 1} values : $Z_{j,h} = 1$ if (j, h) is in *s* ;
- $T = (T_i, i = 1..(2+H)n\text{-}1$, with {0, 1} values : $T_i = 1$ if $i \in$ *Stop-Node*(*s*).

Constraints:

- For any $i = 0..(2+H).n\text{-}1$: $\Sigma_{(j, h) \in Cross(i)} L(j).Z_{j,h} \leq CAP$;
- For any $j = 1..m$, $\Sigma_{h = 0..H} Z_{j,h} \leq 1$; (**No Redundancy Constraint**)
- For any $i = 1..(2+H).n\text{-}1$, (j, h) in *Stop*(*i*): $Z_{j,h} \leq T_i$; (**Coupling Constraints**)

Maximize : $\Sigma_j \lambda_j . Z_{j,h} - \Sigma_i T_i$, which should be > 0}.

If we suppose that we are able to efficiently solve this model, then it turns out that relaxing the integrality constraint of the *Stop-Number Set Covering oriented* Reformulation provides us with a very good lower bound, as shown by the following table 2:

- *Gr_n-m-H-t-CAP* has the same meaning as in sub-section ***3.2***.
- *OPT* is the optimal value of *Stop-Number*, and *V* is the lower bound obtained by relaxing the integrality constraint in the *Stop-Number Set Covering* oriented Reformulation;
- *CPU* is the CPU-Time (in seconds) which was necessary in order to get the *V* value;

- *COL* is the number of columns which have been generated in order to get *V*.

Instance	K_{Stop}	*OPT*	*V*	*GAP (%)*	*CPU*	*COL*
Gr_10-28-0-5-10	6	29	28.63	1.29	0.97	82
Gr_10-28-1-5-10	4	24	23.40	2.51	26.09	224
Gr_10-28-2-5-10	3	22	22.00	0.00	232.49	413
Gr_10-28-3-5-10	2	22	21.69	1.42	2278.34	539
Gr_10-28-4-5-10	2	22	21.50	2.27	3045.29	508
Gr_10-33-0-5-10	10	37	37.00	0.00	2.50	100
Gr_10-33-1-5-10	5	31	30.34	2.12	56.96	287
Gr_10-33-2-5-10	4	29	28.67	1.14	314.39	384
Gr_10-33-3-5-10	3	29	28.20	2.76	3724.86	446
Gr_10-33-4-5-10	2	29	Fail-		time out	499
Gr_10-38-0-5-10	8	33	31.85	3.49	55.44	199
Gr_10-38-1-5-10	4	29	27.72	4.42	117.82	372
Gr_10-38-2-5-10	3	28	26.70	4.64	1624.9	533
Gr_10-38-3-5-10	2	28	Fail-		time out	582
Gr_10-38-4-5-10	2	27	Fail-		time out	626
Gr_15-42-0-5-10	12	50	50.00	0.00	3.35	117

Table 2: ***Column Generation with No Hypothesis about the loads L(j), j = 1..m.***

But unfortunately, the issue which is raised by the *Stop-Number-Price* problem appears to be difficult.

Practically, running times required by CPLEX12 in order to run *Stop-Number-Price* do not allow to use it inside a global *Branch/Price* program. Experiments show that relaxing the integrality constraint in *Stop-Number-Price* usually provides a very poor lower bound. As a matter of fact, the very specific structure of the objective function, which is made of the difference of two antagonistic quantities:
$\Sigma_j \lambda_j .Z_{j,h}$ and $\Sigma_i T_i$, is a part of the difficulty. Moreover, we may check that:

Theorem 3: *The Stop-Number-Price problem is NP-Hard, even in the case when H = 0, when the loads L(j), j = 1..m, are all equal to 1, and when CAP = + ∞.*

Proof: As a matter of fact, we only need to prove that if *Q* is some integral number in {1, .. , *n*-1}, then solving *Stop-Number-Price* while imposing and $\Sigma_i T_i = Q$ is NP-Hard. Let us consider any non oriented graph *G* whose node set is the set {1, .. , *n*-1}. With every edge [*x*, *y*], *x* < *y*, in this graph we may associate some demand *j* such that *o*(*j*) = *x* and *d*(*j*) = *y*, and set $\lambda_j = 1$. Then we see that computing a solution (*Z*, *T*) of *Stop-Number-Price* such that $\Sigma_i T_i = Q$ and $\Sigma_j \lambda_j .Z_{j,h} \geq Q.(Q-1)/2$ means finding a complete sub-graph of *G* with exactly *Q* nodes. *End-Proof.*

Still, if *H* = 0, and if *CAP* is considered as being fixed, our problem happens to be time-polynomial:

Theorem 4: *In case CAP is fixed and H = 0, then Stop-Number-Price can be solved in polynomial time.*

Proof. It can be solved through dynamic programming, as a shortest path problem set on an acyclic digraph F with a number of vertices which depends in a polynomial way on n and m. We define a *state* as being any integer valued vector $V = (V_1,..,V_{CAP})$, such that $n \geq V_1 \geq V_2 \geq .. \geq V_{CAP} \geq -1$. A *state* corresponds to the possible *load profile function*s of a vehicle when it arrives on some *stop node i*, while being loaded according to decisions which have been taken on nodes i' such that $i' \leq i$: the vehicle will carry a load $\geq c$ when arriving at any node i_1 such that $i_1 \leq i + V_c$ and this load is going to be less than c when the vehicle leaves node $i + V_c$; $V_1 = -1$ means that the vehicle is empty when it arrives to i. Clearly, vector V tells us whether the vehicle is supposed to unload at node i, and, so, whether i is currently scheduled as a *stop node* for the vehicle. We denote by SV the set of all possible states. Then the nodes in the digraph F are all 3-uples (i, j, V), $i = 0..n$-1, $j = 1..m$, such that $o(j) = i$, $V \in SV$, augmented with 2 fictitious nodes *Start* and *End*. An arc $((i_0, j_0, V_0), (i_1, j_1, V_1))$ exists in F if one of the following cases holds:

- Case 1: $i_1 = i_0 + 1$, j_1 = (smallest index value j, such that $o(j) = i_1$), $V_1 = V_0 - 1$: the meaning is that the vehicle moves from i_0 to $i_0 + 1$ without loading $L(j_0)$;
- Case 2: $i_1 = i_0 + 1$, j_1 = (smallest index value j, such that $o(j) = i_1$), V_1 derives from V_0 by taking into account the additional load $L(j_0)$ between $o(j_0) = i_0$ and $d(j_0)$ and the fact the vehicle has been moving from i_0 to $i_0 + 1$ after loading $L(j_0)$: the meaning is that the vehicle moves from i_0 to $i_0 + 1$ after loading $L(j_0)$;
- Case 3: $i_1 = i_0$, j_1 = (smallest index value $j > j_0$, such that $o(j) = i_1$), $V_1 = V_0$: the meaning is that the vehicle remains located at i_0 and does not load $L(j_0)$;
- Case 4: $i_1 = i_0 + 1$, j_1 = (smallest index value $j > j_0$, such that $o(j) = i_1$), V_1 derives from V_0 by taking into account the additional load $L(j_0)$ between $o(j_0) = i_0$ and $d(j_0)$: the meaning is that the vehicle remains located at i_0 and decide to load $L(j_0)$.

In cases 1 and 3, the arc $((i_0, j_0, V_0), (i_1, j_1, V_1))$ is provided with a null length. In cases 2 and 4, it is provided with a length which expresses the impact on the *Stop Number* of the insertion of demand j_0 into of the vehicle, taking into account current state V_0. One easily checks that solving our *Stop-Number-Price* problem means computing a shortest path in this acyclic digraph F. *End-Proof.*

3.3 Experimental Comparison of the Stop-Number, Vehicle-Number, and Global-Riding Models.

The tests which we present here aim at comparing the behavior of the variants *Vehicle-Number* and *Global-Riding* of *Stop-Number*. At stake is the behavior of any multi-criterion model which would simultaneously involve not only reliability but also some of the usual criteria of the DARP models: global running costs, individual quality of service criteria. Our intuition is that, while minimizing the *Vehicle Number* or the *Global Riding Time* has probably no impact on the *Stop Number*, conversely, minimizing the *Stop Number* should tend to optimize those other criteria.

In do it while considering the same instances as in previous sub-section ***3.2***., and solve the related *Vehicle-Number* and *Global-Riding* models while using the CPLEX12 library. We denote by:

- K_{Rid} and K_{Veh} respectively the *vehicle number K* associated with optimal solution of the *Global-Riding* and the *Vehicle-Number* models.
- *Gl-Rid* the optimal value of *Global-Riding*(K_{Rid}); *Gl-Rid-Stop* the *Global Riding* value induced by an optimal value of *Stop-Number*; *Gap* the gap (*Gl-Rid-Stop* - *Gl-Rid*)/ *Gl-Rid*.

We get the following results:

Instance	K_{SVeh}	K_{Sto}	K_{Rid}	*Gl-Rid*	*Gl-Rid-Stop*	*Gap*
Gr_10-28-0-5-10	6	6	6	12	12	0.00
Gr_10-28-1-5-10	3	4	3	9	12	33.33
Gr_10-28-2-5-10	2	3	2	8	12	50.00
Gr_10-28-3-5-10	2	2	2	8	9	12.50
Gr_10-28-4-5-10	2	2	2	8	11	37.50
Gr_10-33-0-5-10	10	10	10	20	20	0.00
Gr_10-33-1-5-10	5	5	5	15	15	0.00
Gr_10-33-2-5-10	4	4	4	14	15	7.14
Gr_10-33-3-5-10	3	3	3	13	15	15.38
Gr_10-33-4-5-10	2	2	2	12	12	0.00
Gr_10-38-0-5-10	8	8	8	16	16	0.00
Gr_10-38-1-5-10	4	4	4	12	12	0.00
Gr_10-38-2-5-10	3	3	3	11	12	9.09
Gr_10-38-3-5-10	2	2	2	10	15	50.00
Gr_10-38-4-5-10	2	2	2	10	11	10.00
Gr_15-42-0-5-10	11	12	11	22	24	9.09

Table 3: ***Comparing Models (No hypothesis on the loads L(j))***

Comments: Values K_{Stop}, K_{Veh} and K_{Rid} are most often very close: minimizing the *Stop Number* or the *Global Riding* also tends to minimize the *Vehicle Number*. By the same way, *Gl-Rid-Stop* is usually close to *Gl-Rid*: minimizing the *Stop Number* also tends to minimize the *Global Riding* value.

4. A GRASP INSERTION BASED HEURISTIC for STOP-NUMBER

If we come back to the practical context which motivates the *Stop-Number* model, we should think that our ultimate goal is the real time management of automated vehicles. So, we propose now a heuristic approach, whose main characteristics is that it is likely to be suited to *on line* utilization, since it relies on an insertion mechanism: at some time during the process, while vehicles are already partially loaded, current demands are successively inserted into the vehicles. This class of algorithm links in a well-fitted way *off line* and *on line* paradigms, since, when dealing with an *on line* instance, we have to insert, in real time, newly emitted demands into the current roadmap of the vehicle fleet, while taking into account those, among the formerly emitted demands, which have not been completely services yet.

As a matter of fact, what we propose here is a GRASP heuristic (see [19]) algorithm *GRASP-Stop- Number-Insert*, which can be decomposed into a non deterministic greedy initialization process and a descent loop, according to the following scheme:

GRASP-Stop-Number-Insert (*R*: *Replication* Parameter);
For r =1..R **do**
  While all demands have not been inserted **do**
    Pick up j in {1..m}, which has not been inserted yet

and Insert *j* in the current vehicle set; (**Initialization Process**)
EndWhile
Not Stop
While Not Stop **do**
Try to perform a local transformation of the current solution (*Z, T*) which makes decrease the related quantity $\Sigma_{k,i}\ T_{k,i}$; (**Local Search Process**)
EndWhile
EndFor
Keep the best solution (*Z, T*) ever obtained.

4.1 The Initialization Process

Since we suppose here that all demands have to be accepted, and since we do not know, at the beginning of the process, the number of vehicles which are going to be involved, we use the number *K* of vehicles as a variable, while introducing a fictitious empty vehicle which helps dealing with the cases when the Insertion process requires an additional vehicle. Then the initialization loop

While all demands have not been inserted do
Pick up j in J, which has not been inserted yet and Insert j in the current vehicle set;
may be specified as follows:

GRASP-Stop-Number-Insert Initialization *loop*;

$J \leftarrow \{1..m\}$; $K \leftarrow 0$; *Service*(1) $\leftarrow$ *Nil*; (**Comment*: *Service*(*k*), $k = 1..K+1$, represents the current service of vehicle *k*; $K + 1$ denotes the *Fictitious Vehicle*, and *K* the current *Vehicle Number**)
While $J \neq Nil$ **do**
Randomly pick up $j_0 \in J$, according to some *priority* rules, and Remove it from *J*; (I1)
Compute k_0 in 1..*K*, $h_0 = 0..H$, such that inserting j_0 into *Service*(k_0) with *waiting lap* value $h(j_0) = h_0$ is feasible and the *Quality* level is the highest possible;
Insert (j_0, h_0) into *Service*(k_0); (I2)
If $k_0 = K + 1$ **then** increase *K* by 1; **EndIf**
EndWhile
The final result is provided by the Vehicle Number K and by the collection Service(k), k = 1..K.

We must provide now more details about the priority rules of instruction (I1) and about the way the *Quality* measure, which commands the choice of both the insertion vehicle k_0 and the *waiting lap* value h_0, is defined.

As a matter of fact, we do in such a way that, at any time during the process, we are provided, for any vehicle *k* and any non inserted demand in *J*, with:

- A *Profile* function P_k: for any node $i = 0..(2+H).n - 1$, $P_k(i)$ is the residual capacity of vehicle *k* between *i* and $i+1$;
- A Boolean valued function *B-Stop*: for any $i = 1..(2+H).n - 1$, $k = 1..K$, *B-Stop*(*k*, *i*) = 1 if *k* loads or unloads some inserted demand at node *i*, that means if $i \in$ *Stop-Node*(*Service*(*k*)).

This information enables us to design both priority rules of (I1) and the *Quality* measure of (I2) as follows:

Priority Rule (Instruction (I1)): at any iteration of the "*While J ≠ Nil do*" loop, we compute, for any non inserted demand j in J, the set K_j of the vehicles which have enough *residual capacity* to accept demand j, that means which are such that there exists $h = 0..H$ satisfying:

- for any $i = o(j) + h \ldots d(j) + h$ -1, $L(j) \leq P_k(i)$.

Then the *priority* rule targets demands j which are such that $Card(K_j)$ does not exceed the current value $\text{Min}_{j \in J}\, Card(K_j) + 1$.

Choosing k_0 and h_0: the Quality Criterion.

Let us suppose that the target demand j_0 has been identified. Then, in order to compute the insertion parameters k_0 and h_0, we set, for any k in K_{j0}:

- $INSER(j_0, k) = \{h = 0..H$ such that for any $i = o(j_0) + h \ldots d(j_0) + h$ -1, $L(j_0) \leq P_k(i)\}$.

$INSER(j_0, k)$ provides us with the *waiting lap* values h which make possible inserting j into *Service*(k).

Then, for any k in K_{j0} and any h in $INSER(i_0, k)$, we define the *additional stop* quantity $Add\text{-}Stop(j_0, k, h)$ as being the number $2 - B\text{-}Stop(k, o(j_0) + h.n) - B\text{-}Stop(k, d(o(j_0) + h.n)$ of additional *stop nodes* which would be induced by the insertion of j_0 into vehicle k according to *waiting lap* h.

Next, for any k in K_{j0} and any i in $1..(H+2).n - 1$, we define the *pyramidal residual capacity* of vehicle k in node i as being the quantity $PRC(k, i) = \Sigma_{i1 \in Z}\, P_k(i\text{-}i_1).\Phi(i_1)$, where the *pyramidal shape function* Φ is defined by:

- $\Phi(t) = 0$ if $Abs(t)$ = Absolute Value of $t \geq n$;
- $\Phi(t) = (1 - Abs(t)/n)$ else.

Intuitively, this *pyramidal residual capacity* evaluates the feasibility of the insertion into k of a demand j which would induce a stop in i. It is computed as the result of a kind of convolution sum with kernel function Φ.

Then our *Quality* criterion is defined as the *additional stop* quantity $Add\text{-}Stop(j_0, k, h)$, and, in case of tie-breaks, by the value $PRC(k, i)$ of the *pyramidal residual capacity* which measures the reusability of the additional stop node i in case we decide to insert j_0 into k according to *waiting lap* h. That means that, in case we need to create additional stop nodes for target vehicle k_0, we do it in such a way that those additional nodes become reusable, and that demands which involve the same stop nodes might also be inserted into vehicle k_0.

More specifically, we consider 3 cases:

- First case: there exists k, h such that $Add\text{-}Stop(j_0, k, h) = 0$. Then we randomly choose k_0, h_0 such that $Add\text{-}Stop(j_0, k_0, h_0) = 0$, and assign vehicle k_0 and waiting time h_0 to demand j_0.
- Second case: the first case does not hold, but there exist pairs (k, h) such that $Add\text{-}Stop(j_0, k, h) = 1$. For any such a pair (k, h), assigning vehicle k and waiting time h to j_0 means creating an additional stop node $i(k, h)$ for k. Then we choose (k_0, h_0) such that the resulting *pyramidal residual capacity* $PRC(k_0, i(k_0, h_0))$ be the largest possible.
- Third case : neither the first case nor the second case holds. That means that for any pair (k, h), $h \in INSER(j_0, k)$, assigning vehicle k and waiting time h to j_0 means creating 2 additional stop nodes $i_1(k, h)$ and $i_2(k, h)$. Then we choose (k_0, h_0) in such a way that the *pyramidal residual capacity* $PRC(k_0, i_1(k_0, h_0)) + PRC(k_0, i_2(k_0, h_0)$ be the largest possible.

Remark: only the third case makes possible the use of a new vehicle. Experiments show that this process also tends to minimize the *Vehicle Number K*.

4.2 The Local Search Process

We take here advantage of the generic features of the insertion mechanism, which give rise in a natural way to a local transformation operator *Trans-Insert*($J1$):

Trans-Insert (J_1: *Removal* Parameter):
Remove all demands of J_1 from the current solution K, *Service*(k), $k = 1..K$; (I3) Reinsert the demands of J_1 into the solution, while following the loop (I1) of the above *Grasp-Stop-Number-Insert-Initialization* process, and considering that demands of $J - J_1$ remain inserted as before.

Then the local search process works as follows:

Grasp-Stop-Number-Insert-LS Process (P: *Trial* Parameter; m_1: *Size* Parameter):
We start from a current solution K, *Service*(k), $k = 1..K$; Not *Stop*; **While** Not *Stop* **do**   Not *Stop1*; *Counter* ← 1;   **While** Not *Stop1* and *Counter* ≤ P **do**   Pick up a subset J_1 of {1..m}, with cardinality m_1;   Try *Trans-Insert*(J_1);   **If** the resulting *Stop Number* $\Sigma_{k,i}\ T_{k,i}$; is improved **then**   *Stop1;* Perform *Trans-Insert*(m_1);   **Else** *Counter* ← *Counter* + 1;   **EndIf**   **EndWhile**   **If** *Counter* > P **then** *Stop*; **EndIf** **EndWhile**

P and m_1 act as parameters for the above local search process.

Performing the *Removal* (I3) Instruction: What remains to be specified here, is the way we proceed in order to perform the (I3) instruction which chooses those demands of J_1 which are removed at every trial of the *Trans-Insert* operator. The idea is to choose those demands which may be considered as being poorly inserted, that means whose removal is likely to make decrease in a significant way the *Stop Numbe*r. More precisely, we consider a parameter value Q and we set, for any (i, k) such that $T_{i,k}$ is non null: $NV(i, k)$ = the number of demands which have been assigned to vehicle k and which are currently scheduled to load or unload in i.

Then we perform the (I3) instruction by computing the set W_0 of the $2.Q$ pairs (i, k) with smallest $NV(i, k)$ value, by next randomly selecting a subset W_1 of Q pairs inside W_0, and by finally removing all demands j which contribute to some of the $NV(i, k)$ quantities, (i, k) in W_1 . Clearly, Q also becomes a parameter of the *GRASP-Stop-Number-Insert* algorithm.

4.3 Numerical Experiments: Behavior of the Initialization Process

We consider the instances *Gr_n-m-H-L-CAP* as before. *R* denotes here the value of the *Replication* parameter. For *R* = 1, 10, 100, 1000:

- V_{Ini} denotes the value of the best solution ever obtained at the end of the process,
- *K* denotes the number of vehicles involved in this solution,
- *GAP* denotes the gap between V_{Ini} and *OPT*.
- In the case *R* = 1000, we also provides (Tables 5, 5.bis),
    - the CPU time (in seconds) which was required in order to run the process,
    - the worst stop number *V-Max* obtained during the process,
    - the value *V-Mean* of the average value of the *Stop Number* obtained for all 1000 replications. We get:

	R = 1			***R =10***			***R = 100***			***R = 1000***		
Instance	V_{Ini}	***K***	***GAP(%)***	V_{Ini}	***K***	***GAP (%)***	V_{Ini}	***K***	***GAP (%)***	V_{Ini}	***K***	***GAP (%)***
Gr_10-28-0-5-10	37	7	27.59	31	6	6.90	31	6	6.90	29	6	0.00
Gr_10-28-1-5-10	34	4	41.67	29	4	20.83	28	3	16.67	27	3	12.50
Gr_10-28-2-5-10	39	3	77.27	28	2	27.27	27	2	22.73	25	2	13.64
Gr_10-28-3-5-10	30	2	36.36	30	2	36.36	26	2	18.18	24	2	9.09
Gr_10-28-4-5-10	32	2	45.45	29	2	31.82	26	2	18.18	25	2	13.64
Gr_10-33-0-5-10	43	10	16.22	39	10	5.41	38	10	2.70	37	10	0.00
Gr_10-33-1-5-10	38	5	22.58	37	5	19.35	35	5	12.90	34	5	9.68
Gr_10-33-2-5-10	37	4	27.59	34	4	17.24	34	4	17.24	32	4	10.34
Gr_10-33-3-5-10	33	3	13.79	33	3	13.79	33	3	13.79	32	3	10.34
Gr_10-33-4-5-10	36	2	24.14	35	3	20.69	32	2	10.34	31	2	6.90
Gr_10-38-0-5-10	36	8	9.09	36	8	9.09	36	8	9.09	35	8	6.06
Gr_10-38-1-5-10	40	5	37.93	36	4	24.14	32	4	10.34	31	4	6.90
Gr_10-38-2-5-10	37	3	32.14	32	3	14.29	32	3	14.29	30	3	7.14
Gr_10-38-3-5-10	33	2	17.86	33	2	17.86	32	2	14.29	30	2	7.14
Gr_10-38-4-5-10	39	2	44.44	34	2	25.93	31	2	14.81	31	2	14.81
Gr_15-42-0-5-10	57	11	14.00	55	11	10.00	53	12	6.00	52	12	4.00

Table 4: ***Impact of the Replication Parameter with no hypothesis on loads L(j).***

	R = 1			***R =10***			***R = 100***			***R = 1000***		
Instance	V_{Ini}	***K***	***GAP(%)***	V_{Ini}	***K***	***GAP (%)***	V_{Ini}	***K***	***GAP (%)***	V_{Ini}	***K***	***GAP (%)***
Gr_10-28-0-1-5	33	4	43.48	31	4	34.78	26	4	13.04	26	4	13.04
Gr_10-28-1-1-5	28	2	40.00	25	2	25.00	23	2	15.00	22	2	10.00
Gr_10-28-2-1-5	31	2	63.16	22	2	15.79	21	2	10.53	21	2	10.53
Gr_10-28-3-1-5	25	1	38.89	21	1	16.67	21	1	16.67	20	1	11.11
Gr_10-28-4-1-5	27	1	50.00	22	1	22.22	21	1	16.67	20	1	11.11
Gr_10-33-0-1-5	38	4	46.15	33	4	26.92	32	4	23.08	31	4	19.23
Gr_10-33-1-1-5	29	2	31.82	29	2	31.82	26	2	18.18	25	2	13.64
Gr_10-33-2-1-5	29	2	38.10	29	2	38.10	27	2	28.57	24	2	14.29
Gr_10-33-3-1-5	29	1	45.00	24	2	20.00	24	2	20.00	23	1	15.00
Gr_10-33-4-1-5	33	1	65.00	25	1	25.00	24	1	20.00	23	1	15.00

Gr_10-38-0-1-5	38	5	52.00	35	5	40.00	32	5	28.00	31	5	24.00
Gr_10-38-1-1-5	30	3	36.36	28	3	27.27	28	3	27.27	28	3	27.27
Gr_10-38-2-1-5	30	2	42.86	30	2	42.86	28	2	33.33	26	2	23.81
Gr_10-38-3-1-5	32	2	52.38	28	2	33.33	26	2	23.81	24	2	14.29
Gr_10-38-4-1-5	29	1	38.10	26	1	23.81	25	1	19.05	25	1	19.05
Gr_15-42-0-1-5	51	6	37.84	50	6	35.14	48	6	29.73	46	6	24.32

Table 4.bis: ***The Unit Case***

Instance	V_{Ini}	***V-Max***	***V-Mean***	***CPU (s)***
Gr_10-28-0-5-10	29	41	34.42	0.05
Gr_10-28-1-5-10	27	39	32.74	0.06
Gr_10-28-2-5-10	25	39	32.10	0.07
Gr_10-28-3-5-10	24	38	30.28	0.07
Gr_10-28-4-5-10	25	36	30.47	0.08
Gr_10-33-0-5-10	37	46	41.28	0.08
Gr_10-33-1-5-10	34	44	38.51	0.09
Gr_10-33-2-5-10	32	45	37.85	0.11
Gr_10-33-3-5-10	32	43	37.43	0.12
Gr_10-33-4-5-10	31	43	36.70	0.12
Gr_10-38-0-5-10	35	49	40.56	0.10
Gr_10-38-1-5-10	31	46	37.67	0.12
Gr_10-38-2-5-10	30	44	36.97	0.13
Gr_10-38-3-5-10	30	45	36.21	0.14
Gr_10-38-4-5-10	31	42	36.50	0.15
Gr_15-42-0-5-10	52	63	57.11	0.16

Table 5: ***More on the Case R =1000 with no hypothesis on loads L(j).***

4.4 Numerical Experiments: Behavior of the whole GRASP Process

V_{Gr}, *K*, *GAP*, *CPU* are as in tables 4 and 5, but for the whole GRASP process. *IMP* is the improvement ration $IMP = (V_{Ini} - V_{Gr})/V_{Ini}$ induced by the local search loop.

	R = 1			***R = 100***			***R = 1000***				
Instance	V_{Gr}	***K***	***GAP(%)***	V_{Gr}	***K***	***GAP(%)***	V_{Gr}	***K***	***GAP(%)***	***IMP (%)***	***CPU***
Gr_10-28-0-5-10	32	7	10.34	29	6	0.00	29	6	0.00	0.00	0.10
Gr_10-28-1-5-10	29	4	20.83	25	4	4.17	25	4	4.17	7.41	0.12
Gr_10-28-2-5-10	28	3	27.27	25	3	13.64	24	3	9.09	4.00	0.12
Gr_10-28-3-5-10	28	2	27.27	24	2	9.09	23	2	4.55	4.17	0.11
Gr_10-28-4-5-10	25	2	13.64	24	2	9.09	23	2	4.55	8.00	0.13
Gr_10-33-0-5-10	43	10	16.22	38	10	2.70	37	10	0.00	0.00	0.17
Gr_10-33-1-5-10	36	6	16.13	33	5	6.45	33	5	6.45	2.94	0.17
Gr_10-33-2-5-10	34	4	17.24	31	4	6.90	31	4	6.90	3.13	0.20

Gr_10-33-3-5-10	33	3	13.79	32	3	10.34	31	3	6.90	3.13	0.20
Gr_10-33-4-5-10	34	3	17.24	32	2	10.34	30	2	3.45	3.23	0.18
Gr_10-38-0-5-10	36	8	9.09	33	8	0.00	33	8	0.00	5.71	0.17
Gr_10-38-1-5-10	34	5	17.24	31	5	6.90	31	4	6.90	0.00	0.17
Gr_10-38-2-5-10	37	3	32.14	31	3	10.71	30	3	7.14	0.00	0.18
Gr_10-38-3-5-10	32	2	14.29	31	2	10.71	29	2	3.57	3.33	0.18
Gr_10-38-4-5-10	32	2	18.52	30	2	11.11	30	2	11.11	3.23	0.20
Gr_15-42-0-5-10	53	12	6.00	52	12	4.00	50	12	0.00	3.85	0.38

Table 6: ***Behavior of the GRASP Proces, with no hypothesis on loads L(j).***

	R = 1			***R = 100***			***R = 1000***				
Instance	V_{Gr}	***K***	***GAP(%)***	V_{Gr}	***K***	***GAP(%)***	V_{Gr}	***K***	***GAP(%)***	***IMP (%)***	***CPU (s)***
Gr_10-28-0-1-5	27	4	17.39	24	4	4.35	23	4	0.00	11.54	0.07
Gr_10-28-1-1-5	25	2	25.00	20	2	0.00	20	2	0.00	9.09	0.07
Gr_10-28-2-1-5	26	2	36.84	21	2	10.53	19	2	0.00	9.52	0.08
Gr_10-28-3-1-5	19	1	5.56	19	1	5.56	19	1	5.56	5.00	0.08
Gr_10-28-4-1-5	26	1	44.44	19	1	5.56	18	1	0.00	10.00	0.09
Gr_10-33-0-1-5	32	4	23.08	28	4	7.69	27	4	3.85	12.90	0.10
Gr_10-33-1-1-5	27	2	22.73	24	2	9.09	24	2	9.09	4.00	0.09
Gr_10-33-2-1-5	29	2	38.10	23	2	9.52	22	2	4.76	8.33	0.10
Gr_10-33-3-1-5	27	1	35.00	22	1	10.00	22	1	10.00	4.35	0.10
Gr_10-33-4-1-5	24	1	20.00	22	1	10.00	21	1	5.00	8.70	0.11
Gr_10-38-0-1-5	31	5	24.00	27	5	8.00	27	5	8.00	12.90	0.10
Gr_10-38-1-1-5	30	3	36.36	25	3	13.64	24	3	9.09	14.29	0.11
Gr_10-38-2-1-5	28	2	33.33	24	2	14.29	24	2	14.29	7.69	0.12
Gr_10-38-3-1-5	26	2	23.81	24	2	14.29	23	2	9.52	4.17	0.12
Gr_10-38-4-1-5	26	1	23.81	24	1	14.29	24	1	14.29	4.00	0.13
Gr_15-42-0-1-5	42	6	13.51	38	6	2.70	38	6	2.70	17.39	0.24

Table 6.bis: ***The Unit Case***

4.5 Discussion about the Experiments.

We notice that the local search loop induces a very satisfactory improvement ratio. Globally, GRASP, combined with the insertion mechanism, appears as very well-fitted to the current situation, since it provides us with small error gaps and small running costs , while being very easy to adapt, because of its generic features, to real time contexts.

References

1. J.W. BAUGH, D.K. KAKIVAYA., J.R.STONE, Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Engineering Optimization, 30(2), 91-124, (1998).
2. Ü. BILGE., J.M. TANCHOCO, AGV Systems with Multi-Load Carriers: Basic Issues and Potential Benefits, Journal of Manufacturing Systems, 16 (3) , 159–174, (1997).

3. R. BORNDORFER, M. GROTSCHEL, F. KLOSTERMEINER, C. KUTTNER, Telebus Berlin: Vehicle routing scheduling in a dial a ride system, Konrad Zuse Zent. Info. Tech. Berlin, (1997)
4. P. COLL, J. MARENCO, I. DIAZ, P. ZABALA, Facets of the graph coloring polytope, Ann. Operat. Res. 116, p 79-90, (2002)
5. J.F. CORDEAU, G. LAPORTE, Dial/Ride: models and algorithms; An. OR 153-1, p 29-46, (2007)
6. JF. CORDEAU, A branch and cut algorithm for the Dia/Ride, Ope. Res. 54-3, p 573-586, (2006)
7. J.F. CORDEAU, M. GENDREAU, G. LAPORTE, J.Y. POTVIN, F. SEMET, A guide to vehicle routing heuristics, Jour. Op. Res. Soc., 53-5, p 512-522 (2002)
8. D. CORNAZ, V. JOST, A one to one correspondence between coloring and stable sets, Operat. Res. Letters (36), p 673-676 (2008)
9. De PAEPE, K.K. LENSTRA, S. JGALL, R. SITTERS, L. STOUGIE, Computer aided complexity classification of Dial Ride Problems, INFORMS Journal of Computing 22, p 1130-1152 (2004)
10. J. DESROSIERS, Y.DUMAS, F.SOUMIS, A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences, 6, 301–325, (1986).
11. M.GOLUMBIC, Algorithmic Graph Theory and Perfect Graphs, Academic Press (1980)
12. P. HANSEN, M. LABBE, D. SCHINDL, Set covering and packing formulations of graph coloring: algorithms and first polyhedral results, Discrete Optimization 6, p 135-147 (2009)
13. L. HIGGINS, J.B. LAUGHLIN, K. TURNBULL, Automatic vehicle location and advanced paratransit at Houston METROLift, Proc Transport. 2000 Research Board Conf. (2000)
14. R.M. JORGENSEN, J.LARSEN, K.B. BBERGVINSDOTTIR, Solving the dial-a-ride problem using genetic algorithms. Journal of the Operational Research Society, 58(10), 1321-1331, (2007).
15. Y. LUO, P. SCHONFELD, Reinsertion heuristic for static Dial/Ride, Trans. Res. B 41, p 736-755 (2007)
16. E. MALAGUTI, M. MONACCI, P. TOTH, An exact approach for the vertex coloring problem, Disc. Optimization (2010)
17. A. MEHROTRA, M.A. TRICK: A column generation approach for graph coloring, INFORMS Journal of Computing (8), p 344-354 (1996)
18. SCIIP, URL http://scip.zib.de.
19. I. MENDEZ-DIAZ, P. ZABALA, A cutting plane algorithm for graph coloring, Disc. Applied Math. 154, p 826-847 (2006)
20. K. PALMER, M. DESSOUKY, T. ABELMAGUID, Impact of management practices and advanced technologies on demand responsive transit systems, Transportation Research A 38, p 495-509 (2004)
21. C.H. PAPADIMITRIOU, M. YANNANAKIS, Scheduling interval ordered tasks, SIAM Journ. Computing 8, pp 405-409 (1979)
22. S.N. PARRAGH, K.F.DOERNER, R.F.HARTL, Variable neighborhood search for the dial-a-ride problem, Computers & Operations Research, 37, 1129–1138, (2010)
23. A. QUILLIOT, S. DELEPLANQUE, Constraint propagation for the Dial and Ride problem wit split loads, Recent Advances in Computational Optimization, Studies in Computational Intelligence, Vol 470, p 31-50, Springer (2013)
24. M. RESENDE, C. RIBEIRO, Greedy Random Adaptative Procedure (2002)

MISTA 2015

A mass-flow MILP formulation for energy-efficient supplying in assembly lines

Maria Muguerza · Cyril Briand · Nicolas Jozefowiez · Sandra U. Ngueveu · Victoria Rodríguez · Matias Urenda Moris

Abstract This paper focuses on the problem of supplying the workstations of assembly lines with components during the production process. For that specific problem, this paper presents a Mixed Integer Linear Program (MILP) that aims at minimizing the energy consumption of the supplying strategy. More specifically, in contrast of the usual formulations that only consider component flows, this MILP handles the mass flow that are routed from one workstation to the other.

Keywords Supplying strategy · Assembly lines · Energy efficiency · MILP.

1 Introduction

In general, feeding systems of assembly lines are composed by a central warehouse, several workstations organized in sequence and a fleet of vehicles (tow trains) in charge of delivering the components to the workstations. The components are packed in pallet or boxes. The supermarket is a decentralized area of material supplies, located next to the assembly line. For building up a supplying strategy, time is discretized in a set of delivering periods. For each period, a workstation has a component consumption (possibly periodic) expressed in terms of boxes. At each tour, the tow trains load the boxes which have to be shipped to the assembly line, follow a supplying route, and stop at the appropriate workstations for delivering its boxes. The supplying routes are usually fixed and start and finish at the supermarket. The number of boxes that a

This work was supported by the ECO-INNOVERA-1rst call EASY (ANR-12-INOV-0002).

Maria Muguerza · Cyril Briand · Nicolas Jozefowiez · Sandra U. Ngueveu
CNRS, LAAS, UPS, INSA, INP, 7 avenue du colonel Roche, F-31400 Toulouse, France
E-mail: (briand,njozefow,ngueveu)@laas.fr

Victoria Rodríguez
Economics and Management School, University of Navarra,, 31080 Pamplona, Spain
E-mail: vrodriguez@unav.es

Matias Urenda Moris
Virtual Systems Research Centre, University of Skvde, PO Box 408, 54128 Skvde, Sweden
E-mail: matias.urenda.moris@his.se

tow train can transport in the same tour is limited. The number of boxes available at each workstation should never exceed the storage capacity of the workstation (which is usually low). A supplying strategy defines whether a vehicle has to stop at each workstation at each time period and the number of boxes that should be delivered.

In a world where natural resources are limited, issues related to energy efficiency are becoming more and more important. Vehicles in factories travel a significant quantity of kilometers for supplying the workstations, causing effects in economic and energetic expenses. Whether they use electric energy or fossil fuel, their energetic consumption is not negligible and more and more attention has to be paid for exhibiting energy-efficient supplying strategies. Several factors that are inherent to the problem have impact on the energy consumption. We are interested in determining the most significant ones.

In the literature related to the Vehicle Routing Problem (VRP), some researchers take interest in minimizing carbon dioxide emissions. One of the major contribution is due to Bektas and Laporte [2] who present the Pollution-Routing Problem (PRP) as an extension of the VRP with Time Windows. The PRP consists of routing a number of vehicles to serve a set of customers within preset time windows, and determining their speed on each route segment, so as to minimize a function comprising emissions and driver costs. The author propose a MILP formulation that allows to optimize both load and speed of the vehicles.The idea of controlling the vehicle velocity on each route segment is fruitful for improving the energy efficiency in the context of long distance transportation problem. However, in a very local transportation context, as the distances travelled during the acceleration phase becomes non-negligible with respect to the one covered at the maximum speed, other parameters can impact the energy consumption.

The problem considered in this paper can be viewed as a particular Inventory Routing Problem (IRP) . We intend to show that minimizing the travelled distance does not necessarily implies the minimization of the energy. We prove that other parameters can significantly influence the energy spending. The remainder of this paper is organized as follows. An energy consumption analysis is proposed in Section 2. A MILP for energy optimization is described in Section 3.

2 Energy modeling

The forces that have more influence on the power consumed by the vehicle are: the traction force ($F_t = m_T a(t)$) and the rolling resistance ($F_r m_T = gC_r$), in Newtons (N). The traction force is used to generate motion between an object and a tangential surface, and it depends on the mass (m_T) and the acceleration of the vehicle ($a(t)$). The rolling resistance is the force resisting the motion when a body rolls on a surface and varies in function of the load (m_T), the rolling coefficient (C_r) and the gravity (g). The parameter (m_T) represents the mass of the vehicle plus the transported load, which varies along the tour. The expresion of the energy consumption is $E = \int m_T(a(t) + gC_r)v(t)dt$.

For sake of simplicity, the acceleration, the deceleration and the maximum speed are assumed known and constant.Thanks to the literature, the rolling coefficient is also known. Regarding the energy consumed between two workstations. Three phases are distinguished according to the vehicle state. The first phase corresponds to the acceleration phase where a peak of energy is produced, due to the acceleration. The

second phase begins when the speed of the vehicle reach its maximum value. Finally, in the deceleration phase, the energy consumption is null.

The travelled distance is directly linked to the energy although it is not the only significant parameter. Indeed, the energy consumption is different depending on the way of delivering the load. The stops at the workstations also have effects on the energy demand. Decreasing the number of vehicle stops in every workstation can reduces the number of acceleration phases, hence the energy.

3 Energy-aware mathematical modeling

In this section, a mixed integer linear programming (MILP) model is presented. This model integrates the previous influential factors, and similarly to the formulation poposed in [1], takes advantage from a basic flow formulation. Nonetheless, instead of taking the flow in terms of number of components into account, the mass of the shipped components is considered. We assume that only one kind of pallet can be delivered to a given workstation, each having a well-known mass. Therefore, once the mass of delivered components known, the number of components can be easily deduced. Reasoning in terms of masses is interesting since the energy spent for bringing a pallet to one location i to another location j is proportional to its mass. Therefore, one can considered directly inside the MILP formulation the energy cost (C_{ij}), which represents the energy consumption for shipping one mass unit directly from i to j with $j > i$.

The component mass brought from i to j during period t is noted M_{ij}^t. Decision variables Z_i^t represent the number of components left at workstation i during period t. They can easily be deduced from the values of the M_{ij}^t variables. Eventually, inventory flow variables IL_i^t, deduced from the Z_i^t values, are also modelled.

Using the above decision variables, the energy minimization MILP can be formulated as follows. The objective function (1) aims at minimizing the energy consumption, which is proportional to the mass M_{ij}^t traversing each arc (i,j) during period t. Constraints (2) model flow conservation together with demand satisfaction. Constraints (3) ensure that the vehicle capacity A is never exceeded and enforce variables Y^t to be set to one when a tour is carried out in period t. The set of equations (4) ensures that the inventory level at workstation i never exceeds the workstation storage capacity c_i. Constraints (5) enforce the difference $\frac{1}{m_i}(\sum_{j<i} M_{j,i}^t - \sum_{j>i} M_{i,j}^t)$ to be integral. The constraints (6) impose that the mass brought back to the depot equals the vehicle mass (0 and $n+1$ being two virtual nodes associated with the depot). Constraints (7) ensure that, whether some components are delivered in workstation i during period t, the vehicle has to stop in this station at that tour. Set of constraints (8) ensure that, whether the vehicle stops in workstation i at time t, there exist an incoming and an outcoming arc selected at workstation i during period t. Constraints (9) ensure that whether there exists a mass flow between two workstations, an arc between these stations has to be selected too. Equations (10)-(12) define the domain of each variable ($m_{\max}$ is the maximum load that the vehicle can transport).

$$\text{Min} \quad z = \sum_{i,j}^{n} \sum_{t}^{NT} C_{ij} M_{ij}^t \tag{1}$$

st:

$$Z_i^t + IL_i^{t-1} - d_i^t = IL_i^t \qquad \forall\ (i,t) \qquad (2)$$

$$\sum_{i=1}^{n} Z_i^t \leq AY^t \qquad \forall\ (t) \qquad (3)$$

$$Z_i^t + IL_i^{t-1} \leq c_i \qquad \forall\ (i,t) \qquad (4)$$

$$Z_i^t - \frac{1}{m_i}(\sum_{j<i} M_{ji}^t - \sum_{j>i} M_{ij}^t) = 0 \qquad \forall\ (i,t) \qquad (5)$$

$$\sum_{i=1}^{n} M_{in+1}^t = m_v Y^t \qquad \forall\ (t) \qquad (6)$$

$$Z_i^t \leq X_i^t c_i \qquad \forall\ (i,t) \qquad (7)$$

$$\sum_{j>i} \phi_{ij}^t = \sum_{j<i} \phi_{ji}^t = X_i^t \qquad \forall\ (i,t) \qquad (8)$$

$$M_{ij}^t \leq m_{\max} \phi_{ij}^t \qquad \forall\ (i,j,t) \qquad (9)$$

$$IL_i^t, M_{ij}^t \geq 0 \qquad \forall\ (i,j,t) \qquad (10)$$

$$Z_i^t \in N \qquad \forall\ (i,t) \qquad (11)$$

$$\phi_{ij}^t, X_i^t, Y_t \in \{0,1\} \qquad \forall\ (i,j,t) \qquad (12)$$

4 Conclusion

We can conclude that taking the transported load, the number of stops and the total travelled distance simultaneously into account is worthy. We propose a MILP formulation that integrates these parameters all together inside the optimization procedure. Nevertheless, the first experiments show that the computational effort required for solving efficiently the model is high. Additional researches are needed in order to boost the optimization procedure using either more compact MILP formulations or more advanced optimization mechanisms such as valid inequalities generation, variable fixing techniques, or decomposition approaches.

References

1. C. Archetti, N. Bianchessi, S. Irnich and M.G. Speranza. Formulations for an inventory routing problem, International Transactions in Operational Research, vol. 21(3), pp 353-374 (2014).
2. T. Bektas and G. Laporte. The Pollution-Routing Problem, Transportation Research Part B 45, pp 12321250 (2011).
asy/

MISTA 2015

Resource-Aware Scheduling for Data Centers with Heterogenous Servers

Tony T. Tran[+] · **Peter Yun Zhang**[°] · **Heyse Li**[+] · **Douglas G. Down**[*] · **J. Christopher Beck**[+]

Abstract This paper presents an algorithm for resource-aware scheduling of computational jobs in a large-scale heterogeneous data center. The algorithm aims to allocate different machine configurations to job classes to attain an efficient mapping between job resource request profiles and machine resource capacity profiles. We propose a three-stage algorithm. The first stage uses a queueing model that treats the system in an aggregated manner with pooled machines and jobs represented as a fluid flow. The latter two stages use combinatorial optimization techniques to take the solution from the first stage and apply it to a more accurate representation of the data center. In the second stage, jobs and machines are discretized. A linear programming model is created to obtain a solution to the discrete problem that maximizes the system capacity. The third and final stage is a scheduling policy that uses the solution from the second stage to guide the dispatching of arriving jobs to machines. Using Google workload trace data, we show that our algorithm outperforms a benchmark greedy dispatch policy. We find that our algorithm is able to provide mean response times up to an order of magnitude smaller than the benchmark dispatch policy. These results show that it is important to consider the heterogeneity of machine configuration profiles in making effective scheduling decisions.

1 Introduction

The cloud computing paradigm of providing hardware and software remotely to end users has become very popular with applications such as e-mail, Google documents, iCloud, and dropbox. Service providers employ large data centers to provide these

[+] Department of Mechanical and Industrial Engineering,
University of Toronto
E-mail: {tran, hli, jcb}@mie.utoronto.ca

[°] Engineering Systems Division
Massachusetts Institute of Technology
E-mail: pyzhang@mit.edu

[*] Department of Computing and Software
McMaster University
E-mail: downd@mcmaster.ca

applications. As the demand for computational resources increases, the supply of services must efficiently scale. Yet, data centers represent a significant capital investment. Not only are servers for a data center expensive, maintaining and running a data center is a substantial investment. Due to the significant cost of these machines, many data centers are not purchased as a whole at one time, but rather built incrementally, adding machines in batches. Data center managers may choose machines based on the price-performance trade-off that is economically viable and favorable at the time [21]. Therefore, it is not uncommon to see data centers comprised of tens of thousands of machines, which are divided into ten or so different machine configurations, each with a large number of identical machines.

Under heavy loads, submitted jobs may have to wait for machines to become available before starting processing. These delays can be significant and can become problematic. Therefore, it is important to provide scheduling support that can directly handle the varying workloads and differing machines so that efficient routing of jobs to machines can be made. We study the problem of scheduling jobs onto machines such that the multiple resources available on a machine (e.g., processing cores and memory) can handle the assigned workload in a timely manner.

We develop an algorithm to schedule jobs on a set of heterogeneous machines to minimize mean job response time, the time from when a job enters the system until it starts processing on a machine. The algorithm consists of three stages. In the first stage a queueing model is used. Here, the system is represented at a very high level with resources and jobs both pooled. In each successive stage, a finer system model is used, such that in the third stage we generate explicit schedules for the actual system. Our experiments are based on job traces from one of Google's compute clusters [18] and show that our algorithm significantly outperforms a natural greedy policy that attempts to minimize the response time of each arrival.

The contributions of this paper are:

- The introduction of a hybrid queueing theoretic and combinatorial optimization scheduling algorithm for a data center, which efficiently maps job resource request profiles to different machine resource capacities.
- An extension to the allocation linear programming (LP) model presented in [3] and used for distributed computing in [2] to a data center that has multiple machines with multi-capacity resources.
- An empirical study of our scheduling algorithm on real workload trace data, which serves as a proof-of-concept of our proposed algorithm.

The rest of the paper is organized into a definition of the data center scheduling problem in Section 2, related work on data center scheduling in Section 3, a presentation of our proposed algorithm in Section 4, and experimental results in Section 5. Section 6 concludes our paper along with some plans for future work.

2 Problem Definition

The data center of interest is one that is comprised of many independent servers (also referred to as machines). We are interested in dealing with a server cluster that has on the order of tens of thousands of machines. These machines are not all identical; the entire machine population is divided into different configurations denoted by the set M. Machines belonging to the same configuration are identical in all aspects.

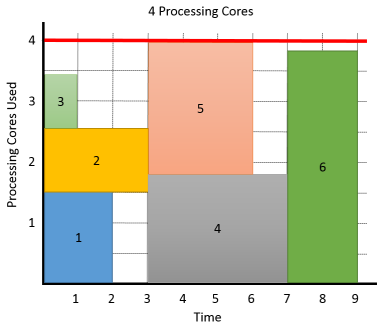


Fig. 1: Processing cores resource consumption profiles

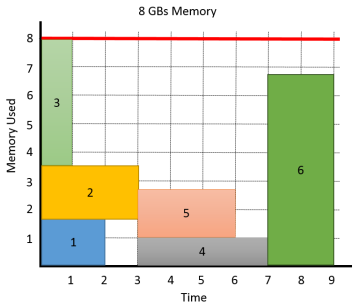


Fig. 2: Memory resource consumption profiles

We classify a machine configuration based on its resources. For example, machine resources may include the number of processing cores and the amount of memory, disk-space, and bandwidth. For our study, we generalize the system to have a set of resources, R, which are limiting resources of the data center. Each machine configuration is defined by the capacity of each resource available in the machines belonging to that configuration. A machine of configuration $j \in M$ has c_{jl} amount of resource $l \in R$ and within a configuration j there are n_j identical machines.

In our data center scheduling problem, jobs must be assigned to the machines with the goal of minimizing the mean response time of the system. We assume that jobs are assigned immediately as they arrive and the assignment cannot be changed. Jobs arrive to the data center dynamically over time. Times between arrivals are independent and identically distributed (i.i.d.). Each job belongs to one of a set of K classes where the probability of an arrival being of class k is α_k. A distribution of resource requirements for a job is defined by the class of the job. We denote the expected amount of resource of type l required by a job of class k as r_{kl}. The processing times for jobs in class k on a machine of configuration j are assumed to be i.i.d. with mean $\frac{1}{\mu_{jk}}$. The associated processing rate is thus μ_{jk}.

Each job is processed on a single machine. However, a machine can process many jobs at once, as long as the total resource usage of all concurrent jobs does not exceed the capacity of the machine. Figures 1 and 2 depict an example schedule of six jobs on a machine with two limiting resources: processing cores and memory. Here, the x-axis represents time and the y-axis is the amount of resource used. The machine has 4 processing cores and 8 GBs of memory. Note that the start and end times of each job are the same in both figures. This represents the job concurrently consuming resources from both cores and memory during its processing time.

Any jobs that do not fit within the resource capacity of a machine must wait until sufficient resources become available. We assume there is a buffer of infinite capacity for each machine where jobs can queue until they begin processing. Figure 3 illustrates the different states a job can go through in its lifetime. Each job begins outside the system and joins the data center once submitted. At this point, the job must be dispatched to one of the machines. This machine may or may not be immediately available for the job. The job must wait in the queue if there are insufficient resources, but can immediately start running if the required resources are available. If the job must join

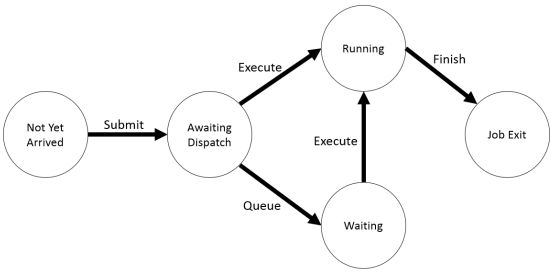


Fig. 3: Stages of job lifetime.

the queue, then it will start running on the machine when resources free up and the job has priority to start processing. Finally, after being processed, the job will exit the data center.

3 Related Work

Scheduling in data centers has received significant attention in the past decade. Many works consider cost saving through decreased energy consumption from lowering thermal levels [22, 23], powering down machines [4, 7], or geographical load balancing [13, 14]. These works often attempt to minimize costs or energy consumption while maintaining some guarantees on response time and throughput.

The literature on schedulers for distributed computing clusters has focused heavily on *fairness* and *locality* [11, 19, 24]. Optimizing these performance metrics leads to equal access of resources for different users and the improvement of performance by assigning tasks close to the location of stored data in order to reduce data transfer traffic. Locality of data has been found to be crucial for performance in systems such as MapReduce, Hadoop, and Dryad. Our work does not consider data transfer or equal access for different users. The works looking at *fairness* and *locality* also differ from our work in that our model focuses on the heterogeneity of machines with regard to resource capacity and how the mix of jobs that may be concurrently processed on a machine is a non-trivial decision.

Ghodsi et al. [8] and Grandl et al. [9] look at scheduling a system with multiple multi-capacity resources (e.g., CPU, memory, disk storage, and bandwidth). Ghodsi et al. [8] propose a scheduling policy, Dominant Resource Fairness, that aims to fairly share resources to each user based on their dominant resource. A dominant resource for each user is found by first normalizing resource requirements using the maximum capacity of the resource over all machines and then taking the resource that has the largest normalized requirement. For example, if a user requests two cores and two GB of memory and the maximum number of cores and memory on any system is four cores and eight GB, the normalized values would be 0.5 cores and 0.25 memory. The dominant resource for the user would thus be cores. Each user is then given a share of the resources such that the proportion of dominant resources for each user is equal to the dominant resource share of others. Note that this may compare resources of different types as the consideration is based on a user's dominant resource. Grandl et al. [9] study a similar

problem, but emphasize the efficient packing of jobs onto machines. They propose the Tetris scheduler, which considers a linear combination of two scoring functions: packing jobs onto machines to best fit the remaining resources, and *least remaining work first* that looks at the remaining work (duration times resource requirements) of a job. The first score favours large jobs, while the second favours small jobs. Tetris chooses the next job to process based on the job with the maximum score. They compare Tetris against Dominant Resource Fairness and show that focusing on fairness alone can lead to poor performance. Their work shows the importance of considering efficient resource allocation, an issue that has had more attention recently. However, to effectively use Tetris, a system manager must tune several parameters to customize the job score for their application. Based on the job score employed, Tetris may over-prioritize large or small jobs and thus starve jobs that do not have high scores by constantly introducing new jobs with higher priority. A comparison of our proposed algorithm and the Tetris scheduler is a key area for future work.

Kim et al. [12] study dynamic mapping of jobs to machines in a heterogeneous environment. Jobs have varying priorities and soft deadlines. They find that two scheduling heuristics stand out as the best performers: *Max-Max* and *Slack Sufferage*. In *Max-Max*, a job assignment is made by greedily choosing the mapping that has the best fitness value based on the priority level of a job, its deadline, and the job execution time. *Slack Sufferage* chooses job mappings based on which jobs suffer most if not scheduled onto their "best" machines. Al-Azzoni and Down [2] schedule jobs to machines using an allocation LP to efficiently pair job classes to machines. The solution of the LP problem maximizes the system capacity and guides the scheduling rules to reduce the long-run average number of jobs in the system. Further, they show that their heuristic policy is guaranteed to be stable if the system can be stabilized. Rasooli and Down [20] extend the allocation LP model to address a Hadoop framework. They compare their work against the default scheduler used in Hadoop and the *Fair-Sharing* algorithm and show that their algorithm greatly reduces the response time, while maintaining competitive levels of fairness with Fair-Sharing. These papers focus on job execution time as the key defining characteristic in machine heterogeneity and do not consider multi-capacity resources of machines.

Chang et al. [5] consider a grid computing system where clusters of resources have varying computing speeds and the bandwidth capacities between clusters are different. The authors develop a scoring algorithm that maps jobs to resources based on the bandwidth availability and cluster load. Maguluri et al. [17] examine a cloud computing cluster where virtual machines are to be scheduled onto servers. Virtual machines require some amount of CPU, memory, and storage space that must fit onto the servers they have been assigned to. Their work assumes that there are different types of virtual machines: *Standard*, *High-Memory*, and *High-CPU*. Each virtual machine type has specified resource requirements and different instances of virtual machines within a single type do not differ. Based on these requirements and the capacities of the servers, the authors determine all possible combinations of virtual machines that can concurrently be placed onto each server. A preemptive algorithm is presented that uses the defined virtual machine combinations. They show that their algorithm is throughput-optimal. An alternative, non-preemptive algorithm is proposed that is close to throughput optimal. The algorithm works by choosing at the beginning of a time slot the mix of virtual machine types on each server to maximize the amount of work that can be done for that time slot. An extension to their work was later done to prove a queue-length optimal algorithm for the same problem in the heavy traffic regime [16]. They propose a

routing algorithm that assigns jobs to servers with the shortest queue (similar to our greedy algorithm presented in Section 5.2) and a mix of virtual machines to assign to a server based on the same reasoning proposed for their throughput optimal algorithm. These works differ from our work since virtual machine types have predetermined resource requirements. Therefore, it is known exactly how virtual machine types will fit on a server without having to reason online about each assignment individually based on their specific requirements. Because the virtual machine sizes are set, inefficiencies due to fragmentation are not a concern as they are in our system. However, resource wastage due to fragmentation still exists from virtual machines not completely filling server capacities. Furthermore, fragmentation occurs *inside* the virtual machine as well since jobs may not use the full resources of a virtual machine type and will then occupy more resources (the size of a virtual machine) than required.

4 Data Center Scheduling

The proposed algorithm, Long Term Evaluation Scheduling (LoTES), is a three-stage queueing-theoretic and optimization hybrid approach. Figure 4 illustrates the overall scheduling algorithm. The first two stages are performed offline and are used to guide the dispatching algorithm of the third stage. The dispatching algorithm is responsible for assigning jobs to machines and is performed online. In the first stage, we use techniques from the queueing theory literature, which represent the data center as a fluid model where incoming jobs can be considered in the aggregate as a continuous flow. We extend the allocation LP model presented by Andradóttir et al. [3] to account for multiple resources. The allocation LP is used to find an efficient allocation of machine resources to job classes. In the second stage, a machine assignment LP model is used to assign specific machines to serve job classes using the results of the allocation LP. In the final stage, jobs are dispatched to machines dynamically as they arrive to the system.

4.1 Allocation LP

Andradóttir et al.'s [3] allocation LP was created for a similar problem but with a single unary resource per machine. The allocation LP finds the maximum arrival rate for a given queueing network such that stability is maintained. Stability is a formal property of queueing systems [6] that can informally be understood as the queue lengths in the system remaining bounded over time.

In our problem, there are $|R|$ resources which must be accounted for. We modify the allocation LP to accommodate these multiple resources. The model combines each machine's resources to create a single super-machine for each configuration. Thus, there will be exactly $|M|$ pooled machines (one for each configuration) with capacity $c_{jl} \times n_j$ for resource l. The allocation LP ignores resource fragmentation within the pooled machines. Fragmentation occurs when a machine's resource capacity cannot be fully utilized as a result of the currently available resources of a machine not being sufficient to admit a job, leaving resources unused with jobs waiting in queue. For example, if a configuration has 30 machines with 8 cores available on each machine and a set of jobs assigned to the configuration requires exactly 3 cores each, the pooled machine would have 240 processors that can process 80 jobs in parallel. However,

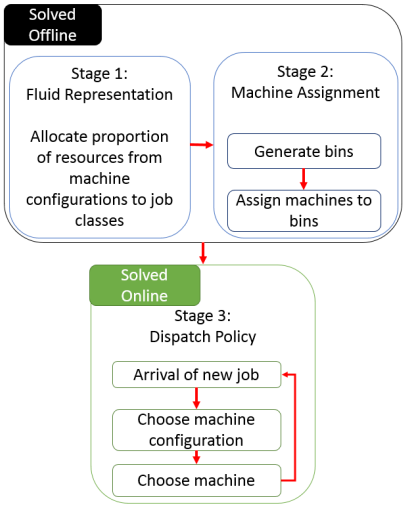


Fig. 4: LoTES Algorithm.

only 2 jobs could be placed on each individual machine. Therefore, only 60 jobs can be processed in parallel. The effect may be further amplified when multiple resources exist as fragmentation could occur for each resource. The subsequent stages of the LoTES algorithm deal with the issue of fragmentation by treating each machine individually (see Section 4.2).

The extended allocation LP is given by (1)-(5) below.

$$\max \quad \lambda \tag{1}$$

$$\text{s.t.} \sum_{j \in M} (\delta_{jkl} c_{jl} n_j) \mu_{jk} \geq \lambda \alpha_k r_{kl} \qquad k \in K, l \in R \tag{2}$$

$$\frac{\delta_{jkl} c_{jl}}{r_{kl}} = \frac{\delta_{jk1} c_{j1}}{r_{k1}} \qquad j \in M, k \in K, l \in R \tag{3}$$

$$\sum_{k \in K} \delta_{jkl} \leq 1 \qquad j \in M, l \in R \tag{4}$$

$$\delta_{jkl} \geq 0 \qquad j \in M, k \in K, l \in R \tag{5}$$

Decision Variables

λ: Arrival rate of jobs

δ_{jkl}: Fractional amount of resource l that machine j devotes to job class k

The LP assigns the fractional amount of each resource that each machine pool should allot to each job class in order to maximize the arrival rate of the system, while maintaining stability. Constraint (2) guarantees that sufficient resources are allocated for the expected requirements of each class. Constraint (3) ensures that the resource profiles of jobs (i.e., the amount of each resource a job class is expected to request) are properly enforced. For example, if the amount of memory required is twice the number of cores required, the amount of memory assigned to the job class from a single machine configuration must also be twice that of the core assignment. The allocation LP does not assign more resources than available due to constraint (4). Finally, constraint (5) ensures the non-negativity of assignments.

Solving the allocation LP will provide δ^*_{jkl} values which tell us how we can efficiently allocate jobs to machine configurations. However, due to fragmentation, the allocation LP solution is only an upper bound on the achievable arrival rate of a system. The bound for the single unary resource problem is tight: Andradóttir et al. [3] show that utilizations arbitrarily close to one are possible. This is not possible when fragmentation occurs.

4.2 Machine Assignment

In the second stage, we use the job-class-to-machine-configuration results from the allocation LP to guide the choice of a set of job classes that each machine will serve. We are concerned with fragmentation and so treat each job class and each machine discretely, building specific sets of jobs (which we call "bins") that result in tightly packed machines and then deciding which bin each machine will emulate. This stage is still done offline and so rather than using the observed resource requirements of jobs, we use the expected values.

In more detail, recall that the δ^*_{jkl} values from the allocation LP provide a fractional mapping of the resource capacity of each machine configuration to each job class. Based on the δ^*_{jkl} values that are non-zero and the particular resource requests of jobs and the capacities of the machines, the machine assignment algorithm will first create job bins. A bin is any set of jobs that together do not exceed the capacity of the machine. A non-dominated bin is a bin which is not a subset of any other bin: if any additional job is added to it, one of the machine resource constraints will be violated. Figure 5 presents the feasible region for an example machine. Assume that the machine has one resource (cores) with capacity 7. There are two job classes, job class 1 requires 2 cores and job class 2 requires 3 cores. The integer solutions within the search space represent the feasible bins. All non-dominated bins exist along the boundary of the polytope since any solution in the polytope not at the boundary will have a point above or to the right of it that is feasible.

We exhaustively enumerate all non-dominated bins. Once a complete set of non-dominated bins is created to represent all assignments of jobs to machines based on expected resource requirements, the machine assignment model decides, for each machine, which bin the machine should emulate. Thus, each machine will be mapped to a single bin, but multiple machines may emulate the same bin.

Algorithm 1 below generates all non-dominated bins. We define K^j, a set of job classes for machine configuration j containing each job class with positive δ^*_{jkl}, and a set b^j containing all possible bins. Given κ^j_i, a job belonging to the i^{th} class in K^j, and

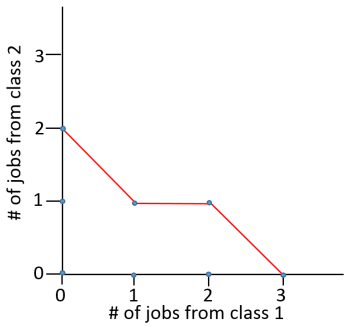


Fig. 5: Feasible bin configurations.

b_y^j, the y^{th} bin for machine configuration j, Algorithm 1 is performed for each machine configuration j. We make use of two functions not defined in the pseudo-code:

- sufficientResource(κ_i^j, b_y^j): Returns true if bin b_y^j has sufficient remaining resources for job κ_i^j.
- mostRecentAdd(b_y^j): Returns the job class that was most recently added to b_y^j.

Algorithm 1 Generation of all non-dominated bins

```
y ← 1
x ← 1
x* ← x
nextBin ← false
while x ≤ |K^j| do
  for i = x* → |K^j| do
    while sufficientResource(κ_i^j, b_y^j) do
      b_y^j ← b_y^j + κ_i^j
      nextBin ← true
    end while
  end for
  x* ← mostRecentAdd(b_y^j)
  if nextBin then
    b_{y+1}^j ← b_y^j − κ_{x*}^j
    y ← y + 1
  else
    b_y^j ← b_y^j − κ_{x*}^j
  end if
  if b_y^j == {} then
    x ← x + 1
    x* ← x
  else
    x* ← x* + 1
  end if
end while
```

Algorithm 1 is run for each machine configuration j. The algorithm starts by greedily filling the bin with jobs from a class. When no additional jobs from a class can be added, the algorithm will move to the next class of jobs and attempt to continue filling

the bin. Once no more jobs from any class are able to fit, the bin is non-dominated. The algorithm then backtracks by removing the last job added and tries to add jobs from other classes to fill the remaining unused resources. This continues until the algorithm has exhaustively searched for all non-dominated bins.

Since the algorithm performs an exhaustive search, solving for all non-dominated bins may take a significant amount of time. If we let L_k represent the maximum number of jobs of class k we can fit onto the machine of interest, then in the worst case, we must consider $\prod_{k \in K} L_k$ bins to account for every potential mix of jobs. We can improve the performance of the algorithm by ordering the classes in decreasing order of resource requirement. Of course, this is made difficult as there are multiple resources. One would have to ascertain the constraining resource on a machine and this may be dependent on which mix of jobs is used.[1]

Although the upper bound on the number of bins is very large, we are able to find all non-dominated bins quickly (i.e., within one second on an Intel Pentium 4 3.00 GHz CPU) because the algorithm only considers job classes with non-zero δ^*_{jkl} values. We generally see a small subset of job classes assigned to a machine configuration. Table 1 in Section 5 illustrates the size of K^j, the number of job classes with non-zero δ^*_{jkl} values for each configuration. When considering four job classes, all but one configuration has one or two job classes with non-zero δ^*_{jkl} values. When running Algorithm 1, the number of bins generated is in the thousands. Without the δ^*_{jkl} values from the allocation LP, we find that there can be on the order of millions of bins.

With the created bins, individual machines are then assigned to emulate one of the bins. To match the δ^*_{jkl} values for the corresponding machine configuration, we must find the contribution that each bin makes to the amount of resources allocated to each job class. We define N_{ijk} as the number of jobs from class k that are present in bin i of machine configuration j. Using the expected resource requirements, we can calculate the amount of resource l on machine j that is used for jobs of class k, denoted $\epsilon_{ijkl} = N_{ijk} r_{kl}$. The machine assignment LP is then

$$\max \quad \lambda \tag{6}$$

$$\text{s.t.} \sum_{j \in M} \Delta_{jkl} \mu_{jk} \geq \lambda \alpha_k r_{kl} \qquad k \in K, l \in R \tag{7}$$

$$\sum_{i \in B_j} \epsilon_{ijkl} x_{ij} = \Delta_{jkl} \quad j \in M, k \in K, l \in R \tag{8}$$

$$\sum_{i \in B_j} x_{ij} = n_j \qquad j \in M \tag{9}$$

$$x_{ij} \geq 0 \qquad j \in M, i \in B_j \tag{10}$$

[1] It may be beneficial to consider the dominant resource classification of Dominant Resource Fairness when creating such an ordering [8].

Decision Variables

Δ_{jkl}: Amount of resource l from machine configuration j that is devoted to job class k

x_{ij}: Total number of machines that are assigned to bins of type i in machine configuration j

Parameters

ϵ_{ijkl}: Amount of resource l of a machine in machine configuration j assigned to job class k if the machine emulates bin i.

B_j: Set of bins in machine configuration j

The machine assignment LP will map machines to bins with the goal of maximizing the arrival rate that maintains a stable system. Constraint (7) is the equivalent of constraint (2) of the allocation LP while accounting for discrete machines. The constraint ensures that a sufficient number of resources are available to maintain stability for each class of jobs. Constraint (8) determines the total amount of resource l from machine configuration j assigned to job class k to be the sum of each machine's resource contribution. In order to guarantee that each machine is mapped to a bin type, we use constraint (9). Finally, constraint (10) forces x_{ij} to be non-negative.

Although we wish each machine to be assigned exactly one bin type, such a model requires x_{ij} to be an integer variable and therefore the LP becomes an integer program (IP). We found experimentally that solving the IP model for this problem is not practical given a large set B_j. Therefore, we use an LP that allows the x_{ij} variables to take on fractional values. Upon obtaining a solution to the LP model, we must create an integer solution. The LP solution will have q_j machines of configuration j which are not properly assigned, where q_j can be calculated as

$$q_j = \sum_{i \in B_j} x_{ij} - \lfloor x_{ij} \rfloor.$$

We assign these machines by sorting all non-integer x_{ij} values by their fractionality $(x_{ij} - \lfloor x_{ij} \rfloor)$ in non-increasing order. Ties are broken arbitrarily if there are multiple bins with the same fractional contribution. We then begin to round the first q_j fractional x_{ij} values up and round all other x_{ij} values down for each configuration. This makes the problem tractable at the cost of optimality. However, given the scale of the problem that we study where a configuration can contain thousands of machines, the value of λ^* produced by the LP solution is typically very close to the value produced by the IP solution.

4.3 Dispatching Jobs

In the third and final stage of the scheduling algorithm, a two-level dispatching algorithm is used to assign arriving jobs to machines. The goal of the dispatching algorithm is to assign jobs to machines so that each machine emulates the bin it was assigned to in the second stage. In the first level of the dispatcher, a job is assigned to one of the $|M|$ machine configurations. The decision is guided by the Δ_{jkl} values to ensure that the correct proportion of jobs is assigned to each machine configuration. In the second level of the dispatcher, the job is placed on one of the machines in the configuration to

which it was assigned. At the first level, no state information is required to make decisions. However, in the second level, the dispatcher will make use of the exact resource requirements of a job as well as the states of machines to make a decision.

Deciding which machine configuration to assign a job to can be done by revisiting the total amounts of resources each configuration contributes to a job class. We can compare the Δ_{jkl} values to create a policy that will closely imitate the machine assignment solution. Given that each job class k has been devoted a total of $\sum_{j=1}^{|M|} \Delta_{jkl}$ resources of type l, a machine configuration j should serve a proportion

$$\rho_{jk} = \frac{\Delta_{jkl}}{\sum_{m=1}^{|M|} \Delta_{mkl}}$$

of the total jobs in class k. The value of ρ_{jk} can be calculated using the Δ_{jkl} values from any resource type l. To decide which configuration to assign an arriving job of class k, we use roulette wheel selection. We generate a uniformly distributed random variable, $u = [0, 1]$ and if

$$\sum_{m=0}^{j-1} \rho_{mk} \leq u < \sum_{m=0}^{j} \rho_{mk},$$

then the job is assigned to machine configuration j.

The second step will then dispatch the jobs directly onto machines. Given a solution x^*_{ij} from the machine assignment LP, we create an $n_j \times |K|$ matrix, $\mathbf{A}^j$, with element $\mathbf{A}^j_{ik}$ equal to 1 if the ith machine of j emulates a bin with one or more jobs of class k assigned. $\mathbf{A}^j$ indexes which machines can serve a job of class k.

The dispatcher will attempt to dispatch the job to a machine belonging to the configuration that was assigned from the first step. Machines are ordered arbitrarily and the dispatcher will search over the machines based on the ordering. The first machine found from those with $\mathbf{A}^j_{ik} = 1$ that has the available resources for the job will begin immediate service; this is a first-fit policy that is used by the dispatcher. In the case where no machines are available, the dispatcher will sort all machines, other than the machines belonging to the configuration that the job was initially assigned to, in non-decreasing order of processing times of the job needing assignment. The dispatcher will then search through these machines for immediate processing and if a machine exists with sufficient resources to immediately process the job, it will begin servicing the job. By allowing for the dispatcher to make assignments to machines with $\delta^*_{jkl} = 0$, we enable free resources to be used immediately. One could expect that a system that is not heavily loaded could benefit from the prompt service of jobs arriving to the system even though the assignment is inherently inefficient according to the allocation LP solution. If there exists no machine that can immediately process the job, the job will enter the smallest queue of the machines belonging to the configuration assigned in the first step with $\mathbf{A}^j_{ik} = 1$. Ties are broken randomly. Following such a dispatch policy attempts to schedule jobs immediately whenever possible with a bias towards placing jobs on bins which have been found to be efficient.

Jobs that are waiting in the queue follow a first-come, first-served (FCFS) order. An arriving job will have to wait until all jobs that arrived earlier have at least entered into service before it too can begin processing on the machine. This ensures that some level of fairness is maintained and prevents jobs with smaller resource requests from jumping forward in the queue and possibly starving jobs with large resource requests.

# of machines	Cores	Memory	$\lvert K^j \rvert$
6732	0.50	0.50	4
3863	0.50	0.25	2
1001	0.50	0.75	1
795	1.00	1.00	2
126	0.25	0.25	2
52	0.50	0.12	1
5	0.50	0.03	1
5	0.50	0.97	2
3	1.00	0.50	2
1	1.00	0.06	1

Table 1: Machine configuration details.

We use this ordering because it is often the default scheduling sequence used in practice for frameworks that run jobs in a data center environment, such as Hadoop [1].

By dispatching jobs using the proposed algorithm, the requirement of system state information is often reduced to a subset of machines that a job is potentially assigned to. Further, keeping track of the detailed schedule on each machine is not necessary for scheduling decisions since the only information used is whether a machine currently has sufficient resources, which job is next to be scheduled in the queue, and the size of the queue.

5 Experimental Results

We test our algorithm using cluster workload trace data provided by Google.[2] This data represents the workload for one of Google's compute clusters over the one month period of May 2011. The data captured in the trace workload provides information on the machines in the system as well as the jobs that arrive, their submission times, their resource requests, and their durations, which can be inferred from finding how long a job is active. However, because we calculate the processing time of a job based on the actual processing time realized in the workload traces, it is unknown to us how processing times may have differed if a job was processed on a different machine. Therefore, we assume that processing times are independent of machine configuration. In-depth analysis on the workload has been previously done [21]; we will be using the data as input for our scheduling algorithm to simulate its performance over the one month period.

Although the information provided is extensive, we limit what we use for our experiments. We do not consider failures of machines or jobs. Resubmitted jobs due to failures are considered to be new, unrelated jobs. Machine configurations change over time due to failures, the acquisition of new servers, or the decommissioning of old ones, but we will only use the initial set of machines and keep that constant over the whole month. Furthermore, system micro-architecture is provided for each machine. Some jobs are limited in which types of architecture they can be paired with and how they

[2] The data can be found at https://code.google.com/p/googleclusterdata/.

Job class	1	2	3	4
Avg. Time (h)	0.03	0.04	0.04	0.03
Avg. Cores	0.02	0.02	0.07	0.20
Avg. Mem.	0.01	0.03	0.03	0.06
Proportion of Total Jobs	0.23	0.46	0.30	0.01

Table 2: Job class details.

interact with these architectures, but we ignore this limitation for our scheduling experiments. It is easy to extend the LoTES algorithm to account for system architecture by setting $\mu_{jk} = 0$ whenever a job cannot be processed on a particular architecture. The focus for our work is on the efficient allocation of server resources to job classes and so we abstract the trace data to look only at resource requests and job durations.

The cluster of interest has 10 machine configurations (we use the configurations provided from the Google workload trace data) as presented in Table 1. Each configuration is defined strictly by its resource capacity and the number of identical machines with that resource profile. The resource capacities are normalized relative to the configuration with the most resources. Therefore, the job resource requests are also provided after being normalized to the maximum capacity of machines.

5.1 Class Clustering

The Google data does not define job classes and so in order for us to use the data to test our LoTES algorithm, we must first cluster jobs into classes. We follow Mishra et al. [18] by using k-means clustering to create job classes. We make use of Lloyd's algorithm [15] to create the different clusters. To limit the amount of information that LoTES is using in comparison to our benchmark algorithm, we only use the jobs from the first day to define the job classes for the month. These classes are assumed to be fixed for the entire month. Due to this assumption and because the Greedy policy does not use class information, any inaccuracies introduced by making clusters based on the first day will only make LoTES worse when we compare the two algorithms.

Clustering showed us that four classes were sufficient for representing most jobs. Increasing the number of classes led to less than 0.01% of jobs being allocated to the new classes and therefore, we use only four classes in our experiments. The different job classes are presented in Table 2.

5.2 Benchmark Algorithm: A Greedy Dispatch Policy

To illustrate the performance of the LoTES algorithm, we propose a Greedy dispatch policy as a benchmark. We chose to compare LoTES against the Greedy dispatch policy because it is a natural heuristic, which aims to quickly process jobs. The dispatch policy, like the LoTES algorithm, attempts to schedule jobs onto available machines immediately if possible. If a machine is found that can immediately process a job, the dispatch policy will make that assignment. In the case where no machines are available

for immediate processing, the policy will choose the machine with the shortest queue of waiting jobs. Ties are broken randomly. However, an assignment cannot be made if the requested resources of a job by themselves exceed the capacity of a machine. If a queue forms, jobs will be processed in FCFS order.

The Greedy dispatch policy is similar to the LoTES algorithm. The key difference in the two approaches is that the LoTES algorithm restricts the set of machines it considers to the set of machines found from solving the higher level allocation problems in the first two stages. By comparing against the Greedy policy, we can test how effective LoTES is and how useful the proper machine-job mapping is to system performance.

5.3 Implementation Challenges

In our experiments, we have not directly considered the time it takes for the scheduler to make dispatching decisions. As such, as soon as a job arrives to the system, the scheduler will immediately assign it to a machine. In practice, decisions are not instantaneous and depending on the amount of information needed by the scheduler and the complexity of the scheduling algorithm, the delay may be an issue. For every new job arrival, the scheduler requires state information of one or more machines. The state of the machine must provide the currently available resources and the size of the queue. As the system becomes busier, the scheduler may have to obtain state information for all machines in the data center. Thus, scaling may be problematic as the algorithms may have to potentially search over a very large number of machines. However, in heavily loaded systems where there are delays before a job can start processing, the scheduling overhead will not adversely affect system performance so long as the overhead is less than the waiting time delays. An additional issue may be present that could reduce performance of the scheduler at heavy loads. The scheduler creates additional load on the network connections within the data center itself. This may need to be accounted for if the network connections become sufficiently congested.

Note, however, that the dispatching overhead of LoTES is no worse than that of the Greedy policy. The LoTES algorithm benefits from the restricted set of machines that it considers when making scheduling decisions, but that does not guarantee that LoTES would not also end up obtaining state information on every machine when the system is heavily loaded. Therefore, a system manager for a very large data center must take into account the overhead required to obtain machine state information. There is work showing the benefits of only sampling state information from a limited set of machines to make a scheduling decision [10]. If the overhead of obtaining too much state information is problematic, we suggest that one can further limit the number of machines to be considered once a configuration has already been chosen. Such a scheduler could decide which configuration to send an arriving job to and then sample N machines randomly from the chosen configuration, where $N \in [1, n_j]$. Restricting the scheduler to only these N sampled machines, the scheduler can dispatch jobs following the same rules as LoTES. This allows the mappings from the offline stages of LoTES to still be used, but with substantially less overhead for the online portion.

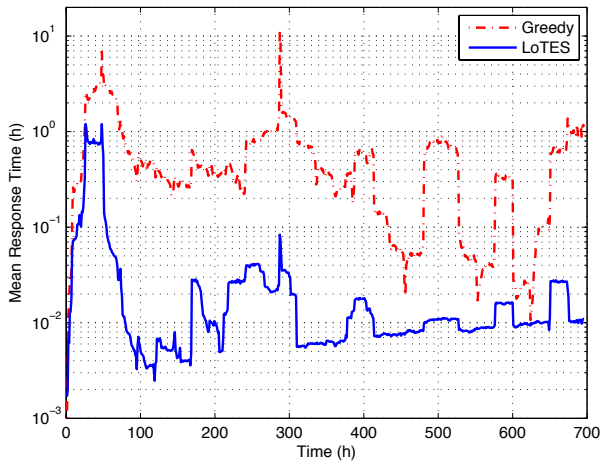


Fig. 6: Response Time Comparison.

5.4 Simulation Results: Workload Trace Data

We simulate the LoTES algorithm and the Greedy dispatch policy using the workload traces from Google. We created an event-based simulator in C++ to emulate a data center with the workload data used as input to our system. The LP models are solved using IBM ILOG CPLEX 12.5. We run our tests on an Intel Pentium 4 CPU 3.00 GHz, 1 GB of main memory, running Red Hat 3.4-6-3. Because the LP models are solved offline prior to the arrival of jobs, the solutions to the first two stages are not time-sensitive. Regardless, the total time required to obtain solutions to both LP models and to generate bins requires less than one minute of computation time. This level of computational effort means that it is realistic to re-solve these two stages periodically, perhaps multiple times a day, if the job classes or machine configurations change due, for example, to non-stationary distributions. We leave this for future work.

Figure 6 presents the performance of the system over the one month period. The graph provides the mean response time of jobs over every one-hour long interval. We include a job's response time in the mean response time calculation in the interval in which the job begins processing. We see that the LoTES algorithm greatly outperforms the Greedy policy. On average, the Greedy policy has response times an order of magnitude longer (15-20 minutes) than the response times of the LoTES algorithm (1-2 minutes). The difference on average shows the strong performance of LoTES, however, a more interesting result is the performance difference when the system becomes heavily loaded. During the one-month period, the Greedy policy has two large spikes in response times that occur where jobs must wait for close to 10 hours around the 70 hour and 280 hour time points. During both occurrences, the LoTES algorithm produces schedules with response times on the order of 1 hour long in the first occurrence, and 10 minutes in the second occurrence.

Figures 7 and 8 provide the core and memory utilization of the machines over time. At the end of each hour, we record the instantaneous utilization of resources over all machines and graph those results. We observe that curves are typically very close to

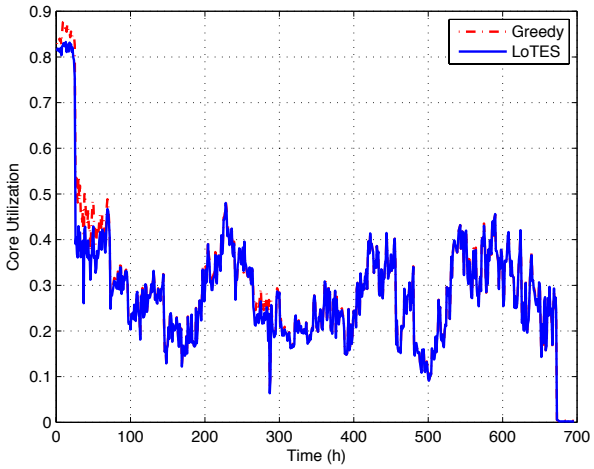


Fig. 7: Processing Core Utilization Comparison.

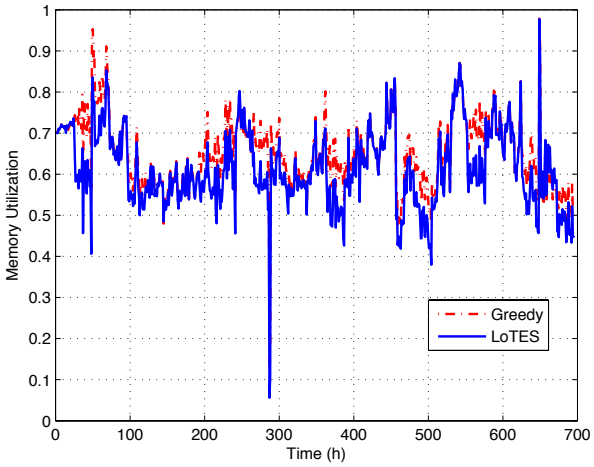


Fig. 8: Memory Utilization Comparison.

each other, but at certain time points, the Greedy policy has higher utilization. We believe the increased utilization is due to the build up of jobs in queue for the Greedy policy. With a large queue, as soon as jobs are completed, the available space is filled again with a waiting job. Since LoTES is doing a better job of increasing throughput in the short term through efficient allocation, queues do not form as often. However, over the long term, as this is an open system and we assume that no jobs are abandoned, the long-run throughput of both systems will be the same and therefore long-run resource utilizations are also the same. As a result, LoTES is doing a better job at smoothing the utilization curves.

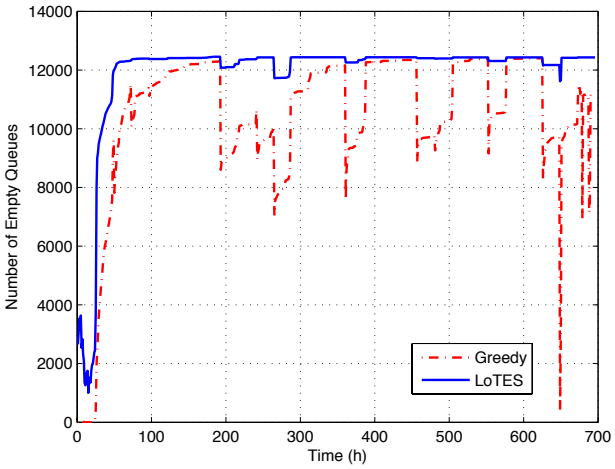


Fig. 9: Empty Queue Comparison.

Figure 9 plots the number of empty queues for both schedulers. Of the 12,583 machines present in the system, we graph the number of machines that have an empty queue during each hour of the simulation. Although the queue length at a machine may change during the hour long period, we only record the state of the queue at the end of the hour. We see that very quickly, the LoTES algorithm is able to keep the queues of all machines relatively empty. However, the Greedy policy often has a large number of machines with a queue. This queue formation leads to the higher resource utilization and the increased response times.

6 Conclusion and Future Work

In this work, we developed a scheduling algorithm that creates a mapping between jobs and machines based on their resource profiles to improve the response time of the system. The algorithm consists of three stages where a fluid representation and queueing model are used at the first stage to fractionally allocate job classes to machine configurations. The second stage then solves a combinatorial problem to generate possible assignments of jobs on machines. An LP model is developed to maximize system capacity by choosing which of the generated sets of jobs that each machine should aim to emulate. The final stage is an online dispatching policy that uses the solution from the second stage to decide on the machine to assign to each incoming job. Our algorithm was tested on Google workload trace data and was found to reduce response times by up to an order of magnitude when compared to a benchmark dispatch policy. This improvement in performance is also computationally cheaper than the benchmark policy during the online scheduling phase since the proposed algorithm often requires state information for fewer machines when making assignment decisions.

The data center scheduling problem is very rich from the scheduling perspective and can be expanded in many different ways. Our algorithm assumes stationary arrivals over the entire duration of the scheduling horizon. However, the real system is not

stationary and the arrival rate of each job class may vary over time. Furthermore, the actual job classes themselves may change over time as resource requirements may not always be clustered in the same manner. As noted above, the offline phase is sufficiently fast (about one minute of CPU time) that it could be run multiple times per day as the system and load characteristics change. Beyond this we plan to extend the LoTES algorithm to more accurately represent dynamic job classes. This would allow the LoTES algorithm to learn and predict the expected job population and make scheduling decisions with these predictions in mind. Not only do we wish to be able to adjust our algorithm to a changing environment, but we also wish to extend our algorithm to be able to more intelligently handle situations where there is high variance in the mix of job classes in the environment. The high variance will lead to system realizations that differ significantly from the bins created in the second stage of the LoTES algorithm.

We also plan to study the effects of errors in job resource requests. We used the amount of requested resources of a job as the amount of resource used over the entire duration of the job. In reality, most jobs may end up using less or more resources than requested due to the fact that users may under or overestimate their resource requirements. In addition, the utilization of a resource may change over the duration of the job itself. We plan to incorporate these uncertainties regarding resource usage to improve system utilization. This adds difficulty to the problem because instead of creating a schedule where we know the exact amount of requested resources once a job arrives, we only have an estimate of the requests and must ensure that a machine is not underutilized or oversubscribed.

Acknowledgment

This work was made possible in part due to a Google Research Award and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Apache Hadoop. http://hadoop.apache.org
2. Al-Azzoni, I., Down, D.G.: Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems. IEEE Transactions on Parallel and Distributed Systems **19**(12), 1671–1682 (2008)
3. Andradóttir, S., Ayhan, H., Down, D.G.: Dynamic server allocation for queueing networks with flexible servers. Operations Research **51**(6), 952–968 (2003)
4. Berral, J.L., Goiri, Í., Nou, R., Julià, F., Guitart, J., Gavaldà, R., Torres, J.: Towards energy-aware scheduling in data centers using machine learning. In: Proceedings of the 1st International Conference on energy-Efficient Computing and Networking, pp. 215–224. ACM (2010)
5. Chang, R.S., Lin, C.Y., Lin, C.F.: An adaptive scoring job scheduling algorithm for grid computing. Information Sciences **207**, 79–89 (2012)
6. Dai, J.G., Meyn, S.P.: Stability and convergence of moments for multiclass queueing networks via fluid limit models. IEEE Transactions on Automatic Control **40**(11), 1889–1904 (1995)
7. Gandhi, A., Harchol-Balter, M., Kozuch, M.A.: Are sleep states effective in data centers? In: International Green Computing Conference (IGCC), pp. 1–10. IEEE (2012)
8. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: Fair allocation of multiple resource types. In: NSDI, vol. 11, pp. 24–24 (2011)

9. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. In: Proceedings of the 2014 ACM conference on SIGCOMM, pp. 455–466. ACM (2014)
10. He, Y.T., Down, D.G.: Limited choice and locality considerations for load balancing. Performance Evaluation **65**(9), 670–687 (2008)
11. Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., Goldberg, A.: Quincy: fair scheduling for distributed computing clusters. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 261–276. ACM (2009)
12. Kim, J.K., Shivle, S., Siegel, H.J., Maciejewski, A.A., Braun, T.D., Schneider, M., Tideman, S., Chitta, R., Dilmaghani, R.B., Joshi, R., et al.: Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. Journal of Parallel and Distributed Computing **67**(2), 154–169 (2007)
13. Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., Nguyen, T.D.: Reducing electricity cost through virtual machine placement in high performance computing clouds. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 22. ACM (2011)
14. Liu, Z., Lin, M., Wierman, A., Low, S.H., Andrew, L.L.: Greening geographical load balancing. In: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 233–244. ACM (2011)
15. Lloyd, S.: Least squares quantization in PCM. IEEE Transactions on Information Theory **28**(2), 129–137 (1982)
16. Maguluri, S.T., Srikant, R., Ying, L.: Heavy traffic optimal resource allocation algorithms for cloud computing clusters. In: Proceedings of the 24th International Teletraffic Congress, p. 25. International Teletraffic Congress (2012)
17. Maguluri, S.T., Srikant, R., Ying, L.: Stochastic models of load balancing and scheduling in cloud computing clusters. In: Proceedings IEEE INFOCOM, pp. 702–710. IEEE (2012)
18. Mishra, A., Hellerstein, J., Cirne, W., Das, C.: Towards characterizing cloud backend workloads: insights from Google compute clusters. ACM SIGMETRICS Performance Evaluation Review **37**(4), 34–41 (2010)
19. Ousterhout, K., Wendell, P., Zaharia, M., Stoica, I.: Sparrow: distributed, low latency scheduling. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 69–84. ACM (2013)
20. Rasooli, A., Down, D.G.: COSHH: A classification and optimization based scheduler for heterogeneous hadoop systems. Future Generation Computer Systems **36**, 1–15 (2014)
21. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing, pp. 1–13. ACM (2012)
22. Tang, Q., Gupta, S.K., Varsamopoulos, G.: Thermal-aware task scheduling for data centers through minimizing heat recirculation. In: IEEE International Conference on Cluster Computing, pp. 129–138. IEEE (2007)
23. Wang, L., Von Laszewski, G., Dayal, J., He, X., Younge, A.J., Furlani, T.R.: Towards thermal aware workload scheduling in a data center. In: Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on, pp. 116–122. IEEE (2009)
24. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European conference on Computer systems, pp. 265–278. ACM (2010)

MISTA 2015

Examination timetabling at Technische Universität Berlin

Mirjana Lach · Gerald Lach · Erhard Zorn

Abstract We present a new integer programming based model for the examination timetabling problem at Technische Universität Berlin. This problem has to be solved to assign written examinations to rooms and timeslots at universities. Our model not only respects students who have to take several exams in one examination period, but it also allows for the preparation time of the students between consecutive exams. The new model has been successfully used at Technische Universität Berlin with more than 32,000 students.

1 Introduction

The examination timetabling problem (ETP) describes the assignment of exams at a university to rooms and timeslots without conflicts for students who have to take several exams in one examination period. Usually the exams take place in a limited period after the lecture period or before the next lecture period. The exam timetable may be subject to additional constraints varying among universities. We describe how to generate an exam timetable also respecting time between different exams of each student, enough allowing to prepare for the next exam. Generating satisfactory exam timetables at large universities is a sophisticated task; exams manually planned may easily result in conflicts. In the last decade, at many European universities the number of exams has increased significantly. This is, among other reasons, due to the Bologna process at European universities, a consequence of an increasing number of courses of studies and courses with final written examinations. At large universities like Technische Universität Berlin (TU Berlin) with more than 32,000 students, this task cannot

Mirjana Lach
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: mirjana.lach@math.tu-berlin.de

Gerald Lach
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: gerald.lach@math.tu-berlin.de

Erhard Zorn
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: erhard@math.tu-berlin.de

be done manually. We present an integer programming (IP) based approach which has been successfully used at TU Berlin since 2010.

2 Problem and solution outline

The *Examination Timetabling* problem is a special case of the timetabling problem discussed in [4] and hence $\mathcal{NP}$-hard. Consequently, it cannot be expected that an algorithm can be found which solves the problem in polynomial time—unless $\mathcal{P}$ equals $\mathcal{NP}$.

In the literature, different techniques are used dealing with this problem, amongst others heuristics ([5], [12], [3], [11]) and IP based techniques ([10], [2], [6]). Due to different requirements on the generated timetable, not all approaches are directly comparable to the approach introduced in this paper, regarding the quality of the solutions or the size of the problems.

As discussed in [8], linear integer programming can be successfully applied to the *University Course Timetabling* problem; the authors used problem instances representative for large universities like TU Berlin, and which fulfilled typical requirements.

Because of the different requirements of both problems, the solution technique for the *University Course Timetabling* problem from [8] cannot be applied directly to the *Examination Timetabling* problem. However, the idea of [8] was used to decompose the whole problem into two subproblems and to solve them separately. The solution procedure of the first subproblem affects the second one in a certain manner in order to obtain a conflict-free solution of the first subproblem. Consequently, the existence of a conflict-free solution of the second subproblem is guaranteed and can be found with another solution procedure. Finally, the solutions of the two subproblems are merged into a complete solution of the original problem.

3 Examination timetabling at TU Berlin

The requirements for exam timetables vary significantly among universities. Therefore, we will describe the requirements at TU Berlin in detail.

Timeslots

Timeslots identify the time frames of a day when an exam may take place. They are disjoint and defined by a starting time and an end time. A typical example of timeslots used at TU Berlin is a Monday 09:00–12:00 a.m. within a period of three after the end of the lecture period. An exam may be assigned to a fixed timeslot whereby the assignment of the other exams has to comply with this. This is a rare exemption that requires a substantive reason. Therefore, this can only be done by an administrator. A lecturer who offers an exam may indicate which timeslots during the exam period come into consideration. If an exam has to take place in more than one room, they will be assigned at the *same* timeslot—for obvious reasons.

Rooms

The capacity of a room characterizes the number of seats which can be used for the participants of an exam. In order to impede the communication among the students during an exam, a certain physical distance between the attendees has

to be considered. Therefore, the examiner selects a 'filling factor', e.g. 1/6 (which corresponds to every second row, every third seat in a row for the examinees). The examiner also specifies the number of rooms he is able to monitor (at the same timeslot) with his staff and the number of students he is expecting to attend his exam. In each room, there can only take place one exam at the same timeslot. Furthermore, the examiner may request for rooms with different priorities. The preferred rooms have to fulfill the data specified by the examiner (capacity and number).

Time for preparation

For every exam a particular time period (in days) for preparation may be defined. This ensures that a student can prepare his exam in sufficient time without the need for preparing more than one exam at the same time. For a very tight exam period this may at least ensure that there is a minimal time period (e.g. one day) between two exams of the same student to allow for recapitulation.

Conflict matrix

The conflict matrix discloses which pairs of exams belong to a curriculum of one term (1st term, 2nd term etc.) of a course of studies and may therefore not take place at the same timeslot. A temporal distance may be observed between them respectively (temporal conflict). The temporal distance complies with the time needed for preparing the subsequent exam of the concerning two exams. If the time for preparing for an exam is zero, then it has to be prevented that the concerning exam is taking place at the same day as another exam which is in conflict with the former one.

Curricula inclose the most important information to create a conflict-free exam timetable. For most of the obligatory courses there is a determined term (1st term, 2nd term etc.) in which students of a specific course of studies are expected to take the course/exam. For students failing there are often additional exams offered in the next exam period (before the next lecture period). Therefore, the curricula inclose the most reliable information for the conflict matrix.

But there are also courses students may *choose* to attend; e.g. they have to choose two out of five courses in a special field, or students may select courses totaling 10 credits. It would be absurd to represent the exams of all these courses in a conflict matrix: In most cases students have a free choice in which term they want to attend the above mentioned courses. Therefore, they would have to be in conflict with every other course! Fortunately, most of these 'free' courses will be attended by students in their 2nd or 3rd year when they successfully passed most of the obligatory courses. And often these are courses with few students that additionally do not offer written exams but oral exams that are individually arranged. Yet, there are 'free' courses that are attended by many students of a specific course of studies in a specific term according to experience of the lecturers. Therefore, a conflict may be added to the conflict matrix if it is well-known that two exams will be taken by a significant number of students. These exams are not obliged to belong to one of the curricula of the current term of the concerning students. Such additional conflicts may also be used if it is well-known that a considerable number of students will fail a course and repeat it the next term.

4 Solution model for the ETP at TU Berlin

In what follows, we will describe how we developed a procedure to solve the ETP. Our solution was implemented and has already been successfully applied to real data of TU Berlin. To the best of our knowledge, it is the first exact program which may solve instances of the ETP for large universities like TU Berlin [9].

Decomposition model

The decomposition model divides the ETP into two subproblems. The first subproblem consists of the *date assignment*, i.e. every exam is assigned to a timeslot. The second subproblem consists of the *room assignment*, i.e. every exam, which was already assigned to a timeslot, is assigned to the corresponding rooms. More precisely, during the date assignment every exam is assigned to a timeslot without the need for two exams of being in a temporal conflict. In addition, it is assured that after a date assignment a room assignment is possible without a room being occupied by more than one exam at the same time, and every exam which was assigned to a timeslot is assigned to enough rooms. To put it another way, the existence of a conflict-free room assignment is guaranteed after the date assignment. This is achieved by using an exponential number of constraints which can be separated in polynomial time. Consequently, the two components are interdependent.

Let K be the set of exams, TS the set of timeslots, and R the set of rooms. For each $r \in R$ let $K(r)$ be the set of exams for which r is an appropriate room. For each $k \in K$ let $R(k)$ be set of rooms which are adequate for k, and $TS(k)$ the set of timeslots at which exam k may take place. Additionally, we define $NR(k)$ as the number of rooms which are required for exam $k \in K$.

Time conflicts are represented by a conflict graph $G_{\text{conf}} = (V_{\text{conf}}, E_{conf})$: A vertex $v_{k,ts} \in V_{\text{conf}}$ represents a possible timeslots ts for the exam of course k. An edge $v_{k_1,ts_1}v_{k_2,ts_2} \in E_{\text{conf}}$ exists if and only if either exam k_1 must not take place at timeslot ts_1 or exam k_2 must not take place at timeslot ts_2.

The room conflicts are also modeled using a graph $H = (V_{\text{net}} \cup R_{\text{net}}, E_{\text{net}}, s, t, u)$: Again, a vertex $v_{k,ts} \in V_{\text{net}}$ represents a possible timeslot ts for the exam of course k. Furthermore, vertex $r_{r,ts} \in R_{\text{net}}$ represents a timeslot ts where room r can be occupied by an exam; s is the sink and t the target of the network. Based on the vertices the edges

$$E_{\text{net}} = \{(s,v) : v \in V_{\text{net}}\} \cup \{(r,t) : r \in R_{\text{net}}\} \cup \{(v_{k,ts}, r_{r,ts}) : r \in R(k)\} \tag{1}$$

and the capacity

$$u : E_{\text{net}} \to \mathbb{N},\ \ u(e) = \begin{cases} NR(k) & \text{if } e \in \{(s,v) : v \in V_{\text{net}}\} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

are defined. Furthermore, for every $A \subset V_{\text{net}}$ we define $\hat{u}_A$.

$$\hat{u}_A : E_{\text{net}} \to \mathbb{N},\ \ \hat{u}_A(e) = \begin{cases} 0 & \text{if } e \in \{(s,v) : v \in V_{\text{net}} \backslash A\} \\ u(e) & \text{otherwise} \end{cases} \tag{3}$$

Based on these definitions, we deduce the function

$$\text{maxF} : 2^{V_{\text{net}}} \to \mathbb{N},\ \ \text{maxF}(A) \mapsto \text{ Max. Flow on } \hat{H}_A = (V_{\text{net}} \cup R_{\text{net}}, E_{\text{net}}, s, t, \hat{u}_A) \tag{4}$$

First subproblem of the decomposition model

Before we are going to present the first subproblem of the decomposition model, two lemmata demonstrating the connection between these two subproblems are presented.

Lemma 1 *Let $ts \in TS$, $A \subset K$, and $A_{net} = \{v_{k,ts} : k \in A\} \subset V_{net}$. Then there exists a* conflict-free room assignment *for A at timeslot ts if and only if:*

$$\text{maxF}(A_{net}) = \sum_{k \in A} NR(k)$$

Lemma 2 *Let $ts \in TS$, $A \subset K$, $A_{net} = \{v_{k,ts} : k \in K\} \subset V_{net}$, and $x^* \in \mathbb{R}^{|A_{net}|}$. Then there exists a polynomial time separation algorithm that detects an $B \subset A$, $B_{net} = \{v_{b,ts} : b \in B\}$ such that*

$$\text{maxF}(B_{net}) < \sum_{k \in B} NR(k) \cdot x^*_{k,ts}$$

or proofs that for all $B \subset A$, $B_{net} = \{v_{b,ts} : b \in B\}$ holds:

$$\text{maxF}(B_{net}) \geq \sum_{k \in B} NR(k) \cdot x^*_{k,ts}$$

The proofs and a description of the separation algorithm can be found in [9]. Keeping these lemmata in mind, we are able to define the IP model that assigns conflict-free dates to as many exams as possible with respect to the room resources.

objective function

$$\max \sum_{k \in K, ts \in TS} y_{k,ts}$$

constraints

$$\begin{aligned} y_{k1,ts1} + y_{k2,ts2} &\leq 1 && (v_{k_1,ts_1} v_{k_2,ts_2}) \in E_{\text{conf}} \\ \sum_{v_{k,ts} \in K_{\text{net}}} NR(k) \cdot y_{k,ts} &\leq \text{maxF}(K_{\text{net}}) && K_{\text{net}} \subset V_{\text{net}} \\ y_{k,ts} &\in \{0,1\} && v_{k,ts} \in V_{\text{conf}} \end{aligned}$$

Second subproblem of the decomposition model

Assuming having calculated the optimal $y_{k,ts} \in \{0,1\}$ values in the first subproblem, the second subproblem can be modeled similarly to a *min-cost-flow* problem. We suggest that the $y_{k,ts}$ are not variables but constants for the second part of the decomposition model. In the objective function the global sum of room priorities ($\text{prio}(r,k)$) which have been specified by the examiner is minimized.

objective function

$$\min \sum_{k \in K, r \in R, ts \in TS} x_{k,r,ts} \cdot \text{prio}(k,r)$$

constraints

$$\sum_{r \in R(k)} x_{k,r,ts} = NR(k) \cdot y_{k,ts} \qquad k \in K, ts \in TS$$

$$\sum_{k \in K(r)} x_{k,r,ts} \leq 1 \qquad r \in R, ts \in TS$$

$$x_{k,r,ts} \in \{0,1\} \qquad k \in K, r \in R, ts \in TS$$

5 Results

The Software was implemented using *SCIP* [1] and *CPLEX* [7]. The first timetable for 116 exams was generated in Spring 2010 using the approach represented where no conflicting exams overlap assuring that two courses of one curriculum never take place on consecutive days. Courses from six of the seven faculties of TU Berlin participated, that led to more than 7,400 students involved and more than 26,000 exam participants. In particular, students of the first four semesters benefited from the better organization of their exams. The running time for solving the problem instances is less than one minute for all instances tested. In Table 1 the increasing number of participating exams is shown. The room utilization per room denotes the average percentage of occupied seats of a room during an exam. The room utilization per time period denotes the occupied seats during all exams in a specific time period of two weeks after and two weeks before the next lecture period. The room utilization is relevant, e.g., for the steering committee of a university in order to save costs. By using as few rooms as possible for exams, maintenance and external events may be organized in a better way, and the number of external rooms to be rented may be reduced—if necessary at all. A minimum time-lag to the exam day preferred by the examiner is important for the acceptance of the new exam timetabling process. The examiners have to be convinced that an algorithm can generate a better exam timetable—better, e.g., than a timetable generated by a first come, first served principle. The time-lag to a preferred day of a lecturer is less than one day in average. Nevertheless, we found that the procedure and the algorithm for generating an exam timetable can still be improved. This will be discussed in an upcoming paper.

Table 1 Number of exams, participants, room utilization, and time-lag to preferred day

semester	exams	participants	room utilization per room	time-lag to preferred day	room utilization per time period
Spring 2011	176	30,840	82%	0.84	52%
Fall 2011	193	36,875	80%	1.49	56%
Spring 2012	194	35,331	80%	0.76	53%
Fall 2012	204	39,326	81%	0.87	66%
Spring 2013	189	34,946	82%	0.27	49%

References

1. Achterberg, T., Berthold, T., Heinz, S., Koch, T., Wolter, K.: SCIP – Solving Constraint Integer Programs, documentation (2009). http://scip.zib.de
2. Al-Yakoob, S., Sherali, H., Al-Jazzaf, M.: A mixed-integer mathematical modeling approach to exam timetabling. Computational Management Science **7**(1), 19–46 (2010)
3. Bilgin, B., Özcan, E., Korkmaz, E.E.: An experimental study on hyper-heuristics and exam timetabling. In: Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling, pp. 123–140 (2006)
4. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Selected papers from the First International Conference on Practice and Theory of Automated Timetabling, pp. 283–295. Springer-Verlag, London, UK (1996)
5. Eley, M.: Ant algorithms for the exam timetabling problem. In: PATAT'06: Proceedings of the 6th international conference on Practice and theory of automated timetabling VI, pp. 364–382. Springer-Verlag, Berlin, Heidelberg (2007)
6. Fonseca, G., Santos, H.: A new integer linear programming formulation to the examination timetabling problem. In: MISTA'13: Proceedings of the 6th Multidisciplinary International Scheduling Conference, pp. 345–355 (2013)
7. IBM: CPLEX Optimizer (2009). URL http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html. Last visited 20.05.2015
8. Lach, G., Lübbecke, M.: Optimal university course timetables and the partial transversal polytope. In: C. McGeoch (ed.) 7th International Workshop on Efficient and Experimental Algorithms (WEA08), *LNCS*, vol. 5038, pp. 235–248. Springer, Berlin (2008)
9. Lach, M.: Ein Verfahren zur Optimierung der Klausurterminplanng an der TU Berlin. Master's thesis, Technische Universität Berlin, Institut für Mathematik (2008). In German
10. MirHassani, S.A.: Improving paper spread in examination timetables using integer programming. Applied Mathematics and Computation **179**(2), 702–706 (2006)
11. Müller, T.: Real-life examination timetabling. In: MISTA'13: Proceedings of the 6th Multidisciplinary International Scheduling Conference, pp. 248–267 (2013)
12. Zampieri, A., Schaerf, A.: Modelling and solving the italian examination timetabling problem using tabu search (2006)

MISTA 2015

Improving Upper Bounds in High School Timetabling by Matheuristics

George H.G. Fonseca · Haroldo G. Santos · Eduardo G. Carrano

Abstract The High School Timetabling Problem requires assignment of timeslots and resources to events, respecting given constraints. The most common approaches for this type of timetabling problems are meta-heuristics. This work presents a matheuristic approach combining a Variable Neighbourhood Search algorithm with mathematical programming-based neighbourhoods for high school timetabling. The computational experiments on well-known benchmark instances demonstrate the success of the proposed matheuristic approach, improving the 14 out of 17 best known solutions from the XHSTT-2014 archive.

Keywords Matheuristics · High School Timetabling · Third International Timetabling Competition

1 Introduction

The High School Timetabling Problem is faced by many educational institutions around the world. It consists in assigning timeslots and resources to events, respecting several constraints. Generally, this assignment is repeated weekly until the end of the semester. Some common constraints present in this problem are to respect the availability of teachers, to respect a limit of lessons of the same subject taught in a day and to avoid idle times between activities.

The Third International Timetabling Competition (ITC2011) [14] stimulated the development of several approaches to solve this problem. The competition considered

George H. G. Fonseca and Eduardo G. Carrano
Graduate Program in Electrical Engineering
Federal University of Minas Gerais
Av. Antônio Carlos 6627, 31270-901
Belo Horizonte, MG, Brazil
E-mail: george@decsi.ufop.br, egcarrano@ufmg.br

Haroldo G. Santos
Department of Computing
Federal University of Ouro Preto
St. Diogo de Vasconcelos 328, 35400-000
Ouro Preto, MG, Brazil
E-mail: haroldo@iceb.ufop.br

the eXtended Markup Language for High School TimeTabling (XHSTT) format [15], which can specify several features of scheduling problems. More than 40 real world datasets, from 12 different countries, are now available in this format to evaluate the performance of algorithms for high school/university timetabling. In the competition, meta-heuristic approaches achieved the best results, as component of GOAL solver [6]. Other finalists of ITC2011 were Lectio, HySTT and HFT. Lectio ranked second with a Adaptive Large Neighbourhood Search (ALNS) approach [19]. In third place, HySTT developed a Hyper-heuristic approach [8]. HFT ranked fourth with an Evolutionary Algorithm [16].

More recently, GOAL team released a new solver based on the Variable Neighbourhood Search algorithm [6], Kingston [10] developed a solver based on his library for handling XHSTT instances and HySTT team worked towards an improved version of their Hyper-heuristic approach [1]. In mathematical programming field, Kristiansen *et al.* [11] developed the first Integer Programming formulation for XHSTT timetabling problems.

The integration of meta-heuristics and mathematical programming approaches, namely matheuristics, is a growing field in operations research. For example, Toffolo *et al.* [17] presented a matheuristic for Nurse Rostering, Pirkwieser *et al.* [12] presented a matheuristic for Vehicle Routing and Merz *et al.* [4] presented a matheuristic for Flow Shop Scheduling. Sorensen and Stidsen [18] presents some preliminary results in matheuristics for XHSTT timetabling problems. However, no problem-specific neighbourhood was presented in their work. In this way, the main objective of this work is present some new matheuristic neighbourhoods for High School Timetabling along with new results considering these neighbourhoods.

This paper is organized as follows. Section 2 presents a brief description of XHSTT format. In Section 3 the proposed matheuristic for XHSTT and its neighbourhoods are presented. Section 4 presents the computational experiments and discussion of the results. Finally, in Section 5 the concluding remarks are presented.

2 XHSTT Format

An XHSTT instance is composed of four entities:

- **Times**. Contains the possible timeslots for allocations. These timeslots may also be grouped into *TimeGroups*;
- **Resources**. Contains the available resources for assignments. Each resource has a specific *ResourceType*. Resources are also commonly grouped into *ResourceGroups*;
- **Events**. Represents the events to be scheduled. Each event has a duration meaning the amount of times in which it has to be scheduled and a demand for a set of resources. Times and resources may be pre-assigned to the events. When these entities are not pre-assigned, it is expected that a solver makes this assignment. Optionally events may have a workload which is considered to its assigned resources. Events are also commonly grouped into *EventGroups*;
- **Constraints**. Represents the set of constraints that should be respected in a solution for an instance of this problem. Table 1 presents the 16 constraint types available in this format. Each constraint may be either hard or soft. A solution with any hard constraint violation is called infeasible while the soft constraints measure the quality of a solution. Smaller values indicate a better solution. Each

Constraint	Description
Assign Resource	Event resource should be assigned a resource
Assign Time	Event should be assigned a time
Split Events	Event should split into a constrained number of sub-events
Distribute Split Events	Event should split into sub-events of constrained durations
Prefer Resources	Event resource assignment should come from resource group
Prefer Times	Event time assignment should come from time group
Avoid Split Assignments	Set of event resources should be assigned the same resource
Spread Events	Set of events should be spread evenly through the cycle
Link Events	Set of events should be assigned the same time
Order Events	Set of events should be ordered
Avoid Clashes	Resource's timetable should not have clashes
Avoid Unavailable Times	Resource should not be busy at unavailable times
Limit Idle Times	Resource's timetable should not have idle times
Cluster Busy Times	Resource should be busy on a limited number of days
Limit Busy Times	Resource should be busy a limited number of times each day
Limit Workload	Resource's total workload should be limited

Table 1 Different constraint types in the XHSTT format [13].

constraint also has a cost meaning the penalty for a single violation and a cost function dictating how violations will be penalized in the objective function. A deeper description of this format can be found in Post *et al.* [15] and Kingston [9].

3 Matheuristics

Matheuristics are optimization algorithms made by the interoperation of meta-heuristics and mathematical programming (MP) techniques [2]. In the proposed approach the meta-heuristic works at a master level, controlling low level local search procedures. These local search procedures consists of Mixed Integer Programming (MIP) models having a subset of variables fixed to their current values in the incumbent solution and the remaining variables freed to be optimized by a MIP solver.

The main procedure is similar to the proposed by Sorensen and Stidsen [18]. However, the focus of our work is the matheuristic neighbourhoods for this problem. Considering that X represents the set of decision variables, s represents the current solution and $n(.)$ a neighbourhood function, the matheuristic framework is presented in Algorithm 1.

Algorithm 1: Matheuristic

Input: XHSTT instance P.
Output: Best solution s found.
$s \leftarrow$ MIP solution considering only hard constraints;
while $elapsedTime < timeout$ **do**
  obtain variables $v \leftarrow n(s)$;
  fix variables $X - v$ to their current value;
  invoke MIP solver with a short time limit;
  unfix all variables;
return s;

In Line 1 the Algorithm starts with the MIP model searching for a feasible solution (i.e. one that does not violate any hard constraint). In Line 3, the algorithm selects a set of variables regarding a specific neighbourhood. In Line 4 all variables, except the selected ones have their values fixed. In Line 5 the MIP solver is invoked with a small time limit (see Table 2). After that, all variables are unfixed for the next iteration of the algorithm.

The following subsections present the considered MIP model and four ways (neighbourhoods) to select which variables will be unfixed in the MIP local search procedure.

3.1 MIP Model

The considered MIP Model was recently proposed by Kristiansen *et al.* [11]. This model is able to handle any XHSTT instance. For sake of brevity, only the input data and basic variables of the formulation given by Kristiansen *et al.* [11] will be described in this paper.

The input data for this formulation is a set of times T, a set of resources R, a set of events E and a set of constraints C. An event $e \in E$ has a duration $d_e \in \mathbb{N}$, and a number of event resources, each one denoted as $er \in e$. An event resource defines the requirement of the assignment of a resource to the event, and this resource can be specified to be pre-assigned. If the resource is not pre-assigned, a resource of proper type must be assigned. Furthermore an event resource er can undertake a specific role$_{er}$, which is used to link the event resource to certain constraints. Generally an event has to be split into sub-events whose sum of durations matches the duration of the original event. This formulation creates the 'full set' of sub-events se with different lengths, such that all possible combinations of sub-events for a given event can be handled.

Variable $x_{se,t,er,r} \in \{0,1\}$ takes value 1 if sub-event $se \in SE$ has been assigned time $t \in T$ as starting time and resource $r \in er$ is assigned to event resource $er \in se$, and 0 otherwise. Binary variable $y_{se,t}$ takes value 1 if sub-event $se \in SE$ has been assigned time $t \in T$. To reduce the amount of non-zeros in the MIP model, two auxiliary variables are introduced which inherits their values from $x_{se,t,er,r}$. The variable $v_{t,r} \in \mathbb{N}_0$ denotes the number of times resource $r \in R$ is used in time $t \in T$. Finally, variable $w_{se,er,r} \in \{0,1\}$ takes value 1 if sub-event $se \in SE$ is assigned resource $r \in R$ for event resource $er \in se$, and 0 otherwise.

Upon these input data and variables the constraints in this problem are modelled. Each point of application of each constraint has also a slack variable indicating a penalty to be considered in the objective function. Eventually some constraint specific variables are also required. The complete formulation is presented in Kristiansen *et al.* [11].

3.2 Neighbourhood

3.2.1 Events Neighbourhood

In this neighbourhood, a randomly selected set of n events, along with all related auxiliary variables are unfixed. Thus, it is possible to achieve a local optima regarding how these events are split into sub-events, the time assignment for their sub-events and

the resource assignment for these events. Or course, this local optima has to respect the assignments from fixed variables.

3.2.2 Times Neighbourhood

This neighbourhood randomly selects a set of n times and any variable related to these times is unfixed. With this neighbourhood it is possible to find a local optima solution regarding the use of these timeslots. Also the split of events into sub-events and the resource assignment is optimized.

3.2.3 Resources Neighbourhood

This neighbourhood randomly selects a set of n resources. Each event that has at least one of these resources as pre-assigned is selected. Then any variable related to these events is unfixed. In this way, it is possible to achieve a local optima regarding the times that the resources occupy. Thus constraints like Limit Idle Times and Cluster Busy Times can be optimized for the selected resources.

3.2.4 Variables Neighbourhood

In this neighbourhood n percent of the variables are selected to become unfixed. This is the simplest neighbourhood and it is not problem-dependant. However, problem-dependant neighbourhoods perform much better than this totally random neighbourhood (see Table 3). Thus, this neighbourhood was considered only for comparison purposes.

4 Computational Experiments

All experiments ran on an Intel ® i7 4510-U 2.6 Ghz computer with 8GB of RAM computer under Ubuntu 12.04 operating system. The software was coded in C++ and compiled with GCC 4.6.1. The obtained results were validated by HSEval validator[1]. The time limit was adjusted to be equivalent of 1000 seconds in the benchmark provided by the ITC2011 organizers. Following another ITC2011 rule, the number of available threads was set to 1. Gurobi 6.0 was used as MIP solver.

The presented results are expressed by the pair (H, S), where H and S denote the cost of violation of the hard-constraints and the soft-constraints. When no hard-constraint is violated, only the cost of violation of the soft-constraints is presented. Our solver, along with our solutions and reports, can be found at GOAL-UFOP website[2]. We invite the interested reader to validate our results.

4.1 Neighbourhoods Performance

Table 2 presents the parameters considered in the experiments with proposed neighbourhoods. These parameters were empirically adjusted and smarter ways of selecting these parameters are subject of further research.

[1] `http://sydney.edu.au/engineering/it/` and `~jeff/hseval.cgi`
[2] `http://www.goal.ufop.br/softwares/hstt`

Parameter	Value
Number of events for n_{Events}	20
Number of timeslots for n_{Times}	15
Number of resources for $n_{Resources}$	5
Percentage of variables for $n_{Variables}$	50
Time limit of each MIP local search iteration (sec.)	100

Table 2 Considered parameters in computational experiments.

Table 3 presents the results of the proposed matheuristics over the XHSTT-2014 archive[3]. Only the instances where MIP solver was able to find a feasible solution in the given time limit are presented. Column MIP s_0 presents the initial solution provided for the matheuristics. Sub-columns s^* and $\bar{s}$ presents, respectively, the best and the average cost of solutions between five executions of the algorithm. Last line presents the average ranking of each solver according to the ITC2011 classification rules. For each instance, each solver receives a rank from 1 (best) to 4 (worst) according to the average result obtained in this instance. Then the average rank is calculated for each solver over all instances.

Table 3 Matheuristic results

		n_{Events}		n_{Times}		$n_{Resources}$		$n_{Variables}$	
Instance	MIP s_0	s^*	$\bar{s}$	s^*	$\bar{s}$	s^*	$\bar{s}$	s^*	$\bar{s}$
BR-SA-00	334	51	71.6	5	6.2	5	**5.2**	84	106.0
BR-SM-00	278	166	81.4	59	70.2	58	**64.8**	177	185.8
BR-SN-00	718	378	420.2	118	169.2	53	**69.6**	218	291.0
FI-WP-06	487	101	126.2	14	21.6	5	**6.6**	215	244.4
FI-MP-06	935	132	148.0	80	**82.4**	83	87.2	206	248.4
GR-H1-97	0	0	**0.0**	0	**0.0**	0	**0.0**	0	**0.0**
GR-P3-10	2332	337	446.8	0	**2.8**	39	91.4	2244	2314.4
GR-PA-08	135	49	58.6	3	**4.6**	10	11.4	74	79.8
IT-I4-96	25827	5138	7438.0	36	**64.4**	38	73.8	817	1284.0
Avg. Ranks			3.14		**1.58**		**1.58**		3.58

4.2 Improving Best Known Solutions

Table 4 presents the features of instances from XHSTT-2014 archive. It also presents the lower bound for each instance ($\mathcal{LB}$), the best known solution ($\mathcal{UB}$) and the new best known solution obtained in this work. When the previously best known solution is already optimal it is marked with a dash (-) meaning that it is not possible to improve this bound. To achieve these solutions, the best known solution was taken as initial solution s_0. After that, a improved version of GOAL solver, based in the Variable Neighbourhood Search algorithm [7] is executed with a large time limit (1000 seconds) generating a $s^{'}$ solution. In sequence, $s^{'}$ is taken as input for matheuristics Times neighbourhood and Resources neighbourhood with 1000 seconds time limit each. This procedure generates a $s^{''}$ solution. If $s^{''}$ is better than s_0 the whole process is repeated taking $s^{''}$ as initial solution.

[3] `http://www.utwente.nl/ctit/hstt/archives/XHSTT-2014/`

Table 4 New upper bounds for XHSTT-2014

Instance	Times	Resources	Events	Duration	$\mathcal{LB}$	$\mathcal{UB}$	New $\mathcal{UB}$
AU-BG-98	40	131	387	1564	0.0	(1, 386)	**(1, 365)**
AU-SA-96	60	99	296	1876	0	24	**17**
AU-TE-99	30	76	308	806	0	125	**67**
BR-SA-00	25	20	63	150	5	5	-
BR-SM-00	25	35	127	300	51	51	-
BR-SN-00	25	44	140	350	35	35	-
DK-FG-12	50	438	1077	1077	285	3310	**1775**
DK-HG-12	50	694	1235	1235	(7, 0)	(12, 3124)	**(12, 3056)**
DK-VG-09	60	262	918	918	(0, 0)	(2, 4097)	**(2, 2881)**
ES-SS-08	35	91	225	439	334	336	336
FI-PB-98	40	111	387	854	0	0	-
FI-WP-06	35	41	172	297	0	1	1
FI-MP-06	35	64	280	306	77	83	**77***
GR-H1-97	35	95	372	372	0	0	-
GR-P3-10	35	114	178	340	0	0	-
GR-PA-08	35	31	262	262	0	4	**3**
IT-I4-96	36	99	748	1101	27	34	**28**
KS-PR-11	62	164	809	1912	0	3	3
NL-KP-03	38	587	1156	1203	0	617	**527**
NL-KP-05	37	644	1235	1272	89	1078	**1017**
NL-KP-09	38	194	1148	1274	170	9180	**6265**
UK-SP-06	25	202	1227	1227	(0, 0)	(16, 2258)	**(15, 1892)**
US-WS-09	100	242	628	6354	0	697	**111**
ZL-LW-09	148	37	185	838	0	0	-
ZL-WL-09	42	70	278	1353	0	0	-

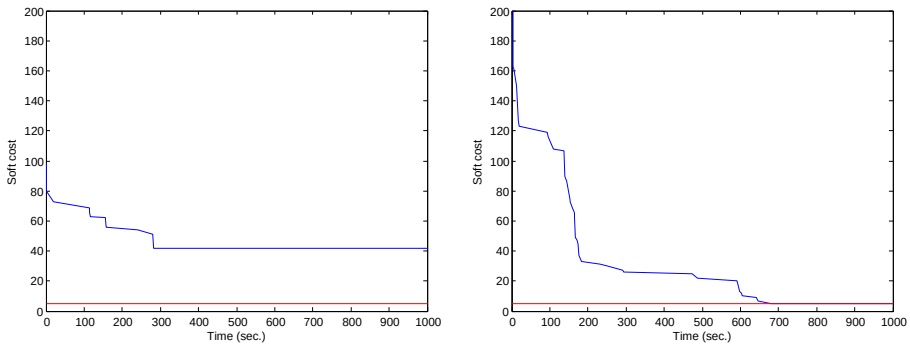
4.3 Discussion of Results

In Table 3 it is possible to conclude that Times neighbourhood and Resources neighbourhood achieved the best results. The first one performed significantly better in the Greek instances, while the second one had the best performance in Brazilian instances. An explanation for this result can be found by analysing the existing constraints in these instances. Greek instances has Link Events constraints, so, unfixing times for the MIP model makes it easier to keep the attendance of this constraint. Thus, all linked events may be moved to other times at the same time. This coincidence is harder to occur in Resources neighbourhood. Meanwhile, Brazilian instances has Cluster Busy Times constraint. Unlock all variables related to a resource is an efficient way to avoid penalties of this constraint.

Problem-dependant neighbourhoods performed well. Even with a short time limit, it was able to achieve the best known solution for 3 out of 9 instances. Actually, two of them were better than the previously best known solution. For the remaining instances the gap to the best known solution was also small (see best known solutions in Table 4). On the other hand, Variables neighbourhood performed poorly in all experiments. This was expected since problem-oriented neighbourhoods can avoid penalties for specific constraints way better than (possibly) non-related variables.

Aiming at improving the best known solutions, the proposed neighbourhoods along with the improved version of GOAL solver [7] were able to improve the best known solution for 14 out of 17 instances. Note also that one of these matches the lower bound and can be claimed optimal. The interaction between GOAL solver and the proposed matheuristics is promising. The main GOAL solver weakness relies on small instances,

where it usually gets stuck in local optima. Meanwhile, matheuristics perform well in this instance set. For big instances, mathematical approaches take a huge processing time to achieve good solutions. An algorithm that uses both GOAL solver's neighbourhoods and matheuristic neighbourhoods would be a robust solver. To illustrate this point, Figure 1 presents the behaviour of GOAL solver and matheuristic in instance BR-SA-00. The red line represents the lower bound for this instance. Note that GOAL solver becomes stuck in a local optima and spends ≈70% of available processing time without any improvement. A matheuristic neighbourhood could, for example, be invoked when situations like these were identified.



(a) GOAL solver (b) Matheuristic

Fig. 1 Behaviour of GOAL solver and Matheuristic in instance BR-SA-00.

5 Concluding Remarks

This work presented four matheuristic neighbourhoods for High School Timetabling. In the computational experiments one can observe the superiority of problem-dependant neighbourhoods over random selection of variables. More specifically, Times and Resources neighbourhoods achieved the best results.

Aiming at improving the best known solutions, the proposed neighbourhoods along with the GOAL solver were able to improve 14 out of 17 previously best known solutions. One of them can be claimed optimal since it matches the lower bound. The integration of these matheuristic neighbourhoods with the existing local search approach of GOAL solver is promising. We strongly believe that an hybrid solver could surpass the GOAL solver. This integration is the subject of further research.

Other possible future works are (1) improve the existing MIP formulation for XHSTT and (2) make a deeper study about neighbourhood sizes and procedures such as Local Branching [5] and Relaxation Induced Neighborhoods (RINS) [3].

Acknowledgements The authors would like to thank the Brazilian agencies CAPES, CNPq and FAPEMIG for the financial support.

References

1. Ahmed, L.N., Ozcan, E., Kheiri, A.: Solving high school timetabling problems worldwide using selection hyper-heuristics. Expert Systems With Applications, in review (2015)
2. Boschetti, M., Maniezzo, V., Roffilli, M., Boluf Rhler, A.: Matheuristics: Optimization, simulation and control. In: M. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, A. Schaerf (eds.) Hybrid Metaheuristics, *Lecture Notes in Computer Science*, vol. 5818, pp. 171–177. Springer Berlin Heidelberg (2009). DOI 10.1007/978-3-642-04918-7_13. URL http://dx.doi.org/10.1007/978-3-642-04918-7_13
3. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming **102**(1), 71–90 (2005)
4. Della Croce, F., Grosso, A., Salassa, F.: A matheuristic approach for the total completion time two-machines permutation flow shop problem. In: P. Merz, J.K. Hao (eds.) Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 6622, pp. 38–47. Springer Berlin Heidelberg (2011). DOI 10.1007/978-3-642-20364-0_4. URL http://dx.doi.org/10.1007/978-3-642-20364-0_4
5. Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1-3), 23–47 (2003). DOI 10.1007/s10107-003-0395-5. URL http://dx.doi.org/10.1007/s10107-003-0395-5
6. Fonseca, G., Santos, H., Toffolo, T., Brito, S., Souza, M.: Goal solver: a hybrid local search based solver for high school timetabling. Annals of Operations Research pp. 1–21 (2014). DOI 10.1007/s10479-014-1685-4. URL http://dx.doi.org/10.1007/s10479-014-1685-4
7. Fonseca, G.H., Santos, H.G.: Variable neighborhood search based algorithms for high school timetabling. Computers & Operations Research **52, Part B**(0), 203 – 208 (2014). DOI http://dx.doi.org/10.1016/j.cor.2013.11.012. URL http://www.sciencedirect.com/science/article/pii/S0305054813003328. Recent advances in Variable neighborhood search
8. Kheiri, A., Ozcan, E., Parkes, A.J.: HySTT: Hyper-heuristic search strategies and timetabling. In: Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012), pp. 497–499 (2012)
9. Kingston, J.H.: A software library for school timetabling (2012). Available at http://sydney.edu.au/engineering/it/~jeff/khe/, Accessed in December / 2012
10. Kingston, J.H.: KHE14: An algorithm for high school timetabling. In: 10th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2014), York, United Kingdom, pp. 26–29 (2014)
11. Kristiansen, S., Srensen, M., Stidsen, T.: Integer programming for the generalized high school timetabling problem. Journal of Scheduling pp. 1–16 (2014). DOI 10.1007/s10951-014-0405-x. URL http://dx.doi.org/10.1007/s10951-014-0405-x
12. Pirkwieser, S., Raidl, G.R.: Matheuristics for the periodic vehicle routing problem with time windows. Proceedings of matheuristics pp. 28–30 (2010)
13. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J.H., Kyngas, J., Nurmi, C., Ranson, D.: An xml format for benchmarks in high school timetabling. In: Annals of Operations Research DOI 10.1007/s10479-010-0699-9., pp. 3867 : 267–279 (2010)
14. Post, G., Di Gaspero, L., Kingston, J., McCollum, B., Schaerf, A.: The third international timetabling competition. Annals of Operations Research pp. 1–7 (2013). DOI 10.1007/s10479-013-1340-5. URL http://dx.doi.org/10.1007/s10479-013-1340-5
15. Post, G., Kingston, J., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research p. 17 (2011). URL http://dx.doi.org/10.1007/s1047901110122. 10.1007/s1047901110122
16. Romrs, J., Homberger, J.: An evolutionary algorithm for high school timetabling. PATAT '12 Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (2012)
17. Santos, H.G., Toffolo, T.A., Gomes, R.A., Ribas, S.: Integer programming techniques for the nurse rostering problem. Annals of Operations Research pp. 1–27 (2014). DOI 10.1007/s10479-014-1594-6. URL http://dx.doi.org/10.1007/s10479-014-1594-6
18. Sørensen, M., Stidsen, T.R.: Hybridizing integer programming and metaheuristics for solving high school timetabling. 10th International Conference on the Practice and Theory of Automated Timetabling pp. 557–560 (2014)
19. Srensen, M., Kristiansen, S., Stidsen, T.: International Timetabling Competition 2011: An Adaptive Large Neighborhood Search algorithm, pp. 489–492 (2012)

MISTA 2015

SAILS: Hybrid Algorithm for the Team Orienteering Problem with Time Windows

Aldy Gunawan · Hoong Chuin Lau · Kun Lu

Abstract The Team Orienteering Problem with Time Windows (TOPTW) is the extended version of the Orienteering Problem where each node is limited by a given time window. The objective is to maximize the total collected score from a certain number of paths. In this paper, a hybridization of Simulated Annealing and Iterated Local Search, namely SAILS, is proposed to solve the TOPTW. The efficacy of the proposed algorithm is tested using benchmark instances. The results show that the proposed algorithm is competitive with the state-of-the-art algorithms in the literature. SAILS is able to improve the best known solutions for 19 benchmark instances.

Keywords Orienteering Problem · Time Windows · Hybrid Algorithm · Simulated Annealing · Iterated Local Search

1 Introduction

The Team Orienteering Problem with Time Windows (TOPTW) is an extension of the Orienteering Problem (OP) [11]. A certain number of paths are required to serve a set of nodes. The visit on each node is limited by a given time window. The score of a particular node will be received once a node is visited within its time window. The main objective of the TOPTW is to maximize the total score from all visited nodes.

Since the OP has been proven as a NP-hard problem [5], it is unlikely that the TOPTW can be solved optimally within polynomial time. It is therefore interesting to propose fast heuristics to solve the problem, especially when we are dealing with real life large-scale applications of TOPTW, e.g. a personalized city trip planner [3, 21].

In this paper, we introduce a hybrid algorithm that combines two well-known metaheuristics, Iterated Local Search (ILS) and Simulated Annealing (SA). Iterated

A. Gunawan, H.C. Lau and K. Lu
School of Information Systems, Singapore Management University
Tel.: +65-68085227
Fax: +65-68280901
E-mail: {aldygunawan, hclau, kunlu}@smu.edu.sg

Local Search [15] is a simple but effective metaheuristic. In general, since it accepts only improving solutions or moves, we consider the incorporation of Simulated Annealing to avoid early termination in local optimality. Simulated Annealing [9] has been successfully applied to several combinatorial optimization problems [12–14]. It has the capability to escape from a local optimum by accepting a worse solution with a probability that changes over time. Our proposed algorithm is competitive with the state-of-the-art algorithms. More precisely, we show that it is able to improve the best known solution values of 19 benchmark instances. Hence, our work also serves as benchmark for future studies.

The paper is organized as follows. In Section 2, the TOPTW is briefly explained, including most recent works related to the TOPTW. Section 3 describes the proposed algorithm, SAILS, in detail. Section 4 is devoted to the experimental results and analysis. Finally, conclusions and ideas for future works are summarized in Section 5 .

2 The Team Orienteering Problem with Time Windows

2.1 Problem Description

The TOPTW is defined as follows. We are given an undirected network graph $G = (N,A)$ where $N = \{0,1,2,\ldots,|N|\}$ is the set of nodes, $A = \{(i,j) : i \neq j \in N\}$ refers to the set of arcs connecting two different nodes i and j and $M = \{1,2,\ldots,|M|\}$ is the set of paths. The non-negative travel time between nodes i and j is represented as t_{ij}. Each node $i \in N$ has a positive score u_i that would be collected the first time the node i is visited, a service time S_i and a time window $[e_i,l_i]$. e_i and l_i refer to the earliest and latest times allowed for starting the visit at node i.

In the TOPTW, it is assumed that node 0 is the start and end nodes, therefore $u_0 = S_0 = 0$. The visit to node i is successful if it begins within a time window $[e_i,l_i]$. Each node can only be visited at most once. The visit is allowed to wait until the time window begins in the case of an earlier arrival. In the context of TOPTW, the number of paths is fixed at $|M|$. Each path $m \in M$ is constrained within the time limit $[e_0,l_0]$. We have $e_0 = 0$ and $l_0 = T^{max}$, where T^{max} is the time budget or the maximum duration of the tour. The main objective is to maximize the total collected score of the visited nodes from $|M|$ paths. The mathematical formulation of the TOPTW can be found in [21].

2.2 Literature Review

Vansteenwegen et al. [21] introduced an Iterated Local Search (ILS) algorithm to solve the TOPTW with emphasis on providing a simple, fast and effective algorithm that can be tailored for a realistic Tourist Trip Design Problem (TTDP). Only two operations of ILS, INSERT and SHAKE, are considered in this deterministic algorithm. A metaheuristic algorithm based on Ant Colony System (ACS) was proposed by Montemanni and Gambardella [16]. The algorithm was further improved by Montemanni et al. [17], namely the Enhanced ACS (EACS) algorithm. The EACS algorithm includes two additional operations to overcome the drawbacks of ACS. Both operations

are related to the consideration of using the best solution found so far during the construction phase and applying the local search procedure only on those solutions on which the local search has not been recently applied.

In addition, Lin and Yu [13] proposed two different versions of Simulated Annealing, Fast SA (FSA) and Slow SA (SSA), in order to tailor two different scenarios. FSA is mainly for the applications that need quick responses, while SSA is more concerned about the quality of the solutions at the expense of more computational time. Labadie et al. [11] introduced an LP-based Granular Variable Neighborhood Search (GVNS) for solving the TOPTW.

Another ILS algorithm was proposed by Gunawan et al. [6] for solving the OPTW. The problem is also considered as the TOPTW with $|M| = 1$. The algorithm is started by generating an initial feasible solution using a greedy construction heuristic. The initial solution obtained is further improved by ILS. ILS is mainly based on several local search components, such as SWAP, 2-OPT, INSERT and REPLACE. The combination between ACCEPTANCECRITERION and PERTURBATION mechanisms is implemented to control the balance between diversification and intensification of the search. Computational results show that ILS is able to improve 8 best known solutions values of benchmark instances.

The idea of combining some advantages has been brought up by many researchers for solving different combinatorial optimization problems. Several taxonomies related to the hybrid algorithm were introduced by Talbi et al. [20] and Puchinger and Raidl [18]. Labadie et al. [10] introduced a hybridization of a Greedy Randomized Adaptive Search Procedure (GRASP) and an Evolutionary Local Search algorithm (ELS) for the TOPTW. Different constructive heuristics based on GRASP are proposed in order to build the initial solutions. Those initial solutions are further improved by the ELS algorithm. Another hybrid algorithm which is based a local search (LS) procedure, Simulated Annealing (SA) and Route Combination (RR) component is proposed by Hu and Lim [7]. Three components are iteratively incorporated within a certain number of iterations. It is shown that 35 new best solutions are found and more than 83% of instances with optimal solutions can be obtained.

Most recently, Cura [1] proposed an Artificial Bee Colony (ABC) algorithm to solve the TOPTW. Hybridization of SA and a new scout bee search behavior based on a local search procedure is introduced to improve the solution quality of benchmark instances. The proposed method is able to produce high-quality TOPTW solutions and comparable to other approaches. There is no new best found solution reported.

3 Hybrid Algorithm

In this section, we describe the proposed algorithm that combines Simulated Annealing (SA) and Iterated Local Search (ILS), namely SAILS. Instead of starting with a randomly generated initial solution which is commonly used in SA, we introduce a greedy construction heuristic for providing an initial solution. The initial solution is further improved by SAILS. By using SA, a new solution with a worse objective function value may be accepted with a certain probability. The possible neighbor-

Algorithm 1 CONSTRUCTION (N, M)

$N^* \leftarrow$node 0
$N' \leftarrow N \backslash$node 0
Initialize $S_0 \leftarrow N^*$
$F \leftarrow$ UPDATEF(N', M)
while $F \neq \emptyset$ **do**
  $\langle n^*, p^*, m^* \rangle \leftarrow$ SELECT(F)
  $S_0 \leftarrow \langle n^*, p^*, m^* \rangle$
  Update $P(m)$
  $N' \leftarrow N' \setminus \{n^*\}$
  $N^* \leftarrow N^* \cup \{n^*\}$
  $F \leftarrow$ UPDATEF(N', M)
end while
return S_0

Algorithm 2 UPDATEF (N', M)

$F \leftarrow \emptyset$
for all $n \in N'$ **do**
  for all $m \in M$ **do**
    for all $p \in P(m)$ **do**
      if insert node n in position p of path m is feasible **then**
        calculate $ratio_{n,p,m}$
        $F \leftarrow F \cup \langle n, p, m \rangle$
      end if
    end for
  end for
end for
Sort all elements of F in descending order based on $ratio_{n,p,m}$
Select the best f elements of F and remove the rest
return F

hoods are generated by implementing ILS. The details of the SAILS algorithm are described in the following sub-sections.

3.1 Greedy Construction Heuristic

The greedy construction heuristic is outlined in Algorithm 1. The idea of generating an initial solution is adopted from the one proposed by Gunawan et al. [6]. The earlier version is only dedicated for $|M| = 1$. Here, the heuristic is extended for $|M| > 1$. N' and N^* denote the sets of unscheduled and scheduled nodes, respectively ($N' \cup N^* = N$). N^* is initialized by the start and end nodes, node 0, while N' consists of all unscheduled nodes. S_0 refers the current feasible solution obtained so far, represented as m-row vectors. Each row is initialized with start and end nodes, node 0.

The construction heuristic is started by generating a set of all feasible candidate nodes to be inserted, F. Each element of F, which represents a feasible insertion of node n in position p of path m, is represented as $\langle n, p, m \rangle$. All possibilities of inserting an unscheduled node in position p of path m are examined. A insertion $\langle n, p, m \rangle$ is

Algorithm 3 SELECT (F)

```
SumRatio ← 0
for all ⟨n, p, m⟩ ∈ F do
    SumRatio ← SumRatio + ratio_{n,p,m}
end for
for all ⟨n, p, m⟩ ∈ F do
    prob_{n,p,m} ← ratio_{n,p,m} / SumRatio
end for
U ← rand(0, 1)
AccumProb ← 0
for all ⟨n, p, m⟩ ∈ F do
    AccumProb ← AccumProb + prob_{n,p,m}
    if U ≤ AccumProb then
        ⟨n*, p*, m*⟩ ← ⟨n, p, m⟩
        break
    end if
end for
return ⟨n*, p*, m*⟩
```

feasible if after the insertion, all scheduled nodes do not violate their respective time windows and the total spent time of path m does not exceed T^{max}.

Let $P(m)$ be a set of positions of scheduled nodes on path m. For each possible insertion, the benefit of insertion $ratio_{n,p,m}$ is calculated by equation 1. $Diff_{n,p,m}$ represents the difference between the total time spent before and after the insertion of node n in position p of path m. All elements of F are then sorted in descending order based on $ratio_{n,p,m}$ values. Only a subset of elements, f, would be kept. Algorithm 2 summarizes the algorithm of generating F.

$$ratio_{n,p,m} = \left(\frac{u_n^2}{Diff_{n,p,m}} \right) \tag{1}$$

If F is not an empty set, Algorithm 3 is run in order to select which $\langle n^*, p^*, m^* \rangle$ to be inserted. Each $\langle n, p, m \rangle$ corresponds to a particular probability value, $prob_{n,p,m}$. The probability is calculated by Equation 2:

$$prob_{n,p,m} = \left(\frac{ratio_{n,p,m}}{\sum_{\langle i,j,k \rangle \in F} ratio_{i,j,k}} \right) \tag{2}$$

The selection of $\langle n^*, p^*, m^* \rangle$ from F is based on Roulette-Wheel selection concept [4]. This method assumes that the probability of selection is proportional to the benefit of insertion of an individual, $ratio_{n,p,m}$. The accumulative of probability values, *AccumProb*, is initially set to 0. A random number $U \sim rand[0,1]$ is generated. We then select a particular $\langle n^*, p^*, m^* \rangle$ and update the value of *AccumProb* iteratively. This loop will be terminated when $(U \leq AccumProb)$ and the corresponding $\langle n^*, p^*, m^* \rangle$ is then selected. S_0, N' and N^* will also be updated. The greedy construction heuristic is terminated when $F = \emptyset$.

3.2 SAILS

Given the initial solution generated from the greedy construction heuristic, we propose a hybridization between Simulated Annealing (SA) and Iterated Local Search (ILS) to further improve the quality of the initial solution. The outline of SAILS is presented in Algorithm 4. The SA algorithm requires three parameters T_0, α and INNERLOOP. T_0 refer to the initial temperature. α is a coefficient used to control the speed of the cooling schedule. INNERLOOP denotes the number of iterations at a particular temperature.

Let S_0, S^* and S' be the current solution, the best found solution so far and the starting solution for each iteration, respectively. At the beginning, the current temperature $Temp$ is equal to T_0 and will be decreased after INNERLOOP iterations by using the following formula: $Temp = Temp \times \alpha$ $(0 < \alpha < 1)$.

At a particular value of temperature, we apply two components of ILS: PERTURBATION and LOCALSEARCH in order to explore neighborhoods of S_0. For each iteration, we calculate the difference between two solutions S_0 and S', denoted as δ. If δ is greater than 0, which implies that the improvement of the objective function does exist, S' is replaced by S_0. If S_0 also improves S^*, S^* is then replaced by S_0. On the other hand, if the solution generated is worse, a random number between 0 and 1, r, is generated and compared with $\exp(\delta/Temp)$. If this worse solution is accepted ($r < \exp(\delta/Temp)$), we update S'; otherwise, we return to S'. For each iteration, if there is no improvement of S^*, we increase the number of no improvement NOIMPR by one. In [21], the solution will only be accepted if it is better than the best found, otherwise the number of non-improvement iteration will be increased by one.

The main difference of the standard SA and our SAILS lies in the additional strategy applied. We include the intensification strategy. The idea of this strategy is as follows. If there is no improvement of the solution obtained after a certain number of iterations LIMIT, we focus the search once again starting from the best solution obtained S^*. Finally, the entire algorithm will be run within the computational budget TIMELIMIT.

The neighborhoods of the current solution is generated by ILS. Two components of ILS are considered: PERTURBATION and LOCALSEARCH. Two different steps implemented in PERTURBATION are: EXCHANGEPATH and SHAKE. If the number of iterations without improvement, NOIMPR, is larger than THRESHOLD1 and (NOIMPR + 1) Mod THRESHOLD2 = 0, EXCHANGEPATH would be executed; otherwise, SHAKE would be selected. THRESHOLD1 and THRESHOLD2 are two pre-set parameters. In EXCHANGEPATH step, all nodes from two different paths are selected and swapped. The strategy of selecting two different paths are based on generating of permutations by adjacent transposition method [8]. EXCHANGEPATH will only be implemented if the number of paths is more than one. Otherwise, we implement SHAKE.

The SHAKE step is adopted from [21]. One or more nodes will be removed from each path m, which depends on two integer values, CONS and POST. CONS indicates how many consecutive nodes to remove for a particular path while POST indicates the first position of the removing process on a particular path. If we reach the last scheduled node, the process will then be back to the first node after the start node,

Algorithm 4 SAILS (N,M)

```
S_0 ← CONSTRUCTION(N,M)
S* ← S_0
S' ← S_0
Temp ← T_0
NOIMPR ← 0
while TIMELIMIT has not been reached do
   INNERLOOP = 0
   WHILE INNERLOOP < MAXINNERLOOP DO
      S_0 ← PERTURBATION(S_0,N*,N',M)
      S_0 ← LOCALSEARCH(S_0,N*,N',M)
      δ ← S_0 − S'
      IF δ > 0 THEN
         S' ← S_0
         IF S_0 IS BETTER THAN S* THEN
            S* ← S_0
            NOIMPR ← 0
         ELSE
            NOIMPR ← NOIMPR + 1
         END IF
      ELSE
         r ← rand[0,1]
         IF r < exp(δ/Temp) THEN
            S' ← S_0
         ELSE
            S_0 ← S'
         END IF
         NOIMPR ← NOIMPR + 1
      END IF
      INNERLOOP ← INNERLOOP + 1
   END WHILE
   Temp ← Temp × α
   IF NOIMPR > LIMIT THEN
      S_0 ← S*
      S' ← S_0
      NOIMPR ← 0
   END IF
end while
return S*
```

node 0. Both CONS and POST are initially set to 1. After each SHAKE step, POST is increased by CONS. CONS would also be increased by 1 after a fixed number of consecutive iterations, e.g. 2 iterations.

If POST is greater than the size of the smallest path, POST is subtracted with the size of the smallest path to determine the new position POST. If CONS is greater than the size of the largest path, or S^* is updated, CONS is reset to one. Take note that CONS is always increased by 1 for each iteration and would be set to 1 if it equals to $\frac{n}{3\times|M|}$ in [21]. After removing CONS nodes, we update N' and N^* accordingly. F is then regenerated based on Algorithm 2 and an unscheduled node that needs to be inserted is selected using Algorithm 3. This is repeated until $F = \emptyset$.

Table 1 presents six operations in LOCALSEARCH that are run consecutively and applied to S_0. When m = 1, only SWAP1, 2-OPT, INSERT and REPLACE are con-

Table 1: LOCAL SEARCH operations.

Operations	Descriptions
SWAP1	Exchange two nodes within one path
SWAP2	Exchange two nodes within two paths
2-OPT	Reorder the sequence of certain nodes within one path
MOVE	Move one node from one path to another path
INSERT	Insert nodes into a path
REPLACE	Replace one scheduled node with one unscheduled node

sidered. SWAP1 is applied by exchanging two scheduled nodes within one particular path with the lowest remaining travel time. We examine all possible combinations of selecting two different nodes. SWAP1 is executed if it is able to increase the remaining travel time of selected path and there is no constraint violation. The idea of SWAP1 is extended to two different paths with the lowest and the second lowest remaining travel times, namely SWAP2. This operation will be accepted if the total remaining travel times from both paths is increased. Both SWAP1 and SWAP2 would be terminated when there is no further improvement in terms of the remaining travel times.

2-OPT is started by selecting one path with the lowest remaining travel time. All possible combinations of selecting two different nodes are enumerated and the sequence of scheduled nodes is reversed as long as there is no constraint violation. It has to increase the remaining travel time of the selected path. This would be terminated until no further improvement in terms of the total of remaining travel time of the selected path.

MOVE is performed by reallocating one node from one path to another path. It is started from the first scheduled node n^* from first path m^*. We try to insert node n^* in another path. First, F is generated by using Algorithm 2 where $N' = \{n^*\}$ and $M = M \setminus \{m^*\}$. If $F \neq \emptyset$, node n^* would be reallocated using Algorithm 3. Otherwise, the process will continue to the next scheduled node. This operation would be terminated if node n^* is moved successfully or the last scheduled node of the last path $|M|$ is reached.

The purpose of INSERT is to insert one unscheduled node to a particular path. It is started by generating F based on Algorithm 2 and selecting node $i \in N'$ to be inserted by using Algorithm 3. After the insertion, S_0, N', N^* and F are updated accordingly. This is repeated until $F = \emptyset$. In the last operation REPLACE, one scheduled node $i \in N^*$ is replaced with one unscheduled node $j \in N'$. The operation is started by selecting path m with the highest remaining travel time, followed by selecting one node $j \in N'$ with the highest score u_j. We then check each position p of the selected path and examine whether selected node j can replace the node in position p. Once this operation is successful, the process will continue to the next unscheduled node j and repeat the operation. Otherwise, the operation would be terminated.

Table 2: Benchmark Instances

References	Names	Instance Sets	$\|N\|$	$\|M\|$
[19]	Solomon	c100, r100, rc100	100	1 to 4
	Cordeau	pr01 - pr10	[48, 288]	
[16]	Solomon	c200, r200, rc200	100	1 to 4
	Cordeau	pr11 - pr20	[48, 288]	
[21]	Solomon	c100, r100, rc100	100	
		c200, r200, rc200	100	up to number of vehicles
	Cordeau	pr01 - pr10	[48, 288]	

Table 3: Estimation of single-thread performance

Algorithm	Experimental environment	Estimate of single-thread performance
IterILS	Intel Core 2 with 2.5 GHz processor	0.92
ACS	Dual AMD Opteron 250 2.4 gigahertz CPU, 4 gigabytes RAM	0.39
SSA	Intel Core 2 CPU, 2.5 gigahertz	0.92
GVNS	Intel Pentium (R) IV, 3 gigahertz CPU	0.39
I3CH	Intel Xeon E5430 CPU clocked at 2.66 gigahertz, 8 gigabytes RAM	1.16
SAILS	Intel(R) Core(TM) i5 CPU with 3.2 GHz processor, 12 GB RAM	1

4 Computational Results

4.1 Benchmark Instances and Approach Comparison

The benchmark instances are categorized into three groups, as listed in Table 2. All benchmark instances can be accessed at `http://www.mech.kuleuven.be/en/cib/op`. The first two groups are considered as "INST-M" which contain four instance sets: "Solomon 100", "Solomon 200", "Cordeau 1-10" and "Cordeau 11-20". The last group is known as "OPT". The optimal solution for each instance in this group is known as the total score of all nodes on the network graph [7].

The performances of SAILS are compared against the state-of-the-art algorithms: Iterated Local Search (IterILS) [21], Ant Colony System (ACS) [16, 17], Slow Simulated Annealing (SSA) [13], Granular Variable Neighborhood Search (GVNS) [11] and Iterative Three-Component Heuristic (I3CH) [7]. In order to ensure the fairness among algorithms, we also follow the same approach by using the *SuperPi* benchmark [7] to adjust the computational time to the speed of the computers used in other solutions. The main idea is to set the performance of our machine to be 1 and estimate the single-thread performance of other processors by multiplying with the single-thread performance estimation, as shown in Table 3.

We propose two different scenarios for running SAILS. In the first scenario, we refer to the computational time used by ACS since we are more concerned about the quality of the solution rather than the solution time. Only ACS uses the computational budget, while the rest use the number of iterations. Our experiments use 35% of ACS's computational budget (= 3600 seconds). Therefore, the computational budget for each instance is set to $35\% \times 0.39 \times 3600$ seconds ≈ 492 seconds using our

Table 4: New best known solution values found by SAILS (first scenario)

Instance	m	Old BK	New BK	Instance	m	Old BK	New BK
r206	1	1029	1032	pr18	2	938	946
r208	1	1112	1115	r104	3	777	778
rc206	1	895	899‡	rc104	3	834	835
r107	2	536	538	pr02	3	942	943
pr04	2	925	926	r104	4	972	973
pr09	2	905	909	rc103	4	974	975
c204	2	1480	1490	rc107	4	980	985
pr13	2	832	843				

‡ Same result with that of ILS [6]

processor (refer to Table 3). In the second scenario, we conduct experiments in which SAILS is set to the same computational time of I3CH. It has been proven that I3CH outperforms other algorithms, such as IterILS, SSA and GVNS [7].

For SAILS, each instance is executed in 10 runs with different random seeds. ACS was executed in 5 runs whereas GVNS was also executed 10 runs. IterILS, SSA and I3CH were only executed once and reported one solution for each instance. Some parameter settings adopted from [6] are as follows: f = 5, THRESHOLD1 = 20 and THRESHOLD2 = 3. Other SA parameters have been selected according to preliminary experiments using a subset of instances. The values of parameters considered are as follows: $\alpha \in \{0.5, 0.75, 0.9\}$, $Temp \in \{500, 1000, 1500, 2000\}$ and MAXINNERLOOP $\in \{50, 100\}$. Only one parameter is set to a constant value, using the formula: LIMIT = 0.05 $\times$ MAXINNERLOOP. All possible combinations were run in order to obtain the final parameter values: α = 0.75, T_0 = 1000 and MAXINNERLOOP = 50.

4.2 Computational Results

We report a comprehensive analysis of the results obtained by SAILS. Table 4 presents 15 new best known solutions (*BK*s) obtained by SAILS, 40% of them are from instances with $m = 2$ while each of other m values has 20% of new *BK*s. Around 33% of new *BK*s are from Cordeau et al.'s datasets which is harder to solve compared against Solomon's datasets [2]. We only report the results of Cordeau et al.'s datasets for m = 1 to 4 due to space constraints, as shown in Tables 5-8. The complete results is available at `http://centres.smu.edu.sg/larc/Orienteering-Problem-Library`.

Tables 5 - 8 consist of two identical structure parts. The first column shows the instance name. The second column contains the best known solution value *BK* from one of the state-of-the-art algorithms: IterILS, ACS, SSA, GVNS and I3CH. The following three columns present maximum, average and minimum solution values obtained by SAILS from 10 runs. The "*BG* (%)" column refers to the percentage gap between *BK* and the maximum (best) solution obtained by a particular algorithm. "*AG* (%)" provides the percentage gap between *BK* and the average solution obtained by a particular algorithm. The last three columns show maximum, average and minimum computational times (in seconds) required to obtain the best found within the given computational time. The new *BK* are highlighted in **bold**.

Table 5: Detailed results of SAILS on Cordeau et al.'s instances with $m = 1$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	*308*	308	308	308	0.0	0.0	5.7	2.4	0.0	pr11	353	353	353	353	0.0	0.0	412.3	124.7	10.6
pr02	*404*	404	404	404	0.0	0.0	230.8	43.4	6.8	pr12	442	441	440	439	0.2	0.5	87.9	48.2	6.2
pr03	*394*	394	394	394	0.0	0.0	40.7	18.7	0.4	pr13	466	458	455.9	455	1.7	2.2	407.8	152.3	17.7
pr04	*489*	489	482.8	471	0.0	1.3	425.0	120.3	16.2	pr14	567	552	545.2	525	2.6	3.8	408.5	128.1	26.2
pr05	*595*	592	591.1	590	0.5	0.7	481.6	189.5	34.3	pr15	707	707	686.4	662	0.0	2.9	299.2	156.8	52.6
pr06	591	579	565	553	2.0	4.4	262.5	90.0	14.5	pr16	674	650	636	621	3.6	5.6	462.3	135.2	44.2
pr07	*298*	298	298	298	0.0	0.0	13.9	3.5	0.1	pr17	362	362	362	362	0.0	0.0	74.4	37.0	7.8
pr08	*463*	463	463	463	0.0	0.0	30.6	10.1	2.6	pr18	539	539	538	533	0.0	0.2	490.9	98.9	13.3
pr09	*493*	493	491.8	490	0.0	0.2	153.8	70.8	5.9	pr19	562	560	546.1	536	0.4	2.8	397.3	151.2	4.1
pr10	594	583	577.7	563	1.9	2.7	490.2	208.2	30.5	pr20	667	648	635.5	627	2.8	4.7	356.5	151.0	48.1

Table 6: Detailed results of SAILS on Cordeau et al.'s instances with $m = 2$

Instance	*BK*	SAILS			*BG*	*AG*	Time			Instance	*BK*	SAILS			*BG*	*AG*	Time		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	502	502	502	502	0.0	0.0	91.5	22.3	2.2	pr11	566	566	563.5	559	0.0	0.4	195.1	35.1	5.8
pr02	715	712	701.8	693	0.4	1.8	451.4	136.6	18.6	pr12	774	766	757.8	743	1.0	2.1	260.4	130.4	26.3
pr03	742	742	732.3	710	0.0	1.3	437.4	252.7	96.9	pr13	832	**843**	826.8	811	-1.3	0.6	476.5	174.7	48.8
pr04	925	**926**	909.4	898	-0.1	1.7	431.0	218.5	24.6	pr14	1017	986	965.4	924	3.0	5.1	471.4	275.4	88.2
pr05	1101	1092	1068.7	1038	0.8	2.9	352.7	218.5	75.7	pr15	1219	1206	1175.6	1130	1.1	3.6	470.1	323.2	195.6
pr06	1076	1045	1022.2	993	2.9	5.0	482.3	303.0	39.7	pr16	1231	1149	1132.7	1107	6.7	8.0	406.2	265.8	131.3
pr07	566	566	566	566	0.0	0.0	372.9	90.2	5.2	pr17	652	646	644.5	643	0.9	1.2	291.6	128.6	13.5
pr08	834	830	821.7	809	0.5	1.5	455.0	202.2	19.9	pr18	938	**946**	924.3	909	-0.9	1.5	439.5	220.0	33.9
pr09	905	**909**	888.2	861	-0.4	1.9	453.0	192.4	35.2	pr19	1034	1023	993.9	956	1.1	3.9	484.9	321.1	135.9
pr10	1129	1111	1074.6	1044	1.6	4.8	424.0	232.9	71.6	pr20	1232	1203	1172.7	1121	2.4	4.8	489.4	314.4	152.4

Table 7: Detailed results of SAILS on Cordeau et al.'s instances with $m = 3$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	622	619	614.5	606	0.5	1.2	390.0	95.6	2.6	pr11	654	654	652.1	649	0.0	0.3	205.6	57.3	5.5
pr02	942	**943**	932.4	920	-0.1	1.0	487.9	246.1	28.2	pr12	1002	993	976.4	959	0.9	2.6	454.4	218.2	33.8
pr03	1010	1004	992.3	972	0.6	1.8	429.4	187.3	23.7	pr13	1145	1132	1119.1	1108	1.1	2.3	360.5	155.6	44.8
pr04	1294	1281	1259.8	1243	1.0	2.6	466.6	282.7	71.6	pr14	1372	1341	1317.2	1294	2.3	4.0	454.8	289.6	143.8
pr05	1482	1459	1434.3	1414	1.6	3.2	490.2	350.1	141.2	pr15	1654	1621	1600.5	1578	2.0	3.2	484.0	408.6	264.7
pr06	1514	1474	1428.6	1392	2.6	5.6	492.4	325.0	136.5	pr16	1668	1574	1554.5	1532	5.6	6.8	472.5	375.7	252.0
pr07	744	741	736.2	732	0.4	1.0	402.6	152.1	7.3	pr17	841	835	827.4	817	0.7	1.6	331.5	92.9	13.5
pr08	1139	1124	1107.4	1082	1.3	2.8	451.5	237.6	33.8	pr18	1281	1250	1230.4	1202	2.4	4.0	397.4	234.2	77.6
pr09	1275	1244	1225.3	1203	2.4	3.9	492.0	260.1	69.0	pr19	1417	1372	1350.4	1311	3.2	4.7	488.9	344.1	137.2
pr10	1573	1537	1490	1448	2.3	5.3	384.6	233.1	75.8	pr20	1684	1651	1608.9	1575	2.0	4.5	472.5	333.2	117.7

Table 8: Detailed results of SAILS on Cordeau et al.'s instances with $m = 4$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	657	657	657	657	0.0	0.0	55.1	8.9	2.0	pr11	657	657	657	657	0.0	0.0	7.0	4.3	0.3
pr02	1079	1068	1057.3	1041	1.0	2.0	476.4	170.9	13.9	pr12	1132	1112	1100.9	1083	1.8	2.7	467.8	174.8	25.5
pr03	1232	1228	1191.7	1157	0.3	3.3	439.7	185.2	79.6	pr13	1386	1363	1332.4	1285	1.7	3.9	398.3	178.7	60.8
pr04	1585	1543	1518.5	1474	2.6	4.2	443.8	222.9	53.7	pr14	1670	1660	1606.5	1564	0.6	3.8	492.7	327.3	160.8
pr05	1838	1774	1735.3	1659	3.5	5.6	453.8	328.9	160.4	pr15	2065	1953	1924.4	1867	5.4	6.8	495.6	389.0	264.6
pr06	1860	1796	1765.8	1729	3.4	5.1	481.6	325.2	140.0	pr16	2065	1945	1914.8	1877	5.8	7.3	481.0	382.4	237.1
pr07	876	869	860	842	0.8	1.8	477.2	167.2	7.6	pr17	934	930	918.6	906	0.4	1.6	488.3	210.5	12.1
pr08	1382	1349	1333.6	1316	2.4	3.5	427.8	227.9	66.0	pr18	1539	1501	1474.9	1450	2.5	4.2	463.4	261.8	82.5
pr09	1619	1573	1540.6	1516	2.8	4.8	467.2	371.4	169.1	pr19	1750	1721	1677.2	1638	1.7	4.2	476.1	375.9	150.8
pr10	1943	1869	1835.6	1802	3.8	5.5	491.9	335.0	166.5	pr20	2062	2047	1954.6	1903	0.7	5.2	459.4	389.7	181.4

Table 9: Overall "Average" Comparison of SAILS to the state-of-the-art algorithms on "INST-M" instances

Instance Set	*Numb*	IterILS		ACS		SSA		GVNS		I3CH		SAILS	
		$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}^{\ddagger}$	$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}$	$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}^{\ddagger}$
$m = 1$													
c100	9	1.11	0.3	0.00	2.4	0.00	19.4	1.22	64.3	0.00	29.3	0.00	1.0
r100	12	1.90	0.2	0.24	148.0	0.11	21.5	2.68	11.4	0.56	33.3	0.00	5.9
rc100	8	2.92	0.2	0.00	55.3	0.00	20.4	3.51	3.8	1.66	29.7	0.06	4.8
c200	8	2.28	1.6	0.58	132.2	0.13	34.5	1.11	74.3	0.40	98.2	0.14	68.3
r200	11	2.90	1.6	3.17	600.9	1.30	42.1	3.38	13.1	1.05	204.9	1.05	281.6
rc200	8	3.43	1.5	2.04	596.2	0.96	46.2	3.96	6.2	2.68	138.8	1.20	158.8
pr01-10	10	4.74	1.7	1.22	627.9	0.98	103.2	1.62	4.8	1.07	126.8	0.93	75.7
pr11-20	10	9.56	1.8	11.87	342.6	3.71	149.3	4.26	9.3	4.28	151.5	2.28	118.3
$m = 2$													
c100	9	0.94	1.0	0.14	309.9	0.00	24.3	0.72	53.9	0.00	101.2	0.03	19.8
r100	12	2.36	0.8	0.55	532.2	0.23	33.7	1.80	23.3	0.58	73.3	0.08	83.7
rc100	8	2.47	0.6	1.27	504.1	0.19	37.2	2.80	7.8	0.90	68.5	0.17	60.4
c200	8	2.54	3.2	1.81	539.7	1.18	49.3	0.58	13.0	0.68	466.7	0.90	124.4
r200	11	2.74	2.1	3.71	1055.8	0.58	84.0	1.30	5.7	0.21	612.9	1.35	270.1
rc200	8	4.14	2.0	3.83	904.3	1.25	73.6	2.57	4.9	0.62	511.5	1.96	233.1
pr01-10	10	6.22	4.4	3.57	729.4	2.45	159.9	1.79	15.1	1.11	287.4	2.09	186.9
pr11-20	10	7.86	4.8	6.15	920.5	3.88	185.4	2.18	31.8	2.70	354.3	3.11	218.9
$m = 3$													
c100	9	2.55	1.4	0.79	402.7	0.33	32.4	0.95	63.7	0.11	221.3	0.50	75.7
r100	12	1.79	1.6	1.30	681.6	0.39	51.5	2.27	28.5	0.21	137.6	0.57	108.0
rc100	8	3.14	1.0	1.07	458.7	0.64	39.3	2.32	13.0	0.27	117.4	0.41	66.0
c200	8	1.93	2.0	1.63	509.7	1.24	54.9	0.19	3.0	0.00	14.3	0.86	154.1
r200	11	0.30	1.3	0.24	452.3	0.08	38.6	0.20	2.7	0.01	105.6	0.13	98.4
rc200	8	1.44	1.6	0.94	588.8	0.27	54.2	0.44	2.9	0.04	190.8	0.40	82.9
pr01-10	10	6.58	8.5	4.21	818.9	2.34	181.1	1.13	33.2	0.36	493.2	2.85	237.0
pr11-20	10	9.19	8.9	7.24	997.7	3.81	231.5	1.80	58.2	1.11	578.1	3.39	250.9
$m = 4$													
c100	9	3.11	2.2	1.27	476.8	0.55	45.5	1.63	51.4	0.10	304.5	1.38	108.0
r100	12	3.31	2.4	1.92	684.3	0.73	53.7	2.28	32.7	0.16	214.4	1.40	117.7
rc100	8	3.18	1.8	2.18	656.3	0.37	62.6	1.79	14.2	0.23	177.3	0.80	107.9
c200	8	0.00	0.9	0.07	3.0	0.00	38.4	0.00	0.2	0.00	0.1	0.00	14.0
r200	11	0.00	0.8	0.00	48.8	0.00	36.5	0.00	0.1	0.00	0.2	0.00	20.9
rc200	8	0.00	1.0	0.01	249.6	0.00	36.9	0.01	0.3	0.00	0.2	0.00	27.3
pr01-10	10	7.08	13.0	3.42	966.6	2.23	235.0	1.60	49.1	0.36	659.0	3.58	234.3
pr11-20	10	8.47	12.6	6.34	997.2	3.95	261.1	2.81	89.8	0.45	847.6	3.97	269.4
Grand Mean		3.50	2.8	2.33	541.5	1.09	81.2	1.74	24.8	0.69	233.8	1.14	124.1

‡ Average computational time to obtain the best found (in seconds)

Table 10: Overall ”Best” Comparison of SAILS to the state-of-the-art algorithms on ”INST-M” instances

Instance Set	*Numb*	IterILS	ACS	SSA	GVNS	I3CH	SAILS
		$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)
$m = 1$							
c100	9	1.11	0.00	0.00	0.56	0.00	0.00
r100	12	1.90	0.00	0.11	1.72	0.56	0.00
rc100	8	2.92	0.00	0.00	1.88	1.66	0.00
c200	8	2.28	0.40	0.13	0.55	0.40	0.00
r200	11	2.90	2.19	1.30	2.45	1.05	0.13
rc200	8	3.43	1.23	0.96	2.53	2.68	0.23
pr01-10	10	4.74	1.06	0.98	0.56	1.07	0.44
pr11-20	10	9.56	11.13	3.71	3.17	4.28	1.14
$m = 2$							
c100	9	0.94	0.00	0.00	0.47	0.00	0.00
r100	12	2.36	0.20	0.23	1.19	0.58	-0.03
rc100	8	2.47	0.33	0.19	0.78	0.90	0.00
c200	8	2.54	1.27	1.18	0.25	0.68	0.25
r200	11	2.74	3.16	0.58	0.67	0.21	0.46
rc200	8	4.14	2.70	1.25	1.68	0.62	0.68
pr01-10	10	6.22	2.59	2.45	0.82	1.11	0.56
pr11-20	10	7.86	5.00	3.88	1.21	2.70	1.40
$m = 3$							
c100	9	2.55	0.22	0.33	0.45	0.11	0.11
r100	12	1.79	0.36	0.39	1.22	0.21	0.11
rc100	8	3.14	0.35	0.64	0.91	0.27	-0.01
c200	8	1.93	1.10	1.24	0.07	0.00	0.35
r200	11	0.30	0.13	0.08	0.11	0.01	0.04
rc200	8	1.44	0.42	0.27	0.32	0.04	0.13
pr01-10	10	6.58	2.96	2.34	0.36	0.36	1.26
pr11-20	10	9.19	5.40	3.81	1.02	1.11	2.02
$m = 4$							
c100	9	3.11	0.36	0.55	1.04	0.10	0.38
r100	12	3.31	0.78	0.73	1.22	0.16	0.39
rc100	8	3.18	0.78	0.37	0.95	0.23	-0.01
c200	8	0.00	0.00	0.00	0.00	0.00	0.00
r200	11	0.00	0.00	0.00	0.00	0.00	0.00
rc200	8	0.00	0.00	0.00	0.00	0.00	0.00
pr01-10	10	7.08	2.76	2.23	1.08	0.36	2.08
pr11-20	10	8.47	5.53	3.95	2.05	0.45	2.05
Grand Mean		3.50	1.69	1.09	1.00	0.69	0.46

Table 11: Comparison of SAILS to the state-of-the-art methods on ”OPT” instances

Instance Set	*Numb*	IterILS		SSA		GVNS		I3CH		SAILS		
		$\overline{BG}(\%)$	$\overline{Time}$	$\overline{BG}(\%)$	$\overline{Time}$	$\overline{AG}(\%)$	$\overline{Time}$	$\overline{BG}(\%)$	$\overline{Time}$	$\overline{BG}(\%)$	$\overline{AG}(\%)$	$\overline{Time}^{\ddagger}$
c100	9	1.41	2.8	1.04	71.4	0.47	3.0	0.00	55.4	0.43	0.87	93.6
r100	12	1.93	2.7	0.42	96.2	1.55	15.3	0.07	1021.0	0.45	1.18	203.7
rc100	8	2.06	3.5	0.35	77.8	1.29	15.2	0.00	66.7	0.54	1.11	147.1
c200	8	0.00	1.0	0.00	38.5	0.00	0.2	0.00	0.7	0.00	0.00	19.8
r200	11	0.62	1.5	0.16	53.6	0.17	2.1	0.07	201.7	0.01	0.22	166.6
rc200	8	0.47	1.6	0.07	38.0	0.16	1.1	0.04	221.2	0.07	0.10	75.0
pr01-pr10	10	2.32	28.0	1.04	520.3	1.25	19.8	0.78	380.0	1.22	1.53	260.1
Grand Mean		1.30	6.1	0.45	133.7	0.74	8.5	0.15	319.4	0.40	0.75	146.3

‡ Average computational time to obtain the best found (in seconds)

Table 12: New best known solution values found by SAILS (second scenario)

Instance	*m*	Old *BK*	New *BK*	Instance	*m*	Old *BK*	New *BK*
r207	1	1072	1074	rc201	2	1384	1385
rc202	1	936	938‡	r112	4	971	972

‡ Same result with that of ILS [6]

Tables 9 reports the average of AG ($\overline{AG}$ (%)) and the average computational time (in seconds) ($\overline{Time}$) for each instance set of "INST-M". Since IterILS, SSA and I3CH were only run once, we also include their average of BG ($\overline{BG}$ (%)) although we cannot directly compare with $\overline{AG}$ (%). The *num* column provides the number of instances in a particular instance set. The values of $\overline{Time}$ for ACS and SAILS refer to the average of computational time (in seconds) in order to obtain the best found from all runs. On the other hand, the ones for IterILS, SSA, GVNS and I3CH refer to the average of computational time (in seconds) for solving one particular instance set. All values reported have been adjusted according to the computer's speed as listed in Table 3.

In general, SAILS is competitive with the state-of-the-art algorithms. IterILS is an algorithm with the main purpose of providing good solutions very quickly, whereas SAILS focuses on finding better solutions at the cost of larger computational times. SAILS outperforms ACS in terms of the computational time and the solution quality. ACS requires 1 hour ($\approx$ 1404 seconds using our PC) while SAILS only requires 492 seconds for solving one instance. The Grand Mean of $\overline{Time}$ of SAILS is around 23% of ACS's Grand Mean. In terms of the solution quality, SAILS is able to reduce the Grand Mean of $\overline{AG}$ up to 48.9%. SAILS also outperforms GVNS in terms of the solution quality. The $\overline{AG}$'s Grand Mean of SAILS and GVNS are 1.14% and 1.74%, respectively although SAILS spends more computational time compared against that of GVNS.

Tables 10 summarizes the comparison among algorithms in terms of the values of $\overline{BG}$. All algorithms except IterILS are able to provide the Grand Mean of $\overline{BG}$ below 1.7%. SAILS is the best compared against other algorithms where the grand Mean of $\overline{BG}$ is only 0.46%. It also has a narrow range of -0.03% to 2.08%. Three instance sets give negative values, meaning that SAILS achieves some improvements of some BKs in those instance sets. Two of them are from rc100 instance sets with $m = 3$ and 4. Table 11 reports the results obtained on "OPT" instances [21]. SAILS outperforms other algorithms, except I3CH in terms of the Grand Mean of $\overline{BG}$. SAILS provides better results with greater computational time. The Grand Mean values of $\overline{AG}$ for GVNS and SAILS are 0.74% and 0.75%, respectively. Thus, we can conclude that SAILS provides the trade-off between the solution quality and computational time, on average.

At first glance, SAILS requires more computational time compared against those of other algorithms except ACS. Therefore, we implement the following second scenario. Additional experiments were done by setting the computational time as the one of I3CH. It has been shown that I3CH outperforms other approaches when using the same computational time [7]. We encountered four additional new BKs, as shown in Table 12. The results of using the same computational time are presented in Tables 13 and 14. We observed that SAILS overall average performance in terms of $\overline{AG}$ is 0.12% better than that of I3CH. I3CH has a wider range for $\overline{BG}$ values. SAILS and I3CH ranges from -0.01% to 3.18% and from 0.00% to 4.28%, respectively. For "OPT" instances, I3CH performs best with the lowest Grand Mean of $\overline{BG}$. The value is only 0.15%. The computational time using I3CH is less than the one used in the first scenario, except for r100 instance set.

Table 15 summarizes the percentage improvement of the solution quality (in average) for all instance sets. In general, we can conclude that SAILS is able to improve

Table 13: Comparison with the same computational time on "INST-M" instances

m	Instance Set	I3CH	SAILS		$\overline{Time}$ (seconds)
		$\overline{BG}(\%)$	$\overline{BG}(\%)$	$\overline{AG}(\%)$	
1	c100	0.00	0.00	0.00	29.3
	r100	0.56	0.00	0.03	33.3
	rc100	1.66	0.00	0.10	29.7
	c200	0.40	0.00	0.32	98.8
	r200	1.05	0.33	1.36	207.5
	rc200	2.68	0.52	1.44	140.1
	pr01-10	1.07	0.37	1.02	126.9
	pr11-20	4.28	1.49	3.13	152.0
2	c100	0.00	0.00	0.18	101.0
	r100	0.58	-0.01	0.31	73.2
	rc100	0.90	0.02	0.32	68.4
	c200	0.68	0.25	0.88	466.7
	r200	0.21	0.51	1.48	616.0
	rc200	0.62	0.51	1.90	512.5
	pr01-10	1.11	0.73	1.81	287.2
	pr11-20	2.70	1.54	2.96	355.4
3	c100	0.11	0.22	0.77	220.9
	r100	0.21	0.14	0.70	137.4
	rc100	0.27	0.00	0.60	117.2
	c200	0.00	3.18	4.21	16.1
	r200	0.01	0.27	0.61	109.8
	rc200	0.04	0.52	1.49	192.3
	pr01-10	0.36	1.17	2.92	492.7
	pr11-20	1.11	1.41	3.05	578.8
4	c100	0.10	0.58	1.52	303.9
	r100	0.16	0.43	1.52	214.0
	rc100	0.23	0.17	0.99	177.0
	c200	0.00	0.00	0.00	1.4
	r200	0.00	0.02	0.13	4.0
	rc200	0.00	0.15	0.34	2.1
	pr01-10	0.36	1.74	3.72	658.2
	pr11-20	0.45	1.92	3.41	847.6
Grand Mean		0.69	0.57	1.36	234.4

the initial solution generated by the Greedy Construction Heuristic. The values range from 0.30% to 19.41%. SAILS performs best for $m = 1$ where the percentage of improvement is varied from 6.20% to 19.41%. Figure 1 shows the Grand Mean values obtained in terms of percentage improvement, as shown in Table 15. We observe that the higher the value of m, the lower the Grand Mean value. It is expected since the problem is more difficult for higher values of m. "OPT" instance sets are the most difficult to solve.

Table 14: Comparison with the same computational time on ”OPT” instances

Instance Set	I3CH	SAILS		$\overline{Time}$ (seconds)
	$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{AG}$(%)	
c100	0.00	2.15	2.92	55.6
r100	0.07	0.79	1.47	1018.6
rc100	0.00	1.15	1.90	66.8
c200	0.00	0.00	0.00	1.7
r200	0.07	0.31	0.97	204.8
rc200	0.04	0.38	0.96	222.5
pr01-10	0.78	1.46	1.89	382.1
Grand Mean	0.15	0.90	1.46	320.1

Table 15: The solution quality improvement by SAILS (in %)

Instance Set	”INST-M”				”OPT”
	$m = 1$	$m = 2$	$m = 3$	$m = 4$	
c100	9.43	10.47	9.45	9.03	5.08
r100	11.13	13.30	14.75	14.68	6.25
rc100	17.73	15.67	15.10	16.04	7.77
c200	6.20	6.15	6.71	0.40	0.30
r200	9.09	7.93	2.39	0.32	3.74
rc200	13.41	10.91	5.39	1.18	4.04
pr01-10	18.57	18.86	15.70	12.80	5.03
pr11-20	19.41	18.63	14.26	11.43	-
Grand Mean	12.87	12.59	10.50	8.17	4.50

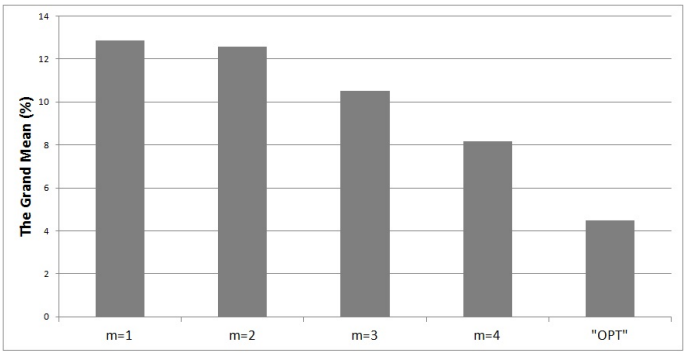


Fig. 1: The Grand Mean values for $m = 1$ to 4

5 Conclusion

In this paper, we present a hybridization of Simulated Annealing and Iterated Local Search, namely SAILS, to solve the TOPTW. The proposed algorithm is run in two different scenarios. The first scenario is to run SAILS with longer computational time since we are more concerned with the solution quality. The second scenario is mainly

tailored for the comparison purpose with the-state-of-the-art algorithms. This is done by setting the computational times to those of one of the-state-of-the-art algorithms, I3CH. Both scenarios are applied to benchmark instances.

Computational results show that SAILS is competitive with the-state-of-the-art algorithms. Simulated Annealing is able to improve the performance of Iterated Local Search by discovering 19 new best known solutions. Two areas of future work can be considered. Using different scenarios for building the initial solutions in order to observe the effect of Simulated Annealing would be one interesting area. And since the Orienteering Problem and its variants have attracted more attention in recent years, SAILS may be potentially applied to solve them.

Acknowledgements This research is supported by Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

References

1. Cura, T.: An artificial bee colony algorithm approach for the team orienteering problem with time windows. Computers and Industrial Engineering **74**, 270–290 (2014)
2. Duque, D., Lozano, L., Medaglia, A.: Solving the orienteering problem with time windows via the pulse framework. Computers and Operations Research **54**, 168–176 (2015)
3. Garcia, A., Arbelaitz, O., Vansteenwegen, P., Souffriau, W., Linaza, M.T.: Hybrid approach for the public transportation time dependent orienteering problem with time windows. In: E. Corchado, M. Romay, A. Savio (eds.) Hybrid Artificial Intelligence Systems, *Lecture Notes in Computer Science*, vol. 6077, pp. 151–158. Springer, Berlin, Germany (2010)
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts (1989)
5. Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics **34**(3), 307–318 (1987)
6. Gunawan, A., Lau, H.C., Lu, K.: An iterated local search algorithm for solving the orienteering problem with time windows. In: G. Ochoa, F. Chicano (eds.) proceedings of the 15th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoStar 2015), 8-10 April 2015, Copenhagen, Denmark, *Lecture Notes in Computer Science*, vol. 9026, pp. 61–73. Springer-Verlag, Berlin, Germany (2015)
7. Hu, Q., Lim, A.: An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research **232**(2), 276–286 (2014)
8. Johnson, S.M.: Generation of permutations by adjacent transposition. Mathematics of Computation **17**(83), 282–285 (1963)
9. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
10. Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R.W.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics **17**(6), 729–753 (2011)
11. Labadie, N., Mansini, R., Melechovskỳ J.and Calvo, R.: The team orienteering problem with time windows: an LP-based granular variable neighborhood search. European Journal of Operational Research **220**(1), 15–27 (2012)
12. Lee, D.H., Cao, Z., Meng, Q.: Scheduling of two-transtainer systems for loading outbound containers in port container terminals with simulated annealing algorithm. International Journal of Production Economics **107**(1), 115–124 (2007)
13. Lin, S.W., Yu, V.F.: A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research **217**(1), 94–107 (2012)
14. Lin, S.W., Yu, V.F., Chou, S.Y.: Solving the truck and trailer routing problem based on a simulated annealing heuristic. Computers and Operations Research **36**(5), 1683–1692 (2009)

15. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)
16. Montemanni, R., Gambardella, L.M.: Ant colony system for team orienteering problem with time windows. Foundations of Computing and Decision Sciences **34**(4), 287–306 (2009)
17. Montemanni, R., Weyland, D., Gambardella, L.M.: An enhanced ant colony system for the team orienteering problem with time windows. In: Proceedings of 2011 International Symposium on Computer Science and Society (ISCCS), pp. 381–384. Kota Kinabalu, Malaysia (2011)
18. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In: J. Mira, J.R. Alvarez (eds.) Artificial Intelligence and Knowledge Engineering Applications: First International Work-Conference on the Interplay between Natural and Artificial Computation, *Lecture Notes in Computer Science*, vol. 3562, pp. 41–53. Springer (2005)
19. Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers and Operations Research **36**(4), 1191–1203 (2009)
20. Talbi, E.G., Hafidi, Z., Geib, J.M.: A parallel adaptive tabu search approach. Parallel Computing **24**(14), 2003–2019 (1998)
21. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers and Operations Research **36**(12), 3281–3290 (2009)

MISTA2015

Cross-training performance of nurse scheduling with the learning effect

F. Akhavizadegan, R. Tavakkoli-Moghaddam, F. Jolai, J. Ansarifar

Abstract In recent years, the demand for health services has increased that causes one of the fundamental problems, such as a shortage of nurses. One of the effective strategies to deal with this problem is to use the cross-trained nurses. Furthermore, the nurse time spending on each bed decreases because of their experiences. The exponential distribution is used as a learning function. The learning effect can represent as the intuitive effect. Therefore, this research applies cross-training and learning effect simultaneously to formulate the nurse scheduling as a multi-objective mathematical model. The first objective minimizes the cost of nurses training and nurses wage, while the second objective function maximizes the nurse utilization. The third objective function reduces the nurse undesirability. Empirical data are collected from a healthcare center in Tehran in order to show the performance of our model. To solve the developed model, the NSGA-II and MOPSO algorithms are proposed and applied. The results show that using cross-trained nurses are the result of increasing the utilization while considering the learning effect can deal with the nursing shortage problem.

Keywords: Nurse scheduling; Cross-training; Learning effect; Nursing shortage

1 Introduction

One of the major challenges in improving the efficiency of healthcare services is nursing shortage that has attracted significant attention during recent years [1]. The National Center for Health Workforce Analysis reports that about 36% of nursing positions will stay vacant at 2020 in the United States [2]. Nature turnover of nurses and intention to leave their positions exacerbates the problem of scheduling nurses. Cross-training applies the idle nurses in the other home to work at home with the heavy demand [3]. Some researchers have considered cross-training of nurses, and models are formulated considering this issue. In this research, we

Faezeh Akhavizadegan
M.Sc. Student, School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
f.akhavizadegan@ut.ac.ir

Reza Tavakkoli-Moghaddam
Professor, School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
tavakoli@ut.ac.ir

Fariborz Jolai
Professor, School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
fjolai@ut.ac.ir

Javad Ansarifar
M.Sc. Student, School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
javad.ansarifar@ut.ac.ir

formulate the nurse scheduling problem by considering the cross-trained and learning effect as well as several objective functions. Minimizing the cost of nurses training and nurses' wage, maximizing the utilization of nurse, and reducing nurse undesirability is considered as three main objective functions of our model. In the literature, the undesirability means unwelcome or unwanted because they are considered the harmful effect on the person or cause unpleasant feeling. This research uses two meta-heuristic algorithms, namely NSGA-II and MOPSO, with tuned parameters to solve the presented model. To the best of our knowledge, this is the first work that presents cross-trained nurses scheduling with the learning effect with the foregoing objectives simultaneously.

The paper is structured as follows. Section 2 reviews the literature related to nursing scheduling and cross-training. Section 3 describes the mathematical model and methodology as well as solution approach is represented in Section 4. The case study and results of our model are shown in Section 5. Section 6 presents the sensitivity analysis of the model. The conclusion and future research area are provided in Section 7.

2 Literature review

The nurse scheduling problem has been attended in recent decades with different models and solutions for this subject. There are a number of comprehensive review papers on nurse scheduling [4-8]. In several review papers, it can be seen that most researchers have investigated the personnel and staff scheduling, specifically about nurses. However, they did not review the cross-training and learning effect on healthcare systems.

Martinelly et al. [9] considered the scheduling of nurses and operating rooms as recent published studies about nurse scheduling. Outpatient nurse scheduling was focused by Wang et al. [10], whereas most researchers consider an inpatient nurse scheduling problem. See also Ko et al. [11], Kim et al. [12], Wong et al. [13]. Santos et al. [14], Drake [15], Zheng and Gong [16], and Burke and Curtois [17] were the recent studies that considered nurse scheduling problems. One of the crucial challenges that many healthcare systems tackled is a shortage of nurses for service. Flinkman et al. [18] and Hayes et al. [19] considered some review papers in a nursing shortage issue and the nurse turnover problem which leads to nurse shortage. Chan et al. [20] and Toh et al. [21] focused on a nursing shortage issue. The first article analyzed the reasons for leaving work undertaken by nurses. The second article represented the relationship between the nursing shortage and other factors. See also Goodin [22], Heinz [23], Lu et al. [24], Ge et al. [25], and Zhu et al. [26] studied the nurse shortage in China.

One of the effective approaches to deal with the nursing shortage uses cross-trained nurses [27] and results in reducing the staffing costs and increasing the nurse's profit [28].

Zimmermann [29] and Li and King [30] considered the cross-training in nurse at the hospital occupational health service and staff planning, respectively. However, they did not consider the cross-training in their mathematical model. Inman et al. [31] and Alonso [32] described the positive and negative impacts of cross-training and effect of team training for the reduction of medical mistakes in a military health structure, respectively. Gnanlet and Gilland [2] considered cross-training in their model and presented the benefits of cross-training. Bard and Purnomo [33] represented a mathematical model for nurse scheduling with cross-trained nurses with some modifications in model constraints. Wright and Bretthauer [1] provided the strategies to deal with the US nursing shortage problem. Wright and Mahar [34] studied on nurse scheduling with cross-training in a hospital. Their results show that cross training results in reducing the costs and increasing the nurse satisfaction. The optimal models for allocation and cross-trained workers into homes in two stages were provided by Easton [35] in an uncertainty environment. Maenhout and Vanhoucke [36] developed the model for nurse scheduling with the cross-training and an objective to efficiently use cross training. Salas et al. [37] focused on the team training in the healthcare environment and proposed eight rules for effective development, performance and evaluation of team training programs. Punnakitikashem et al. [38] considered nurse scheduling with the cross-training effect, and that minimizes the surplus workload on the nurse and personnel cost as their objective. Also, Paul

and MacDonald [27] presented cross training strategies to deal with the nurse shortage problem and investigate the profits of cross training on nurse scheduling.

One of the contributions to this research is to consider the training effect in the formulation of nurse scheduling. By training, nurses become multi skills and can be reached skills needed to work in another home. Park [39] represented that training was recognized as a tool for increasing the flexibility. Each nurse has at least one skill and belongs to one skill level. Each nurse can be assigned to one home based on his/her skill level. During the each shift, nurses can be trained new skill and can improve their skills to work in the other homes.

Considering the learning effect in a cross-trained nurse scheduling problem is the main contribution of this research. The learning curve represents that the nurses' productivity is changed during each shift. If nurses perform the same work for a long time, they get tired and their productivity decreases. On the other hand, by repeating their tasks, the newly nurses can improve their productivity because learning results in reducing the time of doing tasks. Based on the reduction of the learning curve time of doing a task, the working time after some iterations are equal to Pn^{β} Where P represents the initial time of doing tasks, the number of iterations is shown as n and β is a negative coefficient. The learning effect was introduced for the first time by Wright [40], and then Biskup [41] developed the scheduling problem with the learning effect.

Some researchers have formulated the mathematical model to deal with the nurse scheduling. Berrada et al. [42] proposed a multi-objective approach to schedule the nurse with differentiates between hard and soft constraints. Gascon et al. [43] studied the scheduling of the flying squad nurses in hospitals considering the multiple objectives, such as minimizing the number of homes where a nurse works every two-week period and minimizing the number of days that nurses will work in homes during the next month. As recent research, Legrain et al. [44] developed a nurse scheduling model considering the multiple objectives, such as minimizing the total costs, maximizing the nurses' satisfaction, and leveling distributing the workload. The multi-objective nurse scheduling is developed by Burke et al. [45] who considered several objectives to deal with the nurse scheduling, such as minimizing the number of consecutive assignments of a specific shift type during the planning period, and maximizing the number of consecutive working days for part-time nurses during the planning period.

To the best of our knowledge, none of these researches are considered the multiple objectives, such as minimizing regular and Overtime wages and training cost, undesirability for nurses, and nurse utilization. This study maximizes the nurse utilization according to the learning effect concept. In addition to the cross-training, overtime has been taken into account to deal with a shortage of nurses. Our model applies not only the learning effect in cross-training and maximizing utilization as objective, but also some more modifications to the model presented by Wright and Mahar [34] to clearly indicate cross-training of nurses.

3 Proposed model for nurse scheduling

We formulate the problem as a multi-objective mixed integer programming (MIP) model. Minimizing the costs and staff undesirability is considered as the first and second objective functions, respectively. In this study, the utilization is maximized as a new objective. We develop the model proposed by Wright and Mahar [35]. We focus on the learning effect and cross-training nurses simultaneously. Therefore, some modifications are applied to their model [34].

Some assumptions are considered to formulate the nurse problem. Every nurse has a home based on his/her skill and can work on the other homes when the required training is passed and the demands of those homes are high. Each nurse must be working at least one shift per day and does not work in two successive shifts. We consider the utilization as objective function to be maximized. The time spent by a nurse on every bed is decreased by the learning effects. As a result, every nurse can service more beds on his specific shift at the home and

provides services for several times. In other words, some homes required for fewer nurses in some shifts. This action increases the nurse utilization. The sets, parameters and decision variables that will be used to formulate the nurse scheduling problem are indicated below.

Sets:

N	Set of nurses $i \in N$
J	Set of shifts $j \in J$
K	Set of homes $k, e \in K$
N^j	Set of nurses available for shift *j* that nurse *i* is available to work $N^j \in N$
SH^i	Set of shifts that nurse *i* is available to work $SH^i \in J$

Decision Variables:

X_{ijk}^r	1 if nurse *i* works shift *j* at regular time wages on home *k*; 0, otherwise
X_{ijk}^o	1 If nurse *i* works shift *j* at overtime wages on home *k*; 0, otherwise
Y_{ijke}^r	1 If nurse *i* whose home is *k* works shift *j* at regular time wages on home *e*; 0, otherwise
Y_{ijke}^o	1 If nurse *i* whose home is *k* works shift *j* at overtime wages on home *e*; 0, otherwise
Z_{ijke}	1 if nurse *i* whose home is *k* reassigned to home *e* for first time at shift *j*; 0, otherwise
R_{ijk}	Number of shifts until shift *j+1* that nurse *i* works on home *k*
H_{ijkr}	1 if nurse *i* in shift *j* has worked *r* consecutive shifts on home *k*; 0, otherwise
p_{ijk}	Average time nurse *i* spend for every bed at shift *j* on home *k*
UL_{jk}	Utilization shift *j* on home *k*

Parameters:

V_{jk}	The total time that home *k* require for eight-hour shift *j*
su	Upper limits on the number of shifts every nurse should work on planning horizon
sl	Lower limits on the number of shifts every nurse should work on planning horizon
c_{ijk}^r	Regular time wages if nurse *i* works shift *j* on home *k*
c_{ijk}^o	Overtime wages if nurse *i* works shift *j* on home *k*
w_{ijke}^r	Regular time wages if nurse *i* whose home is *k* works shift *j* on home *e*
w_{ijke}^o	Overtime wages if nurse *i* whose home is *k* works shift *j* on home *e*
us_i	Upper limits on the number of undesirable shifts assigned to nurse *me*
up_i	An upper limit on the number of homes which nurse i can be assigned
f_i	An upper limit on the number of overtime shifts assigned to a nurse
a_{ijk}	The undesirability that nurse, I have for shift *j on* home *k*
b_{ijke}	The undesirability that nurse *i* whose home is *k* has for shift *j* on home *e*
β	Negative coefficient
ct_{ke}	Cost of cross-training every nurse from home *k* to home *e*
M	Big number

The cross-trained nurses scheduling problem, according to the learning effect is formulated by:

$$\text{Min } Z_1 = \sum_{i\in N}\sum_{j\in SH^i}\sum_{k\in K}[(c^r_{ijk}\ X^{\ r}_{ijk} + c^o_{ijk}\ X^{\ o}_{ijk}) + \sum_{e\in K, e\neq k}(w^r_{ijke} Y^{\ r}_{ijke} + w^o_{ijke} Y^{\ o}_{ijke})] + \sum_{i\in N}\sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k} ct_{ke} Z_{ijke} \quad (1)$$

$$\text{Min } Z_2 = \sum_{i\in N}\sum_{j\in SH^i}\sum_{k\in K}[a_{ijk}(X^r_{ijk} + X^o_{ijk}) + \sum_{e\in K, e\neq k}(b_{ijke}(Y^r_{ijke} + Y^o_{ijke}))] \quad (2)$$

$$\text{Max } Z_3 = \sum_{k\in K}\sum_{j\in J} UL_{jk} \quad (3)$$

s.t.

$$\sum_{i\in N^j} P_{ijk}[(X^{\ r}_{ijk} + X^{\ o}_{ijk}) + \sum_{e\in K, e\neq k}(Y^{\ r}_{ijek} + Y^{\ o}_{ijek})] \geq v_{jk}\ ;\ j \in SH^i\ ;\ k \in K \quad (4)$$

$$\sum_{j\in SH^i}\sum_{k\in K} X^r_{ijk} + \sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k} Y^r_{ijke} \geq sl\ ;\ i \in N^j \quad (5)$$

$$\sum_{j\in SH^i}\sum_{k\in K} X^r_{ijk} + \sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k} Y^r_{ijke} \leq su\ ;\ i \in N^j \quad (6)$$

$$\sum_{j\in SH^i}\sum_{k\in K}(a_{ijk}(X^r_{ijk} + X^o_{ijk})) + \sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k}(b_{ijke}(Y^r_{ijke} + Y^o_{ijke})) \leq us_i\ ;\ i \in N^j \quad (7)$$

$$\sum_{j\in SH^i}\sum_{k\in K} X^o_{ijk} + \sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k} Y^o_{ijke} \leq f_i\ ;\ i \in N^j \quad (8)$$

$$Y^r_{ijke} + Y^o_{ijke} \leq M\sum_{j'=1}^{j-1} Z_{ijke}\ ;\ i \in N^j\ ;\ K, e \in K\ ; j \in \{2,\ldots, SH^i\}\ ;\ e \neq k \quad (9)$$

$$\sum_{j\in SH^i}\sum_{k\in K}\sum_{e\in K, e\neq k} Z_{ijke} \leq up_i\ ;\ i \in N^j \quad (10)$$

$$\sum_{j'=3j-2}^{3j}((X^r_{ijk} + X^o_{ijk}) + \sum_{e\in K, e\neq k}(Y^r_{ijke} + Y^o_{ijke})) \leq 1\ ;\ i \in N^j\ ;\ k \in K\ ; j = \{1,\ldots, J/3\} \quad (11)$$

$$\sum_{j'=j}^{j+1}((X^{\ r}_{ijk} + X^{\ o}_{ijk}) + \sum_{e\in K, e\neq k}(Y^{\ r}_{ijke} + Y^{\ o}_{ijke})) \leq 1, i \in N^j,\ k \in K, j \in SH^i \quad (12)$$

$$R_{ijk=}\sum_{j'=1}^{j}((X^r_{ijk} + X^o_{ijk}) + \sum_{e\in K, e\neq k}(Y^r_{ijek} + Y^o_{ijek}))\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \in SH^i \quad (13)$$

$$P_{ijk} = p_{ik}(R_{ijk}{}^{\beta})\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \in SH^i \quad (14)$$

$$X^r_{ijk} + X^r_{ij'k'} \leq 1\ ;\ i \in N^j\ ;\ k, k' \in K\ ;\ j, j' \in SH^i\ ;\ j \neq j'\ ;\ k \neq k' \quad (15)$$

$$X^o_{ijk} + X^o_{ij'k'} \leq 1\ ;\ i \in N^j\ ;\ k, k' \in K\ ;\ j, j' \in SH^i\ ;\ j \neq j'\ ;\ k \neq k' \quad (16)$$

$$X^r_{ijk} + Y^r_{ijke} + Y^o_{ijke} \leq 1\ ;\ i \in N^j\ ;\ k, e \in K\ ; e \neq k\ ; j \in SH^i \quad (17)$$

$$X^{\ o}_{ijk} + Y^{\ o}_{ijke} + Y^{\ r}_{ijke} \leq 1\ ;\ i \in N^j\ ;\ k, e \in K\ ; e \neq k\ ; j \in SH^i \quad (18)$$

$$X^r_{ijk} + X^o_{ijk} \leq 1\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \notin SH^i \quad (19)$$

$$\frac{v_{jk}}{\sum_{i\in N^j} P_{ijk}[(X^r_{ijk} + X^o_{ijk}) + \sum_{e\in K, e\neq k}(Y^r_{ijek} + Y^o_{ijek})]} \geq UL_{jk}\ ;\ k \in K\ ; j \in SH^i \quad (20)$$

$$X^r_{ijk} = 0\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \notin SH^i \quad (21)$$

$$X^o_{ijk} = 0\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \notin SH^i \quad (22)$$

$$Y^r_{ijke} = 0\ ;\ i \in N^j\ ;\ k, e \in K; e \neq k\ ;\ j \notin SH^i \quad (23)$$

$$Y^o_{ijke} = 0\ ;\ i \in N^j\ ;\ k, e \in K; e \neq k\ ;\ j \notin SH^i \quad (24)$$

$$X^r_{ijk} \in \{0,1\}\ ; X^o_{ijk} \in \{0,1\}\ ;\ Y^r_{ijke} \in \{0,1\}\ ; Y^o_{ijke} \in \{0,1\}\ ;\ i \in N^j\ ;\ k \in K; j \in SH^i \quad (25)$$

$$P_{ijk} \geq 0\ ;\ i \in N^j\ ;\ k \in K\ ;\ j \in SH^i \quad (26)$$

$$Z_{ijke} \in \{0,1\} \; ; \; i \in N^j \; ; \; k,e \in K \; ; e \neq k \; ; \; j \in SH^i \tag{27}$$

$$\mathrm{R}_{ijk} \in \text{ integer } ; \; i \in N^j \; ; \; k \in K \; ; \; j \in SH^i \tag{28}$$

The total nurses' wages and the cost of cross-training the nurses are minimized by the first objective function. The second objective function minimizes the nurses' dissatisfaction (i.e., undesirable shifts). The third objective function maximizes the sum of utilizations for every shift and all homes.

Constraint (4) certifies that total time of nurses allocated in each home for each shift to assure the time requirements. Constraint (5) guarantees that the number of regular time shifts assigned to each nurse must be more, in which the minimum number of shifts each nurse should work on the planning horizon. Constraint (6) ensures that the number of regular time shifts assigned to each nurse cannot exceed the maximum number of shifts, in which each nurse should work on the planning horizon. Constraint (7) is relevant to that undesirable regular or overtime shift for each nurse must be lower than the number of undesirable shifts assigned to a nurse. The number of overtime shifts assigned to each nurse must be lower than the upper limit on the number of overtime shifts. This constraint is guaranteed by Constraint (8). Constraint (9) represents that nurses cannot be assigned to other homes until their training is completed. The number of homes that nurses can be work is restricted by Constraint (10). Constraint (11) represents that each nurse must be working at least one shift per day and Constraint (12) certifies that each nurse does not work in two successive shifts. The number of shifts that each nurse works in each home shift j+1 is calculated by Constraint (13). Constraint (14) calculates the time nurses spend in each shift in each home by considering the learning effect. Every nurse must have only one home as base home. This constraint is guaranteed by Constraints (15) and (16). Constraints (17) and (18) are used to assign a nurse to other homes when he/she is not assigned to his/her home. Every nurse can work a regular or overtime type in each shift. This constraint is ensured by Constraint (19). Constraint (20) considers the nurse utilization by defining the utilization as the ratio of the total required time all, homes for all shifts to the time that the nurse spends on all homes for all shifts. Constraints (21) to (24) prevent assigning the nurse to the shifts, in which they are not available. Constraints (25) and (27) indicate the binary variables, and Constraint (26) shows the integer variable.

4 Solution methodology

The most important purpose of each multi-objective optimization algorithm is to discover not only one, but also a set of diverse solutions that called Pareto as optimal set. On the other hand, these algorithms attempt to distinguish a precise Pareto-optimal set of solutions that are not dominated and are a uniform distributed throughout the Pareto front. The NSGA-II and MOPSO algorithms have attracted much attention and have successful results in a wide variety of an optimization problem. Yin et al. [46] showed that each of these algorithms is one of the most efficient algorithm to solve the optimization problem of nurse scheduling. As the proposed model belongs to the NP-hard category the NSGA-II and MOPSO algorithms are applied after setting their parameters by the reaction surface methodology. These algorithms are used properly to obtain optimal solutions in a meaningful amount of time, specifically for the large-sized problem. Indeed, the performance of meta-heuristic algorithms very much depends on parameter configuration [47-49]. By setting their parameters for each of them, we can guarantee that these algorithms can obtain, the more precise near-optimal solution. There are several methods to tune the parameters of meta-heuristic algorithms. For example, Asefi et al. [47] and Wang and Liu [48] applied the Taguchi method for tuning parameters, where the surface response methodology (RSM) is used by Azadeh et al. [49] to tune our proposed algorithms.

4.1 Chromosome representation

We consider two chromosomes to code our model in MATLAB®. The feasible solution can get with these two chromosomes. The first one contains the one row with K+2 gens as shown in Figure 1. The first and second genes indicate the nurse's number and home number, as based home, respectively. Other genes indicates additional homes that one if nurses are trained to work there and are assigned this home to the nurse.

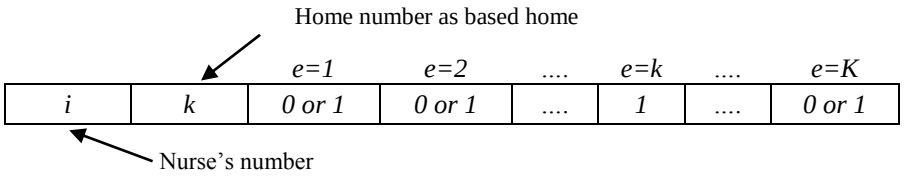


Figure 1. Representation of the first chromosome

The second chromosome is shown in Figure 2 that contains one row and J+3 gens. The first and second gene indicates the nurse's number and home number, respectively. The third gene indicates that the nurse is in regular or overtime and for other genes, if the gen takes 1, a certain nurse allocated to specific home at shift j.

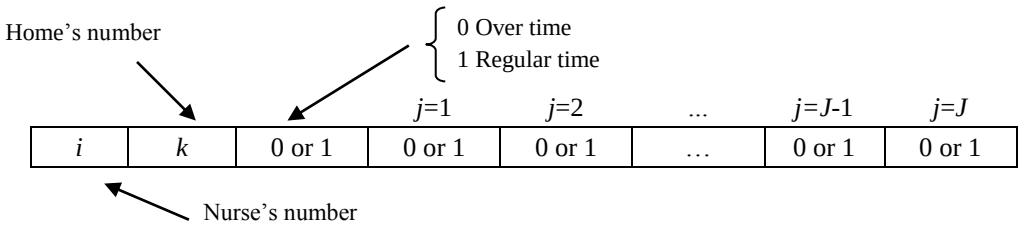


Figure 2. Representation of the second chromosome

4.2 NSGA-II

The genetic algorithm (GA) is the most popular and appropriate meta-heuristic algorithm in order to solve multi-objective optimization problems, because it does not require considering to prioritize, scale, or weigh for objectives [50]. Deb et al [51] introduced the fast non-dominated sorting genetic algorithm (NSGA-II) as its main structure is shown in Figure 3.

Mutation operation

We consider two mutations in chromosomes. In the first mutation that is belonging to the first chromosome, we choose a nurse at random and we put a number between 1 and k in the gene for a home base randomly. In the second mutation that is belongs to the second chromosome, we choose a nurse at random and we do not change the home base's gen. Some other gens are chosen randomly; if the gen's number is equal 1 that was replaced with 0 and if the gen's number is equal 0 that was replaced with 1. For the second chromosome, we choose a nurse and home number randomly. Some gens from 3 to $J+3$ are selected randomly; if the gen's number is equal 1 that was replaced with 0 and if the gen's number is equal 0 that was replaced with 1.

Crossover

We consider two crossovers for two chromosomes. In the first crossover that is belongs to the first chromosome, we choose two nurses at random and we change the basis of their home with each other. In the second crossover, we choose two nurses at random and we select a number from 2 to k+1. From this gens to the end of the chromosome, gens are replaced peer to peer with each other.

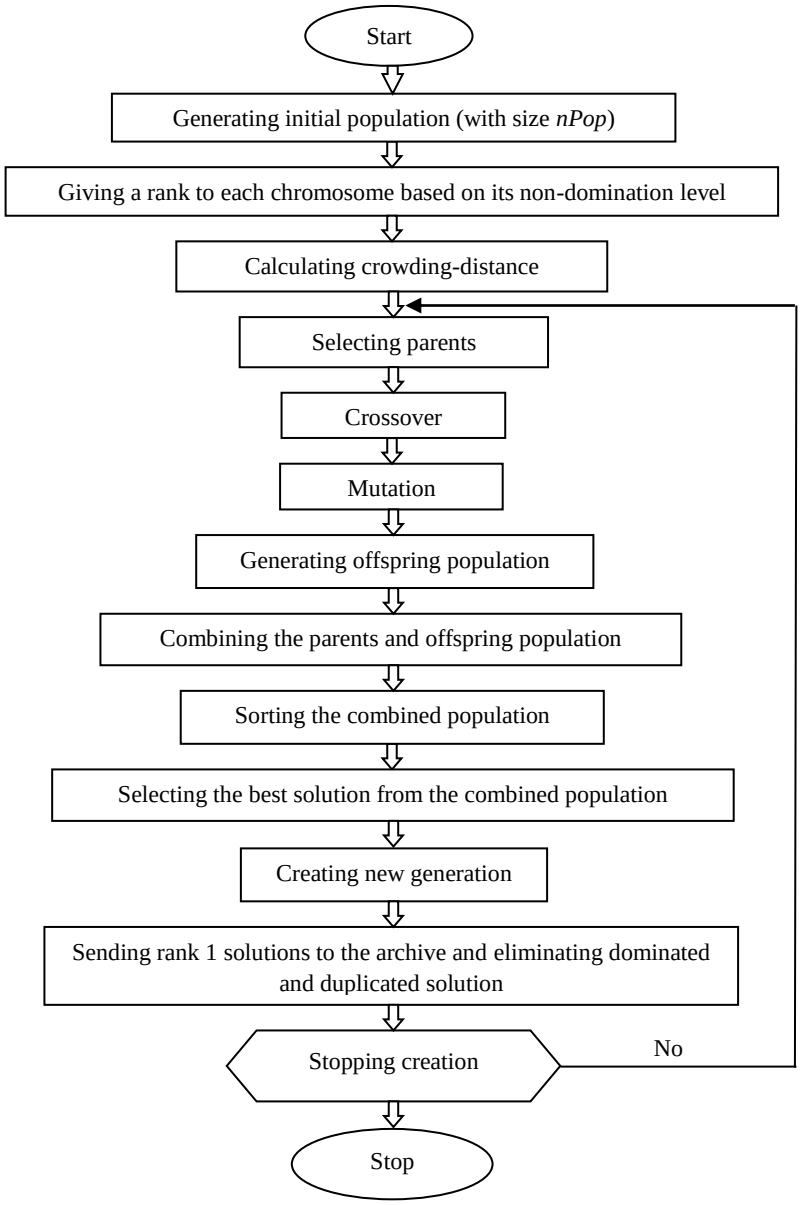


Figure 3. Flow chart of the proposed NSGA-II

4.3 MOPSO

One of the most popular meta-heuristic methods are particle swarm optimization (PSO), because comparison of this algorithm with other evolutionary algorithms is relatively simple and it is an appropriate algorithm to be used for multi-objective optimization problems. This algorithm is introduced on the basis of the movements of a flock of birds or fishes who look for food by Kennedy and Eberhart [52]. Despite all the algorithms are unsuccessful to find an optimal solution in a reasonable time for various types of problems with multiple objectives, MOPSO has demonstrated a suitable performance compared with other evolutionary multi-objective algorithms [53]. Consequently, the objective functions of this research are optimized by using the MOPSO algorithm. This algorithm was proposed by Moore and Chapman [54] for the first time, while the researchers have focused on this area since 2002. We use the MOPSO proposed by Coello et al. [53] to optimize the objective function. Based on this algorithm, the repository is defined as the best non-dominated sorting solutions obtained until now. Position and speed of each particle are calculated by:

$$v_i^{t+1} = wv_i^t + c_1R_1\left\lfloor p_i^t - x_i^t \right\rfloor + c_2R_2\left\lfloor rep_h^t - x_i^t \right\rfloor \quad (29)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (30)$$

where v_i^t and x_i^t show the current velocity of the *i*-th particle and position of the *i*-th particle, respectively. R_1 and R_2 are uniform random numbers between [0, 1]. The parameters c_1 and c_2 represent the personal learning confident and global learning confident, respectively. The inertia weight is considered as *w*. Also, p_i^t is the best experience of the *i*-th particle and rep_h^t Represents the best nominated repository member that is chosen by the roulette wheel selection method. The major structure of the MOPSO is represented in Figure 4.

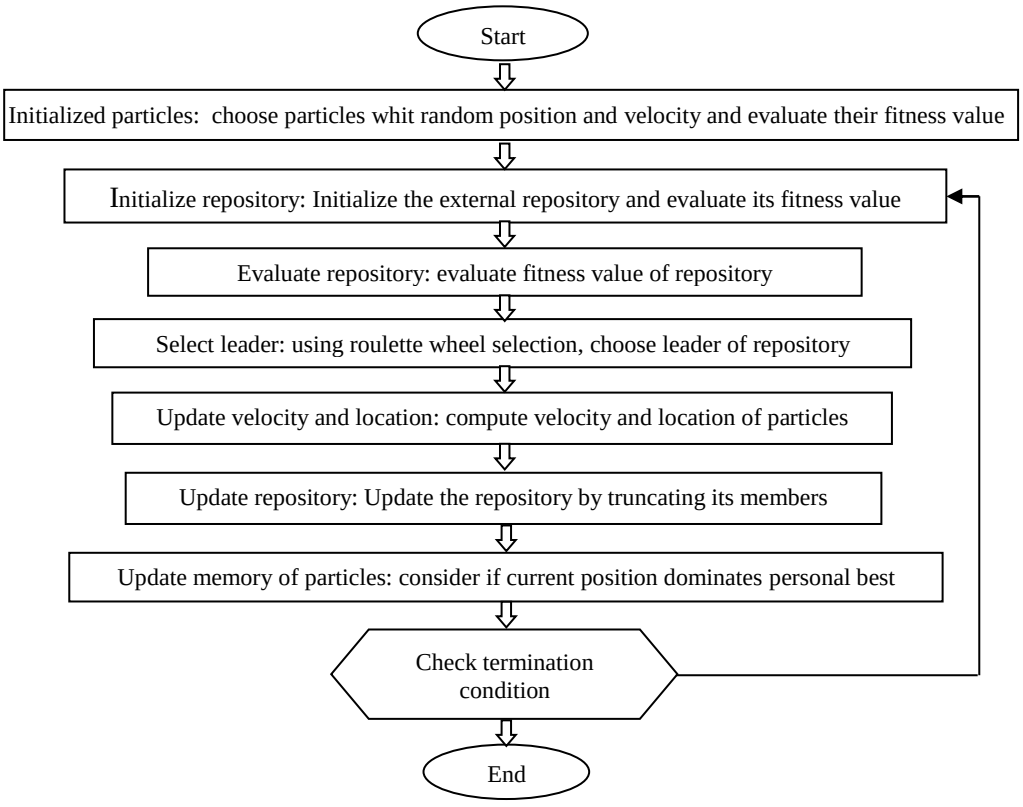


Figure 4. Flow chart of the proposed MOPSO

4-4- Parameter tuning

In this section, we tune some NSGA-II parameters, such as a number of solutions in the initial population (*Npop*), crossover probability (*pc*) and mutation probability (pm). The The Taguchi method is applied to design experiments. The experiment's factors are *Npop*, *pc*, *pm*. three levels are considered for each factor and MINITAB® software is used to design the experiment and analyze its result. The medium-sized problem is used to tune the parameters of NSGA-II. The results of experiments and analysis of the parameters are shown in Table 2.

Based on the analysis of the Taguchi design, we can find out which amount of parameters is appropriate and lead to better results. The abstracts of diagrams and results of this analysis are summarized in Table 3.

Table 2. Taguchi design of the experiment and results for NSGA-II

Experiment	Parameters			*Pareto front*
	Npop	*Pc*	*Pm*	
1	75	0.4	0.4	100
2	75	0.5	0.5	100
3	75	0.6	0.6	96
4	100	0.4	0.5	100
5	100	0.5	0.6	100
6	100	0.6	0.4	100
7	125	0.4	0.6	100
8	125	0.5	0.4	94
9	125	0.6	0.5	87

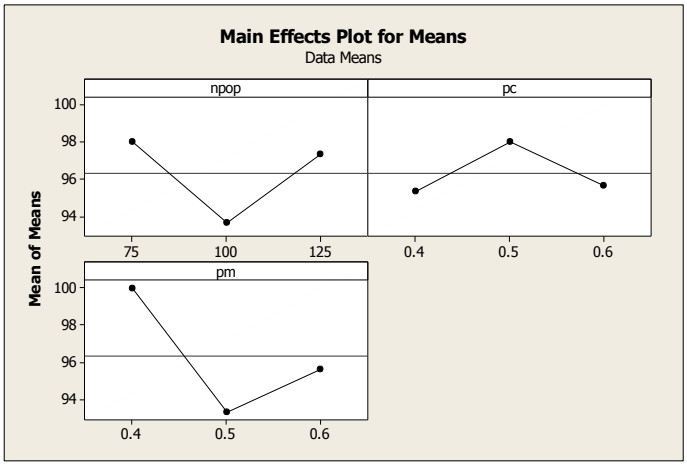


Figure 5. Analysis of the Taguchi design for the NSGA-II parameters

Table 3. Results of parameters tuning for NSGA-II

Parameters		
Npop	*Pc*	*Pm*
75	0.5	0.4

For the MOPSO algorithm, we consider the number of experiments in the personal learning confident (c_1), global learning confident (c_2), Interia weight (*w*), repository size (*nRep*), and interia weight damping rate (*wdap*) as the MOPSO parameters to tune this parameter. We apply the Taguchi method to design the experiment. The medium-sized

problem is used to tune the parameters of MOPSO. The results of experiments and analysis of parameters are shown in Table 4. The appropriate amount of parameters that lead to better results from the analysis of the Taguchi method is represented in Table 5.

Table 4. Taguchi design of the experiment and results for MOPSO

Experiment	Parameters					Number of repository
	c_1	c_2	*w*	*nRep*	*wdap*	
1	0.5	0.5	0.2	20	0.99	20
2	0.5	0.5	0.2	20	0.98	19
3	0.5	0.5	0.2	20	0.97	18
4	0.5	1	0.3	25	0.99	25
5	0.5	1	0.3	25	0.98	25
6	0.5	1	0.3	25	0.97	24
7	0.5	1.5	0.4	30	0.99	30
8	0.5	1.5	0.4	30	0.98	23
9	0.5	1.5	0.4	30	0.97	30
10	1	0.5	0.3	30	0.99	13
11	1	0.5	0.3	30	0.98	30
12	1	0.5	0.3	30	0.97	24
13	1	1	0.4	20	0.99	20
14	1	1	0.4	20	0.98	20
15	1	1	0.4	20	0.97	17
16	1	1.5	0.2	25	0.99	25
17	1	1.5	0.2	25	0.98	24
18	1	1.5	0.2	25	0.97	22
19	1.5	0.5	0.4	25	0.99	25
20	1.5	0.5	0.4	25	0.98	24
21	1.5	0.5	0.4	25	0.97	6
22	1.5	1	0.2	30	0.99	13
23	1.5	1	0.2	30	0.98	30
24	1.5	1	0.2	30	0.97	30
25	1.5	1.5	0.3	20	0.99	20
26	1.5	1.5	0.3	20	0.98	19
27	1.5	1.5	0.3	20	0.97	18

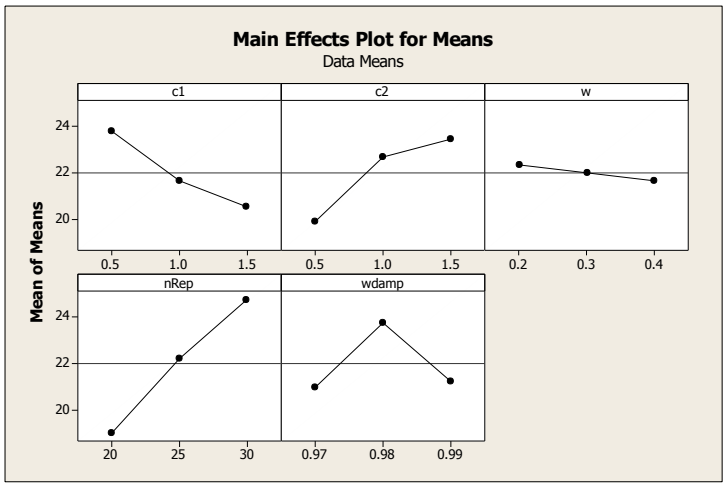


Figure 6. Analysis of the Taguchi design for MOPSO parameters

Table 5. Results of parameters tuning for MOPSO

Parameters				
c_1	c_2	w	*nRep*	*wdap*
0.5	1.5	0.2	30	0.98

5 Numerical experiments

In this section, 15 numerical samples of a real case in Tehran, the capital city of Iran, are considered to illustrate how the proposed model works and to validate both the feasibility and applicability of our model. Table 6 presents data and results of these numerical test problems. We apply NSGA-II and MOPSO to solve these problems. These proposed algorithms are coded in MATLAB® R2013b and run on Intel Core i7 1.60 GHz personal computers with 4 GB RAM. Each test problem runs five times and averages of these results are given in Table 6. This table mentioned the average value of the three objective function of Pareto front and average of running time on each test problem.

Table 6. Considered test problems.

No	Problem info	MOPSO algorithm				NSGA-II algorithm			
	$N \times J \times K$	Z_1	Z_2	Z_3	CPU time (m)	Z_1	Z_2	Z_3	CPU time (m)
1	$5 \times 21 \times 10$	1300	65.64	0.5579	2.18	1150	72.61	0.6563	4.86
2	$5 \times 21 \times 20$	2875	103.86	0.4362	3.28	2375	98.34	0.4291	6.34
3	$10 \times 21 \times 20$	3900	144.31	0.4412	5.46	3325	123.02	0.4846	10.12
4	$10 \times 21 \times 30$	6050	225.91	0.558	9.46	6050	225.91	0.5541	16.61
5	$15 \times 42 \times 30$	9800	230.15	0.3578	12.2	9025	241.93	0.4381	21.83
6	$15 \times 42 \times 40$	21525	625.64	0.4988	19.36	18350	567.6	0.4516	36.7
7	$20 \times 42 \times 40$	35000	610.83	0.4594	27.8	36200	728.74	0.566	45.18
8	$20 \times 42 \times 50$	51300	828.34	0.4065	36.7	48300	794.29	0.4428	54.69
9	$25 \times 63 \times 50$	96500	2469.4	0.4125	47.25	81050	2300.36	0.6386	78.03
10	$25 \times 63 \times 60$	132375	3697.16	0.498	61.3	145450	3310.54	0.694	86.82
11	$30 \times 63 \times 60$	202300	3678.85	0.3829	75.03	173800	3921.22	0.6056	113.1
12	$30 \times 63 \times 70$	345750	4832.3	0.5595	81.28	298650	4148.76	0.4703	128.17
13	$40 \times 84 \times 80$	685600	9657.13	0.5148	96.54	606600	10035.17	0.5544	162.62
14	$40 \times 84 \times 90$	740200	10134.54	0.5936	110.46	767325	9843.61	0.3877	179.4
15	$50 \times 84 \times 100$	913500	17509.39	0.595	127.2	846700	16310.09	0.6672	194.08

Because of illustrating the performance of these algorithms, the results of the NSGA-II and MOPSO algorithms for all problems in Table 6 are compared with each other. The result of NSGA-II and MOPSO are compared with the *t*-test. CPU time and every objective function are considered for the *t*-test. The data related to four factors containing an average of completion time, an average of the first objective function (Z_1), the average of the second objective function (Z_2), and average of the third objective function (Z_3) for Pareto front and the repository solution for each test problem mentioned above from the historical data on a real case of Tehran. First, the normality assumption is investigated and the results show that the output of the algorithm follows a normal distribution. The equality of means ($H_0: \mu_1 = \mu_2$) is considered as the test of two-sample *t*-test. Before the *t*-test, the equality of variances ($H_0: \sigma_1^2 = \sigma_2^2$) has also been checked by the *F*-test. The results of the ANOVA are summarized in Table7. It is concluded that the performance of NSGA-II is equal to MOPSO in all considered factors except the running time. Based on these results, MOPSO performs better in terms of the computational time than the NSGA-II algorithm. However, since NSGA-II can search more solution space, this higher running time is logical.

Table 7. Analysis of variance for comparison between NSGA-II and MOPSO

Factor	*F*-test (equality of variances)			*t*-test (equality of means)		
	F-value	*p*-value	$H_0 : \delta_1^2 = \delta_2^2$	*t*-value	*p*-value	$H_0 : \mu_1 = \mu_2$
Z_1	1.11	0.847	Accepted at α=0.05	0.12	0.903	No Rejected at α=0.05
Z_2	1.10	0.866	Accepted at α=0.05	0.08	0.939	No Rejected at α=0.05
Z_3	0.60	0.358	Accepted at α=0.05	-1.57	0.129	No Rejected at α=0.05
CPU time	0.41	0.106	Accepted at α=0.05	-3.41	0.03	Rejected at α=0.05

6 Sensitivity analysis

In this section, the several parameters used in this proposed model are considered to test the sensitivity of these parameters on the results and objective functions. An important issue that must be addressed is that if the learning effect has an important impact on the other objective or not. The historical data from a real case considered as a case, in which there is no any learning effect. We formulate nurse scheduling without the learning effect and cross-training by removing the related constraints and variables. This model is solved for five times with two mentioned algorithms.

Different values of β (i.e., learning index) are considered to examine its influence on all the objective functions. The results show that the third objective function is very sensitive to learning effects. On the other hand, the first objective function (cost) is worse than the stated learning effect applied. According to Constraint (14), it is reasonable when the learning effect considers the time when each nurse must be spent in each bed and in each home. So this effect results in some homes that need fewer nurses to respond to the demands of the room. Figure 7 represents this fact that if there is no learning effect in the model, the utilization is less than when the learning effect is considered. Figure 8 shows the learning effect on the first objective function. The second objective does not have much sensitive (Figure 9). Therefore, the cost function and utilization can improve by the learning effect.

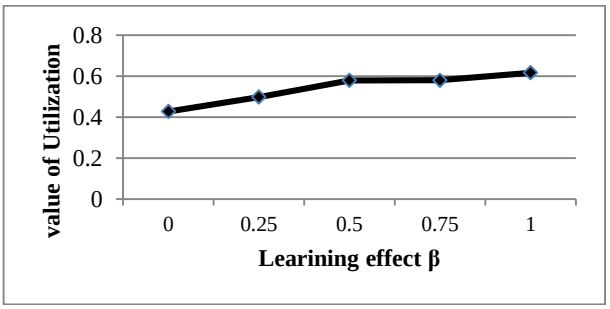


Figure 7. Changes in the utilization value versus changes in the value of learning index β

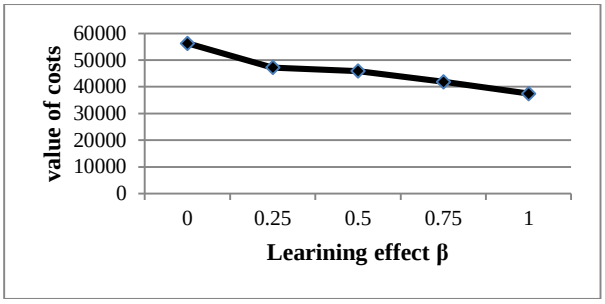


Figure 8. Changes in the value of the first objective versus changes in the value of learning index β

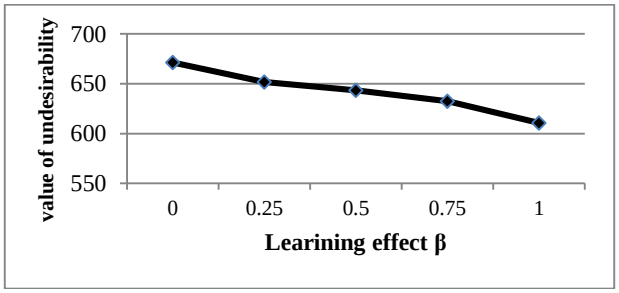


Figure 9. Changes in the value of the second objective versus changes in the value of learning index β

6 Conclusion

This study is an initial work on a cross-trained nurse scheduling problem under the learning effect. Our model has three objective functions that minimize the nurses cost and training cost, minimizes undesirability and maximizes utilization. The NSGA-II and MOPSO algorithms are applied to solve our model. Some problems in different sizes taken from a real case have been considered to compare between these two algorithms. We have considered our model with the learning effect versus without learning effect to illustrate the influence of this factor on the effectiveness. Considering both of cross-training and learning effect concepts can yield a good way to solve the given problem with the nursing shortage. As a direction for the future framework, maximizing the cross-training level can be utilized as another objective. In addition, in order to apply the learning effect, a linear learning function is used. As future work, a non-linear learning function can be used as these functions are more efficient and useful for real problems in the world. Moreover, the future researches can focus on uncertainty circumstance for demand.

References

1. Lilleby, H. E. S., Schittekat, P., Nordlander, T. E., Hvattum, L. M., & Andersson, H. (2012). Competence building with the use of nurse re-rostering. Lecture Notes in Management Science, 4, 70-77.
2. Wright, P. D., & Bretthauer, K. M, Strategies for addressing the nursing shortage: coordinated decision making and workforce flexibility, Decision Sciences, 41(2), 373-401 (2010)

3. Gnanlet, A., & Gilland, W. G, Sequential and simultaneous decision making for optimizing health care resource flexibilities, Decision Sciences, 40(2), 295-326 (2009)
4. Cheang, B., Li, H., Lim, A., & Rodrigues, B, Nurse rostering problems: a bibliographic survey,European Journal of Operational Research, 151(3), 447-460(2003)
5. Burke, E. K., De Causmaecker, P., Berghe, G. V., & Van Landeghem, H, The state of the art of nurse rostering, Journal of scheduling, 7(6), 441-499 (2004)
6. Lankshear, A. J., Sheldon, T. A., & Maynard, A, Nurse staffing and healthcare outcomes: a systematic review of the international research evidence, Advances in Nursing Science, 28(2), 163-174 (2005)
7. Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B, A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem, European Journal of Operational Research, 188(2), 330-341 (2008)
8. Causmaecker, P.D., & Berghe, G. V, A categorisation of nurse rostering problems, Journal of Scheduling, 14(1), 3-16 (2011)
9. Martinelly, C.D., Baptiste, P., & Maknoon, M. Y, An assessment of the integration of nurse timetable changes with operating room planning and scheduling, International Journal of Production Research, 52(24), 7239-7250 (2014)
10. Wang, W. Y., & Gupta, D, Nurse absenteeism and staffing strategies for hospital inpatient units, Manufacturing & Service Operations Management, 16(3), 439-454 (2014)
11. Ko, Y. W., Uhmn, S., & Kim, J, Nurse scheduling problem using approximation algorithms with cost bit matrix, Life Science Journal, 11(7) (2014)
12. Kim, S. J., Ko, Y. W., Uhmn, S., & Kim, J, A strategy to improve performance of genetic algorithm for nurse scheduling problem, International Journal of Software Engineering & its Applications, 8(1) (2014)
13. Wong, T. C., Xu, M., & Chin, K. S, A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department, Computers & Operations Research, 51, 99-110 (2014)
14. Santos, H. G., Toffolo, T. A., Gomes, R. A., & Ribas, S, Integer programming techniques for the nurse rostering problem, Annals of Operations Research, 1-27 (2014)
15. Drake, R. G, The nurse rostering problem: from operational research to organizational reality?, Journal of Advanced Nursing, 70(4), 800-810 (2014)
16. Zheng, Z., & Gong, X, Chemical reaction optimization for nurse rostering problem, Frontier and Future Development of Information Technology in Medicine and Education, 3275-3279 (2014)
17. Burke, E. K., & Curtois, T, New approaches to nurse rostering benchmark instances, European Journal of Operational Research, 237(1), 71-81 (2014)
18. Flinkman, M., Leino-Kilpi, H., & Salanterä, S, Nurses' intention to leave the profession: integrative review. Journal of Advanced Nursing, 66(7), 1422-1434 (2010)
19. Hayes, C., Ponte, P. R., Coakley, A., Stanghellini, E., Gross, A., Perryman, S., & Somerville, J, Retaining oncology nurses: strategies for today's nurse leaders, In Oncology Nursing Forum, 32(6), 1087-1090 (2005)
20. Chan, C. W., & Perry, L, Lifestyle health promotion interventions for the nursing workforce: a systematic review, Journal of Clinical Nursing, 21(15-16), 2247-2261 (2012)
21. Toh, S. G., Ang, E., & Devi, M. K, Systematic review on the relationship between the nursing shortage and job satisfaction, stress and burnout levels among nurses in oncology/haematology settings, International Journal of Evidence-based Healthcare, 10(2), 126-141 (2012)
22. Goodin, H.J., The nursing shortage in the United States of America: an integrative review of the literature, Journal of Advanced Nursing, 43(4), 335-343 (2003)
23. Heinz, D, Hospital nurse staffing and patient outcomes: A review of current literature, Dimensions of Critical Care Nursing, 23(1), 44-50 (2004)
24. Lu, H., While, A. E., & Barriball, K. L, Job satisfaction among nurses: a literature review, International Journal of Nursing Studies, 42(2), 211-227 (2005)

25. Ge, H., Wang, Z., & Yin, D, Facing the Challenge of Adapting to a Life 'Alone' and Nursing Shortage among the Empty Nest Elderly in Southwest China. Life Science Journal, 10(3) (2013)
26. Zhu, J., Rodgers, S., & Melia, K. M, The impact of safety and quality of health care on Chinese nursing career decision-making, Journal of Nursing Management, 22(4), 423-432 (2014)
27. Paul, J. A., & MacDonald, L, Modeling the benefits of cross-training to address the nursing shortage, International Journal of Production Economics, 150, 83-95 (2014)
28. Gnanlet, A., & Gilland, W. G, Impact of productivity on cross-training configurations and optimal staffing decisions in hospitals, European Journal of Operational Research, 238(1), 254-269 (2014)
29. Zimmermann, P. G, The hospital occupational health service: Basic emergency nurse cross-training and differences from emergency services, Journal of Emergency Nursing, 22(6), 591-594 (1996)
30. Li, L. L. X., & King, B. E, A healthcare staff decision model considering the effects of staff cross-training, Health Care Management Science, 2(1), 53-61 (1999)
31. Inman, R. R., Blumenfeld, D. E., & Ko, A, Cross-training hospital nurses to reduce staffing costs, Health Care Management Review, 30(2), 116-125 (2005)
32. Alonso, A., Baker, D. P., Holtzman, A., Day, R., King, H., Toomey, L., & Salas, E, Reducing medical error in the Military Health System: How can team training help?, Human Resource Management Review, 16(3), 396-415 (2006)
33. Bard, J. F., & Purnomo, H. W, Preference scheduling for nurses using column generation, European Journal of Operational Research, 164(2), 510-534 (2005)
34. Wright, P. D., & Mahar, S, Centralized nurse scheduling to simultaneously improve schedule cost and nurse satisfaction, Omega, 1042–1052 (2013)
35. Easton, F. F, Cross-training performance in flexible labor scheduling environments, IIE Transactions, 43(8), 589-603 (2011)
36. Maenhout, B., & Vanhoucke, M, An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems, Omega, 41(2), 485-499 (2013)
37. Salas, E., DiazGranados, D., Weaver, S. J., & King, H, Does team training work? Principles for health care. Academic Emergency Medicine, 15(11), 1002-1009 (2008)
38. Punnakitikashem, P., Rosenberber, J. M., & Buckley-Behan, D. F, A stochastic programming approach for integrated nurse staffing and assignment, IIE Transactions, 45(10), 1059-1076 (2013)
39. Park, P, The examination of worker cross-training in a dual resource constrained job shop, European Journal of Operational Research, 51, 291-299 (1991)
40. Wright, T.P., Factors affecting the cost of airplanes, Journal of Aeronautical Sciences 3, 122-128 (1936)
41. Biskup, D., Single-machine scheduling with learning considerations, European Journal of Operational Research, 115(1), 173-178 (1999)
42. Berrada, I., Ferland, J. A., & Michelon, P., A multi-objective approach to nurse scheduling with both hard and soft constraints, Socio-Economic Planning Sciences, 30(3), 183-193 (1996)
43. Gascon, V., Villeneuve, S., Michelon, P., & Ferland, J. A., Scheduling the flying squad nurses of a hospital using a multi-objective programming model, Annals of Operations Research, 96(1-4), 149-166 (2000)
44. Legrain, A., Bouarab, H., & Lahrichi, N., The Nurse Scheduling Problem in Real-Life, Journal of medical systems, 39(1), 1-11 (2015)
45. Burke, E. K., Li, J., & Qu, R., A Pareto-based search methodology for multi-objective nurse scheduling, Annals of Operations Research, 196(1), 91-109 (2012)
46. Yin, P. Y., Chao, C. C., & Chiang, Y. T, Multi-objective optimization for nurse scheduling, Advances in Swarm Intelligence Springer Berlin Heidelberg, 66-73 (2011)

47. Asefi, H., Jolai, F., Rabiee, M., & Araghi, M. T, A hybrid NSGA-II and VNS for solving a bi-objective no-wait flexible flowshop scheduling problem, The International Journal of Advanced Manufacturing Technology, 75(5-8), 1017-1033 (2014)
48. Wang, S., & Liu, M, Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method, International Journal of Production Research, 52(5), 1495-1508 (2014)
49. Azadeh, A., Farahani, M. H., Kalantari, S. S., & Zarrin, M, Solving a multi-objective open shop problem for multi-processors under preventive maintenance. The International Journal of Advanced Manufacturing Technology, 1-16 (2014)
50. Konak A, Coit Q. W, & Smith A. Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9), 992-1007 (2006)
51. Deb K, Pratap A, Agarwal S, and Meyarivan T. A. M. T, A fast and elitist multi-objective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, 6, 182-197 (2002)
52. Kennedy, J., & Eberhart, R, Particle swarm optimization, Proceedings of the IEEE International Conference on Neural Networks, 4, 1942-1948 (1995)
53. Coello, C., Pulido, G., & Lechuga, M., Handling multiple objectives with particle swarm optimization, IEEE Transactions on Evolutionary Computation, 8(3), 256–279 (2004)
54. Moore, J., Chapman, R., & Dozier, G., Multi-objective particle swarm optimization, Proceedings of the 38th Annual on Southeast Regional Conference, 56-57, (2000)

MISTA 2015

A Fixed Route Dial-a-Ride Problem

Hagai Ilani · Elad Shufan · Tal Grinshpoun

Abstract Dial-a-ride (DARP) is a transportation solution with flexible routes and flexible schedules. The DARP challenge is to optimally fulfill a set of pickup and delivery ride requests using a given vehicle fleet. We hereby present a DARP variant for which the route is known in advance. The problem is then to set up the schedules according to the requests. For a given operation cost, the aim is to maximize user satisfaction, by minimizing the sum of passengers' waiting times. We introduce algorithms for solving two variants of the fixed route DARP – one for a fleet of infinite capacity vehicles, and one for the more general case of vehicles with heterogeneous capacities. Contrary to general DARP which is NP-hard, the presented algorithms are polynomial in the number of ride requests.

1 Introduction

Traditional public transportation systems are very effective in highly-populated areas where the networks of buses, trams, metro, etc., provide adequate solutions in various aspects, such as service frequency, proximity of terminals (stations/stops), and price. However, there are scenarios that require other, designated, transportation solutions. These scenarios are broadly divided into two groups – scenarios that stem from special geographic features and scenarios that involve special populations. The first group usually relates to rural, sparsely-populated, areas, while the second group of scenarios involves populations that require door-to-door service, such as elderly, disabled, or children.

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Beer-Sheva, Israel
E-mail: elads@sce.ac.il

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

The solution to the above scenarios takes the form of personalized and designated transportation solutions, broadly termed *Demand Responsive Transportation* (DRT) [14]. A classic type of DRT is the *Dial-a-Ride Problem* (DARP) [5], in which passengers "dial" to request for "rides" between origins and destinations. The ride requests include release and/or due times. A solution to a DARP is a set of vehicle routes and schedules that complies with various constraints, such as fleet size, vehicle capacity, and driving distance, while trying to accommodate as many passenger requests as possible. The classic DARP examples are of door-to-door transportation for elderly or disabled people.

As many vehicle routing problems, DARP is generally NP-hard [13]. The practical meaning of DARP's complexity is that there is no efficient method to optimally solve the problem. One can better understand the origins of DARP's complexity by disassembling it to its two main ingredients:

1. Finding the routes for the vehicles.
2. Grouping together the passengers that share a ride, and determining the pickup and drop-off times (e.g., the schedule).

The first ingredient is actually a compound instance of the famous Travelling Salesman Problem (TSP) [9]. It is compound, since it may include finding the routes for several vehicles that operate in intersecting areas, resulting in a more complex problem than a single TSP. TSP itself is known to be NP-hard [11].

Contrary to that, determining the schedules is usually an "easier" problem to solve. In fact, it was proven that for a DARP with two fixed locations, termed *Two-Campus Transport Problem* (TCTP), the problem is polynomial in the number of passengers [8]. It was shown that for a TCTP an optimal solution can be found efficiently (in matters of seconds or less) for problems with dozens of passengers. Other polynomial solutions were proposed to resembling problems of two-station railway scheduling [7, 12].

The complexity gaps between the two ingredients of DARP lead to the understanding that it may be worthwhile to focus on problems in which the first task of finding the routes is degenerated. While such relaxation of DARP may seem as a considerable limitation, there are many scenarios where the use of fixed routes is acceptable or even essential. The first group of scenarios includes geographical areas that have just a single main road/highway connecting all the places-of-interest. The *Florida Keys* are a famous example of such an area, but there are smaller, local, such areas in almost every country in the world. In many cases, these are rural areas that may benefit from DRT services. The second group of scenarios includes special populations. For example, the elderly population is often characterised in limited geographical dispersion; many elderly people reside in designated housings, such as retirement villages, housing complexes and assisted living residences. Furthermore, they often share similar places-of-interest, such as social clubs, medical centers, shopping centers, and volunteer centers. Another potential advantage of using fixed routes for the elderly population lies in their preference for stability; elderly people often dislike substantial alterations, so a transportation solution that enables them to travel through known routes, has greater implementation chances in this population than standard DARP solutions.

Whether for geographic reasons or for reasons relating to special populations, the *Fixed Route Dial-a-Ride Problem* (FRDARP) is an important real-life problem that has not received attention in past research. Although more specific than general DARP, FRDARP itself has many variation and considerations, such as the shape of the (fixed) route, characteristics of the vehicle fleet, and objective functions. The various consid-

erations of the model are thoroughly described in Section 2. Algorithmic solutions to two real-life FRDARP scenarios are presented in Section 3. The proposed algorithms, both relying on a reduction [3] to the shortest path problem [6], are polynomial in the number of ride requests. A discussion in Section 4 concludes the paper.

2 Model Considerations

In this section we relate to various considerations of FRDARP. Some of these consideration are relevant to any general DARP.

Terminals: Each ride request is a demand for a transportation from a pickup terminal to a delivery terminal, both are chosen from a set of L terminals: $\{A_0, A_1, \ldots, A_{L-1}\}$.

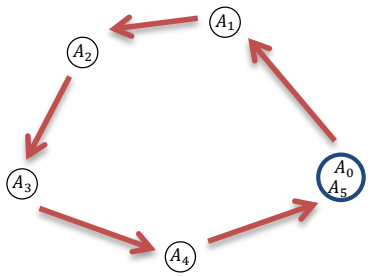
Pickup and delivery times: A passenger requests to be picked from terminal A_i at time t and delivered to a terminal A_j at time t'. The difference between pickup and delivery times, $\Delta t = t' - t$ is expected to be as minimal as possible. In the case of a taxi Δt equals the traveling time between the two terminals. In DARP several passengers (possibly with different requests) share a ride, and therefore Δt may be longer than the traveling time. The quality of service is related to the deviation between the requested departure or arrival times, and the actual scheduled times.

Request types: Some passengers have a rigid constraint only over the delivery time; it is not optional for them to arrive to the delivery terminal later than some deadline t_d. For example, a passenger may have a doctor's appointment, hence should arrive at the corresponding terminal no later than t_d, otherwise she will be late. Requests of this character are called *s-type* requests, since they relate to the *starting* time of the passenger's activity. A different possibility is when the pickup time must not be earlier than some release time t_r. For example, if the passenger is expected to finish a meeting at t_r and requests for a ride back home, then the pickup time cannot be earlier than t_r. Such a request is termed *r-type* request, as it relate to the *returning* time of the passenger, after the activity has been completed. The s-type and r-type requests are known also as outbound and inbound requests, respectively. Both request types can be described by the more general request type for which the costumer determines a *time window*, which can refer to the pickup time or to the delivery time. Other types of requests may be considered. For example, in a round-trip request, from terminal A to terminal A', and then back to A, the passenger might want to specify only the time spent in A' (A' might be a shopping center being visited on the passenger's day off).

Vehicle fleet: All the ride requests should be fulfilled by operating a given vehicle fleet with M vehicles, which may generally have different speed, capacity, disabled service facilities, etc. We distinguish between a fleet of *finite-capacity* vehicles and between the case for which the capacity is not an issue, hence taken as *infinite*. The infinite capacity assumption considerably simplifies the algorithmic solution of the corresponding DARP. We use the term *homogeneous fleet* to describe a fleet of identical vehicles. A fleet with vehicles that are identical with respect to all characteristics, except for their capacity, is referred to as a *heterogeneous fleet*.

Routes: A major DARP issue concerns the problem of determining the vehicle routes.

In a TCTP [8], which is a two-terminal DARP, it was shown that an optimal polynomial solution exists. The problem of route determination makes the general DARP NP-hard. In this paper we generalize the TCTP to a FRDARP with L terminals, by
a general DARP every vehicle
tial set of terminals. The visited
d by the specific solution. Con



(b)

Fig. 1 FRDARP with (a) a line route, and (b) a circular route.

Both the circular and the line routes may have variants. For example, one can consider a circular route with both clockwise and anticlockwise travel directions. A possible line route variant is of a transport that returns to the main depot immediately after reaching the second depot A_{L-1}, without stopping on the way back. An example for such a scenario is a transportation arrangement taking workers back home after they have finished their shifts. Another possible line route variant is of a transport that waits at the second depot A_{L-1} before commencing the way back to the main depot. TCTP is an example of such a scenario [8].

Objective function: A DARP solution is in its nature a compromise between operating costs and quality of service. A standard goal in DARP is to minimize a combined function of cost and inconvenience, which is related to both the ride durations and

the deviations from desired departure/arrival times. The strategy we take in this article is to maximize user satisfaction for a given cost. Hereinafter, the cost simply corresponds to the number of transports, where a transport is a single vehicle that completes the route once. (The cost of a transport is assumed to be independent of the executing vehicle, even though the vehicles might have different capacities.) The route can be repeated a number of times, by several vehicles. Enlarging the number of transports increases the cost and usually decreases the deviation between the requested departure/arrival times and the actual scheduled times. This, in turn, increases user satisfaction. Because the route is fixed, the users know in advance the total ride duration (assuming all vehicles have the same speeds). As a consequence, the departure time deviation is equal to the arrival time deviation. The objective function is some chosen function of these passengers' waiting times.

Terminal waiting: Is it allowed for a vehicle to wait idly at a terminal, even if it carries passengers that have not yet arrived at their destinations? Despite the fact that allowing this option can improve the schedule, it may lead to customer objections. Even if the costumers rationally know that the possibility of waiting at a terminal improves the schedule (at least on the average), emotionally, when they actually wait, there is a good chance that they will become frustrated. In addition, enabling the possibility of terminal waiting considerably expands the search space, which may hinder the applicability of corresponding algorithmic solutions.

Depot and working hours: Is there a single depot or multiple depots? In the case of TCTP [8] it was assumed that vehicles start and end their working day at one depot (at times t_0 and t_f, respectively). However, in the general case it makes sense to allow each vehicle to have its own depots and working hours. Thus, the problem's input should include a list consisting of vehicles' working hours and starting and ending depots.

Each of the above considerations influences the nature of the solution and its complexity. In the following section we consider:

- One-directional circular route with a single depot.
- Two types of vehicle fleets: an infinite capacity fleet and a heterogeneous finite capacity fleet.
- Ride requests which are either s-type or r-type.
- Waiting at terminals is not allowed (except at the depot). Each departure time therefore determines the corresponding arrival time.
- The objective function is the sum of all passengers' waiting times. Our goal is to minimize this function for a fixed number of transports.

3 Algorithmic Solutions

The input for the circular-route DARPs, which are considered in this section, include:

- N requests for rides between pairs of terminals. The r-type or s-type requests are grouped into sets $R^{i,j}$ and $S^{i,j}$, respectively. The indices i, j are terminal indices, with $0 \leq i < j \leq L$. For example, $S^{1,2} = \{s_1^{1,2}, s_2^{1,2}, \ldots, s_{|S^{1,2}|}^{1,2}\}$ is a list of s-type requested departure times from terminal A_1 to terminal A_2. Without loss of generality we assume $s_1^{i,j} \leq s_2^{i,j} \leq \ldots \leq s_{|S^{i,j}|}^{i,j}$. Note that although $s_n^{i,j}$ might

equal $s_{n+1}^{i,j}$ (two different passengers requesting a similar request) the departure times are considered as two distinguished elements of $S^{i,j}$.
- M vehicles with capacity C_m, $m = 1, 2, \ldots, M$. Vehicle m starts working at $t_{0,m}$ and finishes working at $t_{f,m}$.
- D_{ij} is the traveling time between terminals A_i and A_j. We assume it is stationary (i.e., constant during different daily hours) and independent of vehicle type. The traveling time of a transport, D_{0L} is also denoted D.
- Each transport starts and ends at the same depot. The depot is denoted A_0 when considered as a source terminal, or A_L, when considered as a destination terminal.
- K is the total number of transports.

For each transport the schedule determines which vehicle operates the transport, the departure time, and the passengers that are served by that transport. A solution is therefore composed of the triplet $(\mathcal{P}, \mathcal{V}, \mathcal{T})$, where:

- $\mathcal{P} = (p_1, p_2, \ldots, p_K)$ is a partition of the set $\bigcup_{0 \le i < j \le L} (S^{i,j} \cup R^{i,j})$ into K disjoint sets. The partition part p_i represent the ith transport. A partition usually refers to a set $\{1, 2, \ldots, N\}$. Nevertheless, for notation simplicity we hereinafter relate to the partition of the departure-time requests.
- $\mathcal{V} = (v_1, v_2, \ldots, v_K)$ is a list of vehicle indices. Each vehicle is numbered from 1 to M and, according to $\mathcal{V}$, the ith transport is operated by vehicle number v_i.
- $\mathcal{T} = (t_1, t_2, \ldots, t_K)$ is a list of departure times. The ith transport departs from A_0 at t_i.

A solution $(\mathcal{P}, \mathcal{V}, \mathcal{T})$ has to be feasible. The feasibility conditions include:

1. Partitioning constraints due to different request types: On the one hand, s-type requests determine a deadline for the departure time. On the other hand, r-type requests determine a release time. The combination of the two types limits the partition possibilities.
2. Capacity constraints: The passengers that are grouped into the ith transport are served by vehicle v_i. The number of passengers at any given moment cannot exceed C_i, the vehicle's capacity.
3. Time constraints: Each partition part determines a time window for the departure time – the vehicle has to depart early enough in order to satisfy all the s-type requests, but cannot depart too early in order to accommodate all the r-type requests. In addition to that, each vehicle is restricted to work between $t_{i,0}$ and $t_{i,f}$ $(i = 1, 2, \ldots, M)$, and can only serve one transport at a time.

The feasibility conditions are thoroughly discussed in the next subsections. An *optimal* schedule is a feasible schedule that minimizes a chosen objective function. Here, the objective function is the sum of passengers' waiting times. We show that the problem of finding the optimal solution is polynomially solvable. The algorithm is polynomial in the number of requests, for a given number of terminals, vehicles and transports. It is based on reducing the problem to a shortest path problem, similarly to the TCTP reduction [8]. A weighted graph is constructed, with arcs corresponding to transports; a shortest path that consists of at most K arcs (in the constructed graph) corresponds to an optimal schedule.

In Sections 3.1 and 3.2 we present detailed algorithmic solutions to two respective scenarios – FRDARP with an infinite capacity fleet and FRDARP with a heterogeneous fleet. The description of both algorithms is completed with a theoretic analysis of the possible set of departure times (Section 3.3) and a complexity analysis (Section 3.4).

3.1 Infinite capacity fleet

The case of infinite capacity is presented for two reasons. First, in many real-life scenarios vehicle capacity is not an issue, i.e., when the vehicle's capacity is substantially larger than the number of expected passengers. Accordingly, the removal of capacity constraints may result in improved ability to implement larger instances. The second reason is didactic – it is a good starting point for presenting a basic variation, before advancing to the more general case of heterogeneous vehicles with finite capacities.

From an algorithmic point of view, the specific source and destination terminals (in passenger requests) are not important in the infinite capacity scenario; vehicles do not wait at terminals, so a request to depart from A_i (to any terminal A_j) at time t, is equivalent to a request to depart from A_0 at time $t - D_{0i}$. Hence, we define $\bar{S}^{i,j} = \{s_1^{i,j} - D_{0i}, s_2^{i,j} - D_{0i}, \ldots\}$. The set of all s-type requests is given by $S = \cup_{i<j} \bar{S}^{i,j} = \{s_1, s_2, \ldots, s_{N_s}\}$, where $N_s = |S|$ is the total number of s-type requests. Without loss of generality we assume $s_1 \leq s_2 \leq \ldots \leq s_{N_s}$. We similarly define $\bar{R}^{i,j}$, R, and $N_R = |R|$. The optimal solution which is found by the reduction algorithm is such that if the transport of $s \in S$ departs (from A_0) at t, and the transport of $s' \in S$ departs at t', then $s < s' \Rightarrow t \leq t'$. A solution with this property is called an S-ordered solution. The obtained solution is also R-ordered (similarly defined). In the TCTP study it was explicitly proven that an SR-ordered solution always exists [8]. The proof is also valid for the FRDARP case.

Graph's nodes and arcs: A node is a sequence $(h, l; \tau_1, \tau_2, \ldots, \tau_M)$, also denoted $(h, l; \boldsymbol{\tau})$. This node indicates that the requests $\{r_1, r_2, \ldots, r_h\}$ and $\{s_1, s_2, \ldots, s_l\}$ were already handled, and that vehicle number m is available for the next transport from time τ_m $(m = 1, 2, \ldots, M)$. An arc between two nodes represents a possible transport: $(h, l; \boldsymbol{\tau})$ connected to $(h', l'; \boldsymbol{\tau'})$ represents a transport shared by $\{r_{h+1}, \ldots, r_{h'}\}$ and $\{s_{l+1}, \ldots, s_{l'}\}$. If the transport is applied by vehicle number m then $\boldsymbol{\tau'}$ differs from $\boldsymbol{\tau}$ only by the mth component τ'_m, which implies that vehicle m will be ready to the next transport at τ'_m. The weight of an arc equals to the total waiting time of the passengers in the considered transport. It depends on the departure time. Next, we discuss the feasibility conditions and departure time considerations.

SR-constraints: The transport $(h, l; \boldsymbol{\tau}) \rightarrow (h', l'; \boldsymbol{\tau'})$ cannot depart earlier than the latest r-type request in that transport, $r_{\texttt{last}} = r_{h'}$. In addition, it cannot depart after the earliest s-type request, $s_{\texttt{first}} = s_{l+1}$. Therefore, a transport is feasible only if

$$r_{\texttt{last}} \leq s_{\texttt{first}} \tag{1}$$

In the graph construction we consider only arcs which represent transports that are feasible with respect to this SR-constraint.

Departure time and availability: The earliest time of departure is given by $t_{\texttt{min}} = \max\{r_{\texttt{last}}, \tau_m\}$. We therefore have a feasibility condition which is stronger than Condition 1:

$$t_{\texttt{min}} \leq s_{\texttt{first}} \tag{2}$$

The departure time depends on the total number of r-type requests, $n_r = h' - h$, compared to the total number of s-type requests in the considered transport, $n_s = l' - l$. If $n_r \geq n_s$ then $t_{\texttt{depart}} = t_{\texttt{min}}$. In that case, departing at a later time will not decrease

the total waiting times of that transport; it also means that the vehicle's availability for the next transports will be impaired, which is clearly undesirable. However, if $n_r < n_s$, the situation is different. On the one hand, the total waiting times of this transport will be lowered by departing as late as possible (but not later than $s_{\texttt{first}}$). On the other hand, when departing at a later time the vehicle will be less available for the next transports, which in turn may increase the waiting times of the next transports or even render them inapplicable (following their respective Condition 2). Hence, if $n_r < n_s$ then $t_{\texttt{min}} \leq t_{\texttt{depart}} \leq \min\{s_{\texttt{first}}, t_{f,m} - D\}$. According to Theorem 1 (Section 3.3), within this interval, the departure times that should be considered are picked from a *discrete* set. For each departure time, the weight is easily calculated. The choice of $\boldsymbol{\tau'}$ also depends on the selected departure time of the considered transport: $\tau'_m = t_{\texttt{depart}} + D$.

Graph construction and the reduction: The graph is constructed dynamically. The source node is $(0, 0; t_{0,1}, t_{0,2}, \ldots, t_{0,M})$. All the feasible nodes are constructed as described above, according to the SR-constraints and the departure time and availability considerations. The procedure is repeated iteratively. An optimal solution is obtained by finding a shortest path that consists of at most K arcs, which starts at the source node, and ends at one of the destination nodes $(N_R, N_S; \tilde{\tau_1}, \tilde{\tau_2}, \ldots, \tilde{\tau_M})$, with $\tilde{\tau_j} \leq t_{f,m}$, for $m = 1, 2, \ldots, M$. The construction herein broadly follows the respective construction procedure for TCTP [8]. The reduction is based on the one proposed by Chakravarty *et al.* [3] for the more general problem of set partitioning.

3.2 Heterogeneous fleet

The heterogeneous fleet scenario complexifies the problem by adding capacity constraints to all the considerations of the infinite fleet scenario. Now, the graph should represent not just the requested departure times but also the corresponding source and destination terminals of each request. For the partition of S and R of an infinite capacity fleet (Section 3.1), two indices (h, l) were sufficient to represent all the relevant partitions. Here, the partition is related to $(L-1)\cdot(L+2)$ sets[1]: $S^{i,j}$ and $R^{i,j}$ with $0 \leq i < j \leq L$. A node therefore has $(L-1)\cdot(L+2)+M$ coordinates. The reduction will lead to a solution which is S-ordered and R-ordered only within each set $S^{i,j}$ and $R^{i,j}$. This is in contrast to the infinite capacity scenario for which the SR-order relates to the sets S and R of all the requested departure times, regardless of the specific terminals.

Graph's nodes and arcs: We define two vectors with integer components: $\mathbf{h} = (h^{i,j})$ and $\mathbf{l} = (l^{i,j})$, with $0 \leq i < j \leq L$. Written explicitly, $\mathbf{h} = (h^{0,1}, h^{0,2}, \ldots, h^{0,L-1}, h^{1,2}, \ldots, h^{1,L}, \ldots, h^{L-1,L})$. A node is the sequence $(\mathbf{h}^\frown\mathbf{l}; \boldsymbol{\tau})$.[2] This node indicates that all the requests $\{r_1^{i,j}, r_2^{i,j}, \ldots, r_{h^{i,j}}^{i,j}\}$ and $\{s_1^{i,j}, s_2^{i,j}, \ldots, s_{l^{i,j}}^{i,j}\}$, for the respective i and j, have been handled. The vehicle availability issues are exactly the same as in the infinite capacity scenario, and are thus not repeated. An arc connecting $(\mathbf{h}^\frown\mathbf{l}; \boldsymbol{\tau})$ and $(\mathbf{h'}^\frown\mathbf{l'}; \boldsymbol{\tau'})$ represents a transport shared by $\{r_{h^{i,j}+1}^{i,j}, \ldots, r_{h'^{i,j}}^{i,j}\}$ and $\{s_{l^{i,j}+1}^{i,j}, \ldots, s_{l'^{i,j}}^{i,j}\}$, for all relevant pairs of i and j.

[1] It is $(L-1)\cdot(L+2)$ rather than $L\cdot(L+1)$ since requests from A_0 to A_L make no sense and are thus not considered.

[2] $^\frown$ is used as a concatenation sign.

Capacity constraints: The number of passengers present on the vehicle at any given moment is restricted by the vehicle's capacity. Consider a transport represented by $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau}) \to (\mathbf{h'}^\frown\mathbf{l'};\boldsymbol{\tau'})$, where $\boldsymbol{\tau}$ differs from $\boldsymbol{\tau'}$ by the mth component (i.e., vehicle m is the transporting vehicle). Consider the vector $\Delta\mathbf{h} = (\Delta h^{i,j})$, with $0 \le i < j \le L$, which is defined by $\Delta\mathbf{h} = \mathbf{h'} - \mathbf{h}$. The component $\Delta h^{i,j}$ is equal to the number of r-type passengers requesting for a ride from A_i to A_j. Similarly, we define $\Delta\mathbf{l} = (\Delta l^{i,j}) = \mathbf{l'} - \mathbf{l}$ for s-type passengers. The total number of passengers requesting for a ride between A_i and A_j is given by $\Delta\mathbf{n} = \Delta\mathbf{h} + \Delta\mathbf{l}$. The capacity constraint is then given by:

$$n_i \le C_m, \quad i = 0, 1, \ldots, L-1 \tag{3}$$

where $n_i = \sum_{p=0}^{i} \sum_{q=i+1}^{L} \Delta n^{p,q}$ is the number of passengers that are on the way from terminal A_i to the successive terminal A_{i+1}.

SR-constraints, departure time and availability: The only difference between the infinite capacity scenario and the heterogeneous one lies in the calculation of $r_{\texttt{last}}$, $s_{\texttt{first}}$, n_r and n_s (in case the sets S and R are no longer defined). Here,

$$r_{\texttt{last}} = \max_{i<j}\{r^{i,j}_{h'^{i,j}} - D_{0i}\}$$
$$s_{\texttt{first}} = \min_{i<j}\{s^{i,j}_{l^{i,j}+1} - D_{0i}\}$$
$$n_r = \sum_{i<j} \Delta h^{i,j}$$
$$n_s = \sum_{i<j} \Delta l^{i,j}$$

3.3 Discrete available times

Follows is an analysis of the optional departure times of vehicles that should be considered. This analysis regards FRDARPs with a circular route, a fleet of M vehicles, sequences of requested times of all the passengers, and a bound of K transports; it is therefore relevant to both the scenarios of the previous subsections. The following theorem states the possible departure times τ_m $(m = 1, 2, \ldots, M)$ that should be considered in the graph's nodes. The analysis follows the building blocks of the respective analysis given for TCTP [8].

Theorem 1 *If an instance of a circular route DARP has a feasible solution, then there always exists an optimal solution in which any of the departure times from A_0, of any vehicle, takes one of the following forms:*

1. $t_{0,m} + nD$, $1 \le m \le M$ *or*
2. $s^{i,j}_l - D_{0,i} \pm nD$, $0 \le i < j \le L$, $1 \le l \le |S^{i,j}|$ *or*
3. $r^{i,j}_h - D_{0,i} \pm nD$, $0 \le i < j \le L$, $1 \le h \le |R^{i,j}|$ *or*
4. $t_{f,m} - (n+1)D$, $1 \le m \le M$

where $n = 0, 1, \ldots, (K-1)$.

Proof In a given solution, a vehicle may start a transport immediately after it has finished one. For the sake of the proof we define a *block of transports* as a sequence of transports which are operated by the same vehicle one after the other without breaks, and is maximal with respect to that property. A block of transports may contain one transport (if that transport does not start immediately after another, and is not followed by an immediate transport of the same vehicle). Each transport in a given solution is a member of a single block.

By definition, transports in a block satisfy the following properties concerning their departure times and end times:

1. The departure times of two consecutive transports in a block differ by D.
2. A vehicle that operates a block of transports starts the block either at the starting time of that vehicle or after a break.
3. When a vehicle completes operating a block of transports it either has a break or is at the finish time of that vehicle.

Increasing (decreasing) the departure times of all the transports in a given block by $\epsilon > 0$ will increase (decrease) by ϵ the waiting time of each of the r-type passengers in the block's transports and will decrease (increase) by ϵ the waiting time of each of the s-type passengers.

Let $(\mathcal{P}, \mathcal{V}, \mathcal{T})$ be an optimal solution with a minimum sum of departure times. We show that this solution satisfies the theorem's assertion. Consider a block with a majority of s-type passengers. Since $(\mathcal{P}, \mathcal{V}, \mathcal{T})$ is optimal, it must hold that the departure time of the transports of that block are constrained by either an s-type request or by the finish time of the vehicle. Therefore, the departure times of that block are either of the form 2 or 4. For a block with equal or less s-type passengers (compared to r-type passengers), decreasing the departure times of its transports will either decrease the total passengers' waiting time or keep the total waiting time unchanged, while decreasing the sum of departure times. Since $(\mathcal{P}, \mathcal{V}, \mathcal{T})$ is an optimal solution with a minimum sum of departure times, the departure times of the transports of that block are constrained by either some r-type request or by the starting time of the vehicle. In this case, the departure times of that block are either of the form 1 or 3. ■

3.4 Complexity

The complexity and efficiency of the algorithms depend on the number of states (nodes), N_{nodes}, which is considered hereby. N_{nodes} depends on the *partition indices*, (h, l) or $(\mathbf{h}^\frown\mathbf{l})$ for the infinite capacity and heterogeneous fleet algorithms, respectively, as well as on the *available times indices*.

Partition indices: There are $(L-1)(L+2) \sim O(L^2)$ possible node coordinates for the heterogeneous fleet algorithm. For each coordinate there are $|S^{i,j}|$ or $|R^{i,j}|$ options, both are bounded by N, the total number of requests. For the infinite capacity algorithm there are just two possible coordinates that describe the possible partitions, each has at most N options.

Available times indices: Following Theorem 1, each of the M availability time coordinates has $2K + N(2K-1)$ possibilities: $2K$ for times of forms 1 and 4 (starting and ending times) and $N(2K-1)$ for times of forms 2 and 3.

The total number of nodes to be considered for the heterogeneous fleet algorithm is therefore $O(N^{L^2}(KN)^M)$. Since K at most equals N, we get $O(N^{L^2+2M})$ as a worst case bound for the number of nodes. Applying the Bellman-Ford algorithm [1] for finding the shortest path that consists of at most K arcs takes $O(N_{arcs}K) \sim O(N_{arcs}N)$, where N_{arcs} is the number of arcs in the graph. Hence, the total run-time complexity of the heterogeneous fleet algorithm is $O(N^{2L^2+4M+1})$. Correspondingly, the total run-time complexity of the infinite capacity fleet algorithm is $O(N^{4M+5})$. Consequently, for a fixed number of terminals (only in the heterogeneous fleet algorithm) and vehicles (in both algorithms), the algorithms are polynomial in the number of ride requests.

4 Discussion

DARP is a popular solution for demand responsive transportation. In many real-life scenarios the routes are inherently fixed, while the schedules are flexible. In this paper we introduced the FRDARP model that represents such scenarios, and discussed various considerations and variants of the model. The limitation to fixed routes considerably reduces the complexity of the problem, which brings with it two perspectives that are rarely considered for general DARPs: (*i*) situating *user satisfaction* as the primary focus of the objective function, and (*ii*) searching for efficient *optimal* solutions (as opposed to non-optimal heuristics).

Indeed, we presented optimal polynomial-time algorithms for two variants of FRDARP with a circular route. The proposed algorithms may also be used for FRDARP with a line route, as long as the vehicles do not wait at the second depot; this can be achieved by applying minor indexing modifications (to reflect the representation differences between a circular route and a line route).

Other variants that stem from the assortment of model considerations are left for future work. It is especially interesting to investigate how different objective functions affect the proposed algorithmic solutions. It is also motivating to consider terminal waiting, since there are some real-life scenarios in which such waiting is acceptable by the passengers (e.g., train connections in major stations). Moreover, terminal waiting may enable the development of *dynamic scheduling* FRDARP, in which schedules may be modified on-the-fly following new received ride requests. (FRDARP without terminal waiting is *static* by nature due to its rigid routes.)

Another prospect for future work is to use the FRDARP algorithmic solutions as part of heuristic solutions for general DARP. Such solutions may consist of two recurring stages: (*i*) heuristically fix a route, and (*ii*) solve the corresponding FRDARP. It will be interesting to compare such a heuristic paradigm with other DARP heuristics, such as tabu search [4,10] and deterministic annealing [2].

References

1. Bellman, R.: On a routing problem. Quarterly of Applied Mathematics **16**, 87–90 (1958)
2. Braekers, K., Caris, A., Janssens, G.K.: Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transportation Research Part B: Methodological **67**, 166–186 (2014)
3. Chakravarty, A.K., Orlin, J.B., Rothblum, U.G.: A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment. Operations Research **30**(5), 1018–1022 (1982). DOI 10.1287/opre.30.5.1018

4. Cordeau, J.F., Laporte, G.: A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological **37**(6), 579–594 (2003)
5. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: models and algorithms. Annals of Operations Research **153**(1), 29–46 (2007)
6. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerische Mathematik **1**, 269–271 (1959)
7. Gafarov, E.R., Dolgui, A., Lazarev, A.: Two-station single track railway scheduling problem with equal speed of trains. In: Book of Abstracts of the 21st International Symposium on Mathematical Programming (ISMP 2012) (2012)
8. Ilani, H., Shufan, E., Grinshpoun, T., Belulu, A., Fainberg, A.: A reduction approach to the two-campus transport problem. J. of Scheduling **17**(6), 587–599 (2014). DOI 10.1007/s10951-013-0348-7. URL http://dx.doi.org/10.1007/s10951-013-0348-7
9. Lawler, E.L., Lenstra, J.K., Kan, A.R., Shmoys, D.B.: The traveling salesman problem: a guided tour of combinatorial optimization, vol. 3. Wiley New York (1985)
10. Lemouari, A., Guemri, O.: A two-phase scheduling method combined to the tabu search for the darp. International Journal of Applied Metaheuristic Computing (IJAMC) **5**(2), 1–21 (2014)
11. Lenstra, J.K., Kan, A.: Complexity of vehicle routing and scheduling problems. Networks **11**(2), 221–227 (1981)
12. Musatova, E., Lazarev, A.: Algorithm for solving two-stations railway scheduling problem. In: 25th European Conference on Operational Research (2012)
13. de Paepe, W.E., Lenstra, J.K., Sgall, J., Sitters, R.A., Stougie, L.: Computer-aided complexity classification of dial-a-ride problems. INFORMS Journal on Computing **16**(2), 120–132 (2004)
14. Parragh, S.N., Doerner, K.F., Hartl, R.F.: Demand responsive transportation. Wiley Encyclopedia of Operations Research and Management Science (2010)

MISTA 2015

Modeling the single-processor scheduling problem with time restrictions as a parallel machine scheduling problem

Rachid Benmansour · Oliver Braun · Hamid Allaoui

Abstract This paper addresses the single processor scheduling problem with time restrictions. The problem is new in the scheduling literature and was described for the first time by Braun et al. in [3]. A set of n jobs are simultaneously available for processing, non-preemptively, on a single processor which can handle only one job at a time. Furthermore, the number of jobs being executed during any time period of length $\alpha > 0$ is less than or equal to a given integer value $B \geq 2$. The objective function is to minimize the makespan C_{max}.
We present a mixed integer programming (MIP) formulation to solve this problem optimally. In addition, we show that this problem can also be considered as a $B+1$ parallel machine scheduling problem with one dedicated machine among them. The performance of the models are tested by running them on randomly generated instances.

1 Introduction

Scheduling problems are among the most frequently encountered problems in industry and have many applications in other fields. The single machine scheduling problem remains by far the most studied problem in the literature. This is due to its relative simplicity and to the fact that obtained results for such a problem can provide information on the solution of more complicated machine environments such as flow-shop, job-shop or parallel machine scheduling problems. Formally, the problem can be described as follows. A set $N_1 = \{1, 2, \ldots, n\}$ of n independent jobs are simultaneously available for processing at the beginning of the horizon time. Each job has to be processed non preemptively on a single processor that can handle only one job at a time. The objective function considered here is

Rachid Benmansour
University of Valenciennes and Hainaut Cambrésis, France
LAMIH – UMR CNRS 8201
E-mail: rachid.benmansour@univ-valenciennes.fr

Oliver Braun
Trier University of Applied Sciences, Environmental Campus Birkenfeld, 55761 Birkenfeld, Germany
E-mail: o.braun@umwelt-campus.de

Hamid Allaoui
University of Artois, France
E-mail: hamid.allaoui@univ-artois.fr

to minimize the makespan C_{max}. Furthermore, for a fixed integer B, no time interval $[x, x+\alpha)$ is allowed to intersect more than B jobs for any real $x > 0$. These constraints are known as the *time restriction constraints* and they arise in many industrial applications. For instance, when each job being processed requires the use of one of B identical subprocessors, that can be other external resources such as power, personnel, another machine, container, tank, etc. Further, each subprocessor that has been used needs a certain amount of time α to reset itself before another job can use it again (maintenance, cleaning, etc). Hence, it is never possible to process on more than B jobs during an interval $[x, x+\alpha)$. In [3], Braun et al. proved that this problem is NP-hard when the value B is variable and provided a detailed worst-case analysis. They showed, for $B \geq 3$, that any feasible solution can be processed within a factor of $2 - \frac{1}{B-1}$ of the optimum, plus an additional small constant. For $B = 2$, this factor is equal to $\frac{4}{3}$, plus an additional small constant. Recently in [1], Benmansour et al. proposed a MIP formulation to solve this problem optimally. They showed, via numerical experiments, that this MIP formulation, which based on the time indexed variables, is not suited to solve real-world instances. In this paper, we present two MIP formulations based on assignment and positional date variables that are more effective. The first one gives the optimal objective value with the corresponding sequence on the single processor, whereas the second one, modeled as a parallel machine scheduling problem, solves the problem optimally and gives the sequence on each subprocessors in the case where external resources are used.

2 MIP formulations

There exist several MIP formulations for scheduling problems, and are classified based on the choice of the decision variables: *i*) completion time variables, *ii*) time index variables, *iii*) linear ordering variables and *iv*) assignment and positional date variables. For more details, we refer the reader to papers [6] and [4]. In this section we propose two MIP formulations to solve the single processor scheduling with time restrictions problem.

2.1 Assignment and positional date variable model - MIP1

Let $N_1 = \{1, 2, \ldots, n\}$ be the set of jobs to be scheduled on the processor and p_i ($i \in N_1$) the processing time of the job i. Without loss of generality, these processing times are supposed to be integers. The amount of time needed for each subprocessor to reset itself is equal to α and is chosen as $\alpha = \max_{i \in N_1}\{p_i\}$. In addition, we suppose that the number of jobs being executed during any time period of length α is less than or equal to a given integer value $B \geq 2$. The objective is to minimize the makespan.
In order to model this problem, we define a binary assignment variable, x_{ik}, which equals 1 if job i is assigned to position k and equals 0 otherwise. We define also the variable $C_{[k]}$ as the completion time of the job in position k. Similarly, the variables $S_{[k]}$ and $p_{[k]}$ represent, respectively, the starting time of the job in position k and its corresponding processing time. The relationship between these variables is $C_{[k]} = S_{[k]} + p_{[k]}$. Moreover, since in the optimal schedule there may be idle times on the processor, we define the variable $I_{[k]}$ to represent the possible idle time between the instants $C_{[k]}$ and $S_{[k+1]}$. The first MIP formulation (MIP1) with assignment and positional date variables is as follows:

$$min\ S_{[n]} + p_{[n]}$$

$$s.t. \sum_{k=1}^{n} x_{ik} = 1 \quad \forall i \in N_1, \tag{1}$$

$$\sum_{i=1}^{n} x_{ik} = 1 \quad \forall k \in N_1, \tag{2}$$

$$p_{[k]} = \sum_{i=1}^{n} p_i x_{ik} \quad \forall k \in N_1, \tag{3}$$

$$S_{[k+1]} = S_{[k]} + p_{[k]} + I_{[k]} \quad \forall k \in N_1 \backslash \{n\}, \tag{4}$$

$$S_{[k+B]} \geq S_{[k]} + p_{[k]} + \alpha \quad \forall k \in \{1,2,\ldots,n-B\}, \tag{5}$$

$$I_{[k]} \geq 0 \quad \forall k \in N_1 \backslash \{n\}, \tag{6}$$

$$x_{ik} \in \{0,1\} \quad \forall (i,k) \in N_1^2, \tag{7}$$

$$S_{[1]} = 0, \tag{8}$$

$$S_{[k]} \geq 0 \quad \forall k \in N_1 \backslash \{1\}. \tag{9}$$

The constraints (1) guarantee that each job i occupies only one position. The set of constraints (2) states that each position k contains exactly one job. The processing time of the job scheduled at the k^{th} position and its starting time are determined, respectively, by constraints set (3) and constraints set (4). The time restrictions constraint is represented by constraints (5). The first B jobs are scheduled without any restriction since during the interval $[0, C_{[B]}($ no time interval of duration α is able to intersect more than B jobs. For the following jobs, the difference $S_{[k+B]} - C_{[k]}$ must be greater or equal to α as otherwise, the interval $\left[C_{[k]} - \varepsilon, S_{[k+B]}\right($ will intersect $B+1$ jobs for ε sufficiently small positive real number. Finally, the idle time between two jobs, if it exists, is determined by constraints (4) and (6). The last constraints (7) define x_{ik} as binary variables. Constraint (8) states that the first scheduled job starts its processing at time 0. This is obvious as otherwise the optimal schedule can always be shifted to the left.
This MIP formulation gives the optimal sequence of jobs on the processor but, does not provide any information on how to use the B external resources (if there are any). The purpose of the following MIP model is to address this situation.

2.2 Parallel machine scheduling problem - MIP2

In this model, the positive integer number B will be seen as the maximum number of subprocessors that can be used anytime respecting the following constraint. Each subprocessor needs α units of time to recover before another job can use it again. The sub-processors and the main processor form a set of $B+1$ parallel machines. Keeping the same notations as in (2.1), we can say that first machine (M_1) is dedicated to process jobs in N_1, and the remaining machines will process a set of dummy jobs $N_2 = \{n+1, n+2, \ldots, 2n\}$. Each time a job $i \in N_1$ is executed on M_1, the job $i+n \in N_2$ is executed, at the same time, on a certain machine ($M_k, k \neq 1$) and lasts for $p_{i+n} = p_i + \alpha$ units of time. That way, even if the job i is finished, the external resource remains unavailable for α units of time. The two sets form a new set of jobs $N = N_1 \cup N_2$.
We define two additional sets for the machines $M = \{1,2,\ldots,B+1\}$ and $M^* = M \backslash \{1\}$ and a set for the positions of jobs $P = \{1,2,\ldots,n\}$.
The model presented here is based on assignment and positional date variables. It is based on the MIP formulation proposed by Lasserre and Queyranne [5] and extended later by

Blazewicz et al. [2]. For ease of reading, we use the index p to design a position, m to design a machine, and j to design a job. Let the binary variable $x_{jm}^{p} = 1$ if the job j is assigned to position p on machine m; and 0 otherwise. Let the variable F_m^p denotes the completion time of the job at position p on machine m. Finally, the completion time of job j (regardless of its position) is defined by the variable C_j. The second MIP formulation is as follows.

$$min\ C_{max}$$

$$s.t.\ \sum_{p\in P}\sum_{m\in M^*} x_{jm}^{p} = 1 \quad \forall j \in N_2 \tag{10}$$

$$\sum_{p\in P} x_{j,1}^{p} = 1 \quad \forall j \in N_1 \tag{11}$$

$$\sum_{j\in N_1} x_{j,1}^{p} = 1 \quad \forall p \in P \tag{12}$$

$$\sum_{j\in N_2} x_{jm}^{p} \leq 1 \quad \forall p \in P \forall m \in M^* \tag{13}$$

$$F_m^1 \geq \sum_{j\in N_2} p_j x_{jm}^{1} \quad \forall m \in M^* \tag{14}$$

$$F_m^p \geq F_m^{p-1} + \sum_{j\in N_2} p_j x_{jm}^{p} \quad \forall m \in M^*, \forall p \in P\backslash\{1\} \tag{15}$$

$$F_1^1 = \sum_{j\in N_1} p_j x_{j,1}^{1} \tag{16}$$

$$F_1^p = F_1^{p-1} + \sum_{j\in N_1} p_j x_{j1}^{p} \quad \forall p \in \{2,3,...,B\} \tag{17}$$

$$F_1^p \geq F_1^{p-1} + \sum_{j\in N_1} p_j x_{j1}^{p} \quad \forall p \in \{B+1,...,n\} \tag{18}$$

$$F_1^p \geq F_1^{p-B} + \alpha + \sum_{j\in N_1} p_j x_{j1}^{p} \quad \forall p \in \{B+1,...,n\} \tag{19}$$

$$C_j \geq F_m^p - H(1-x_{jm}^{p}) \quad \forall j \in N_2, \forall p \in P \forall m \in M^* \tag{20}$$

$$C_j \geq z_{jp} \quad \forall j \in N_1, \forall p \in P \tag{21}$$

$$z_{jp} \leq H x_{j1}^{p} \quad \forall j \in N_1, \forall p \in P \tag{22}$$

$$z_{jp} \leq F_1^p \quad \forall j \in N_1, \forall p \in P \tag{23}$$

$$z_{jp} \geq F_1^p - H(1-x_{j1}^{p}) \quad \forall j \in N_1, \forall p \in P \tag{24}$$

$$C_j \leq F_1^p + H(1-x_{j1}^{p}) \quad \forall j \in N_1, \forall p \in P \tag{25}$$

$$C_{j+n} = C_j + \alpha \quad \forall j \in N_1 \tag{26}$$

$$C_{max} \geq C_j \quad \forall j \in N_1 \tag{27}$$

$$x_{jm}^{p} \in \{0,1\} \quad \forall (j,p) \in N_1^2, \forall m \in M \tag{28}$$

$$z_{jp} \geq 0 \quad \forall j \in N_1, \forall p \in P \tag{29}$$

Constraints (10) ensure that all jobs in N_2 are assigned to exactly one position on only one machine. Constraint set (11)-(12) state that each job in N_1 is assigned to exactly one position and, each position on machine M_1 contains exactly one job. This is not the case for the other machine since each position, on every machine, can contain at most one job. Constraints (13) guarantee these restrictions. In order to determine the completion time of the job at position

p on each machine in M^*, we introduce constraints (14) and (15). Similarly, constraints (16)-(19) determine the completion time of the job at position p on M_1. This set of constraints includes the *time restriction constraints*. The completion time of job $j \in N_2$ (regardless of its position) is given by constraint set (20). The set of constraints (21)-(24) is a linearization of the following constraints:

$$C_j \geq F_1^p x_{j1}^p \forall j \in N_1, \forall p \in P$$

where $z_{jp} = F_1^p x_{j1}^p$ and H is a large positive integer. The value of H can be chosen as $H = \sum_{i \in N_1} p_i + \lfloor n/B \rfloor \alpha$.
The meaning of these constraints is straightforward. If job $j \in N_1$ is scheduled on position p on M_1 then $C_j \geq F_1^p$. The constraint (25) becomes then $C_j \leq F_1^p$ which leads to $C_j = F_1^p$. Otherwise, i.e. $x_{j1}^p = 0$, the set of constraint (21)-(25) imposes that $C_j \geq 0$ which is no restrictive. The last two constraints are simple. Constraints (26) state that the starting time of job i on machine M_1 is exactly the same as the starting time of job $i+n$ on one of the parallel machines $M_k, k \neq 1$. Constraints (27) calculate the value of the makespan. Finally restrictions (28), respectively (29), define the variables x_{jm}^p as binaries and the variables z_{jp} as continuous and positive.

2.2.1 Illustrative example

Given an instance with $B = 2$ (number of subprocessors), $n = 9$ jobs, $\alpha = 10$ (time for a subprocessor to reset itself), and the processing times of the jobs as given in Table 2.2.1. The optimal objective function value can be obtained by applying the MIP2 formulation as given above. It takes 2.04 seconds to solve the problem using on ILOG CPLEX 12.6 on

Job	1	2	3	4	5	6	7	8	9
p_i	6	5	4	10	9	8	2	8	7

Table 1 Example instance for $n = 9$.

a personal computer with 2.8 GHz CPU and 16GB RAM memory. The optimal sequence for the $n = 9$ problem instance is 7-1-2-5-4-8-9-6-3 as shown in Fig. 1, with a value of the objective function (makespan) $C_{max} = 68$. In this solution, the first job to be scheduled on the processor (i.e. M_1) is job 7. At the same time job 16 ($7+n$) starts its execution on M_2. The completion time of this job is $C_{16} = 12$, which means that even if job 7 is completed at $C_7 = 2$, the machine M_2 remains unavailable for an extra α units of time. The second job in the sequence is job 1. This job cannot use the external resource M_2 just after job 7. Instead it can use M_3 which is still not used yet. etc. The same example was solved with the MIP1 in 0.13 seconds. The optimal sequence is the following 3-8-6-4-5-1-2-9-7 as shown in Fig. 2.

3 Conclusion and Further work

We have proposed two MIP formulations to solve the single-processor scheduling problem with time restrictions. Preliminary results show that the first MIP formulation is way better than the second one which requires a large number of variables. However this model allows the scheduler to obtain detailed information about how to use the external resources (if

Fig. 1 Scheduling on a single processor with two external and identical resources (a)

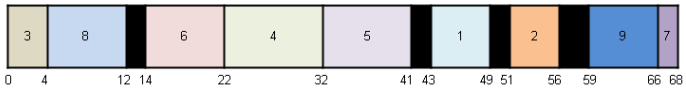


Fig. 2 Scheduling on a single processor with two external and identical resources (b)

any). At the conference we will provide more details and some experimental results on our formulations. We will present also an algorithm to determine which external resource to use in case MIP1 formulation is used. It's more likely that the studied problem is NP-hard even if B is fixed. Our future work is to prove the NP-hardness of the problem and then think about some heuristics and metaheuristics to provide good solutions for the problem.

References

1. Benmansour R, Braun O, Artiba A (2014) On the single-processor scheduling problem with time restrictions. In: Control, Decision and Information Technologies (CoDIT), 2014 International Conference on, IEEE, pp 242–245, DOI 10.1109/CoDIT.2014.6996900
2. Blazewicz J, Dror M, Weglarz J (1991) Mathematical programming formulations for machine scheduling: a survey. European Journal of Operational Research 51(3):283–300
3. Braun O, Chung F, Graham R (2014) Single-processor scheduling with time restrictions. Journal of Scheduling 17(4):399–403
4. Keha A, Khowala K, Fowler J (2009) Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering 56(1):357–367
5. Lasserre JB, Queyranne M (1992) Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. In: IPCO, pp 136–149
6. Unlu Y, Mason SJ (2010) Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. Computers & Industrial Engineering 58(4):785–800

MISTA 2015

An IP-based Model for the Post-Enrollment-based Course Timetabling Problem at TU Berlin

János Höner · Gerald Lach · Erhard Zorn

Abstract In this paper, we present a new IP-based model for the post-enrollment-based course timetabling problem at TU Berlin; this problem has to be solved in order to assign students to tutorials. We show that the new model yields better results than an old one, which uses two separate assignments of students to tutorials and tutorials to timeslots and rooms. The new model is applied to real-world data of 24,700 assignments for 8,600 students in 1,000 tutorials of 80 courses in 170 rooms within 500 seconds on an i7 quad-core machine.

1 Introduction

The *post-enrollment-based course timetabling* problem (PECT problem) is a timetabling problem with which many large universities have to struggle. The goal is to assign students to courses and courses to rooms with the full information regarding which student attends which course. The resulting timetable has to satisfy a number of hard constraints in order to be *feasible*. Usually, there are additional soft constraints or other measures to define the quality of a feasible timetable. A detailed definition of this problem is given for the International Timetabling Competition (ITC) in [9]. In general, this problem is known to be hard to solve [8].

For courses with large numbers of students, there are often additional exercise sessions called *tutorials*. Tutorials are usually held by graduate students—tutors—in small classes. Consequently, there may be a large number of tutorials for one course. The planning of these tutorials is a typical example of the PECT problem. At TU Berlin, the planning of tutorials is managed by means of *MosesKonto*. This software

János Höner
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: hoener@math.tu-berlin.de

Gerald Lach
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: lach@math.tu-berlin.de

Erhard Zorn
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: erhard@math.tu-berlin.de

was developed by innoCampus, a department that develops software to support e-learning and course management at universities.

1.1 Details of tutorial planning at TU Berlin

At TU Berlin, in the first week of the semester, students subscribe for their tutorials and prioritize timeslots in which they would prefer to schedule these tutorials. This data is used to compute a timetable that satisfies as many of the students' priorities as possible. At present, a model based on an integer program is used to optimize the assignment of students to tutorials. The preceding assignment of tutorials to rooms and timeslots is done by hand but with some heuristic software assistance: The user sees how many students of a course prefer which timeslots for their tutorials. Based on this, he can adapt the number of tutorials at specific timeslots, in accordance with the students' wishes—provided there are enough free rooms and tutors for that timeslot.

Tutors of a course who are graduate students also taking other courses inform the person in charge of their course when they will presumably be able to give their tutorials. This information is used to define the maximal number of (simultaneous) tutorials of a course at a given timeslot. As most tutors—graduate students—do not have to attend tutorials themselves, there is a small risk of conflicts.

MosesKonto has been used since 2004 to manage the planning of tutorials at TU Berlin. At first, only the tutorials of five mathematics courses for engineers were planned using MosesKonto; at present the software is used for approximately 1,100 tutorials of more than 90 large courses—the majority of all tutorials at TU Berlin.

Due to the massive increase in the number of courses and tutorials, the preceding assignment of tutorials to rooms and timeslots is strenuous. Additionally, only two-hour timeslots for all courses can be planned using the original model. At TU Berlin, we generally use an 'even' 2-hours time pattern (i.e. courses can take place 8:00–10:00 a.m. and so on). Most of the courses take place within a two-hour slot between 08:00 a.m. and 06:00 p.m. But several courses also use 4-hour timeslots (e.g. laboratories). Furthermore, the resulting timetable may deviate considerably from an optimal one due to the two separate assignments of, first, tutorials to rooms and timeslots and, second, students to tutorials. Therefore, we integrated the process of assigning tutorials to rooms and timeslots into the optimization algorithm in order to attain better timetables and to develop a more automated planning procedure. To achieve this, we adapted the old model to obtain a more advanced one that serves our needs.

1.2 Related work

The problem of constructing timetables is a well-researched field, one that has produced a wide variety of algorithmic solutions. Due to the complexity of these problems, many of them are of a heuristic nature, such as *local search* [11, 2] and *ant algorithms* [10, 1]. Besides these heuristics, the approach to modeling timetabling problems as an *integer program* has gained in popularity only recently. This non-heuristic approach might be a little computation-heavy, but provides an optimal

solution or, at least, some global statement about the distance to optimality. It has already been used in several cases and yields good results [7, 4, 12].

2 Old model at TU Berlin

The goal of the old model is to find an assignment of students to tutorials, as is already planned in the preceding procedure (see Subsection 1.1). In this context, a tutorial is fully defined by a tuple of the kind *(course, timeslot, room)*. If we now assign students to tutorials, we would get tuples of the form *(student, course, timeslot, room)*. To simplify our model, we omit the room in this assignment. If the room is not unique (i.e. there is more than one tutorial of the same course at the same timeslot in different rooms), then the distribution of the students into these rooms can be chosen arbitrarily. This results in the task of finding tuples of the form *(student, course, timeslot)*

The old model, which has been used so far to solve the PECT problem at TU Berlin, is basically an integer program with hard constraints modeled as constraints and the soft constraints modeled as the objective function. It tries to formulate this integer problem. Thus, we need the following variables:

- A set T of q pairwise disjoint two-hour timeslots that, together, cover the whole time span of a timetable
- A set C of n courses
- A set S of m students that attend different courses
- The number of students $\text{cap}(c, t)$ that a course c can admit at a timeslot t. This is the sum of the numbers of students that all tutorials of course c at timeslot t can admit (there may be simultaneous tutorials of a course; these simultaneous tutorials can have different capacities depending on the room and other factors)
- A subset $C_s \subset C$ of all the courses student s has to take

A *feasible timetable* is an assignment of students to courses and timeslots in such a way that:

- no student has to take more than one course at a time;
- there should only be as many students assigned to a course as the course can accomodate at each time; and
- each student has to be assigned to all the courses he or she wants to take.

Such a feasible timetable is called *optimal* if the students' priorities are taken into account as well as possible. The priority of student s for course c at time t is given by $\text{prio}(s, c, t)$ (where a lower value denotes a higher priority).

An assignment of students will be represented as a vector $X \in \{0, 1\}^{|\hat{\mathcal{X}}|}$, where

$$X_{(s,c,t)} = \begin{cases} 1 & \text{if student } s \text{ is assigned to course } c \text{ at time } t \\ 0 & \text{else} \end{cases}$$

and

$$\hat{\mathcal{X}} := \{(s, c, t) : s \in S, c \in C_s, t \in T\}$$

2.1 Formulation as an integer program

Using all the details given above, the PECT problem is formulated as an integer program as follows:

$$\min \sum_{(s,c,t)\in\hat{\mathcal{X}}} X_{(s,c,t)} \cdot \text{prio}(s,c,t)$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{t\in T} X_{(s,c,t)} = 1 && \forall s \in S, c \in C_s \\
& \sum_{c\in C_s} X_{(s,c,t)} \leq 1 && \forall s \in S, t \in T \\
& \sum_{s\in S} X_{(s,c,t)} \leq \text{cap}(c,t) && \forall c \in C, t \in T \\
& X \in \{0,1\}^{|\hat{\mathcal{X}}|}
\end{aligned} \tag{IP-OLD}$$

The first constraint ensures that every student attends exactly one tutorial of each course that he or she wants to take. The second ensures that no student has to attend more than one tutorial at a time. Finally, the third constraint sees to it that the number of students attending one course at the same time does not exceed the course's capacity at that time.

2.2 Graphical interpretation

The above given integer program (IP-OLD) can be interpreted in a graph theoretical way as a *min-cost-flow* problem (for details, see [13]) with additional constraints. This is shown in Figure 1; the graph is composed of a source $\mathcal{S}$, a target $\mathcal{T}$, and three different sets of nodes:

- S: the set of all students
- $T^S := \bigcup_i T^{s_i}$, where $T^{s_i} := \bigcup_{t\in T} t^{s_i}$: a copy of all timeslots for each student s_i (s-timeslots)
- $T^C := \bigcup_j T^{c_j}$, where $T^{c_j} := \bigcup_{t\in T} t^{c_j}$: a copy of all timeslots for each course c_j (c-timeslots)

Each student node s is connected to the source $\mathcal{S}$ and to all of its corresponding s-timeslots in T^s. All the s-timeslots in T^s are connected to the corresponding c-timeslots in T^c, if student s takes course c; in addition, there are edges from each c-timeslot to the target $\mathcal{T}$. The capacities of the edges from the source to the students are given by the number of courses that each student takes. On the edges from c-timeslots to the target, capacities are defined by the number of students that a course can admit. All remaining edges get a capacity of one. Costs only exist for the edges between s-timeslots and c-timeslots; these are given by the priority the student assigned to that combination of course and timeslot.

From that, we get the following four sets of edges and their capacities:

1. $(\mathcal{S}, s) : \forall s \in S$ with $\text{c}\left((\mathcal{S}, s)\right) = |C_s|$

2. $(s, t^s) : \forall s \in S, t^s \in T^s$ with $\mathrm{c}\left((s, t^s)\right) = 1$
3. $(t^s, t^c) : \forall t^s \in T^s, t^c \in T^c$, if student s takes course c with $\mathrm{c}\left((t^s, t^c)\right) = 1$
4. $(t^c, \mathcal{T}) : \forall t^c \in T^C$ with $c\left((t^c, \mathcal{T})\right) = \mathrm{cap}(c, \mathcal{T})$

After defining the following subset of edges

$$\Sigma := \bigcup_{s \in S, c \in C} \sigma_{s,c} \qquad \text{with} \qquad \sigma_{s,c} := \bigcup_{t \in T} (t^s, t^c)$$

and setting the demand and supply to the total number of student tutorial assignments, we can formulate the PECT problem as a *min-cost-flow* problem with a further constraint (1) to ensure that each student attends only one course at a time. This flow problem can again be formulated as an integer program [6] for the set of vertices V and the set of edges E:

$$\begin{aligned}
\min \quad & \sum_{e \in W} \mathrm{cost}(e) x_e \\
\text{s.t.} \quad & \sum_{e \in \delta^+(\mathcal{S})} x_e = \sum_{e \in \delta^-(\mathcal{T})} x_e = \sum_{s \in S} |C_s| \\
& \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 && \forall v \in V \setminus \{\mathcal{S}, \mathcal{T}\} \\
& \sum_{e \in \sigma} x_e \leq 1 && \forall \sigma \in \Sigma \qquad (1) \\
& 0 \leq x_e \leq c(e) && \forall e \in E \\
& x \in \mathbb{Z}^{|E|}
\end{aligned}$$

W is the the set of priority edges (item 3 in the above list) and σ is a subset of edges from T^s to T^c for all $t \in T$ and $s \in S$. $\delta^+(v)$ and $\delta^-(v)$ denotes the set of outgoing and ingoing edges of node v, respectively.

3 New model at TU Berlin

As stated in the Introduction, the old model is simply an assignment of the students to the courses, according to their priorities. The assignment of courses to rooms has to be carried out in a preceding separate step. Although the process is supported by some heuristics, it still includes a lot of work done by hand; moreover there is no way to guarantee optimality. The new model we developed includes the assignment of courses to rooms in the optimization process; therefore, it not only provides us with much better results but also leads to a more advanced and highly automated process to construct the timetables. Besides this structural improvement of the algorithmic part, we also adapted the underlying time model to become more flexible. In the end, our model satisfies the following two new requirements:

1. One tutorial can occupy two or more (not necessarily consecutive) timeslots.
2. The assignment of courses to rooms and times should be done by the algorithm considering the priorities of the students.

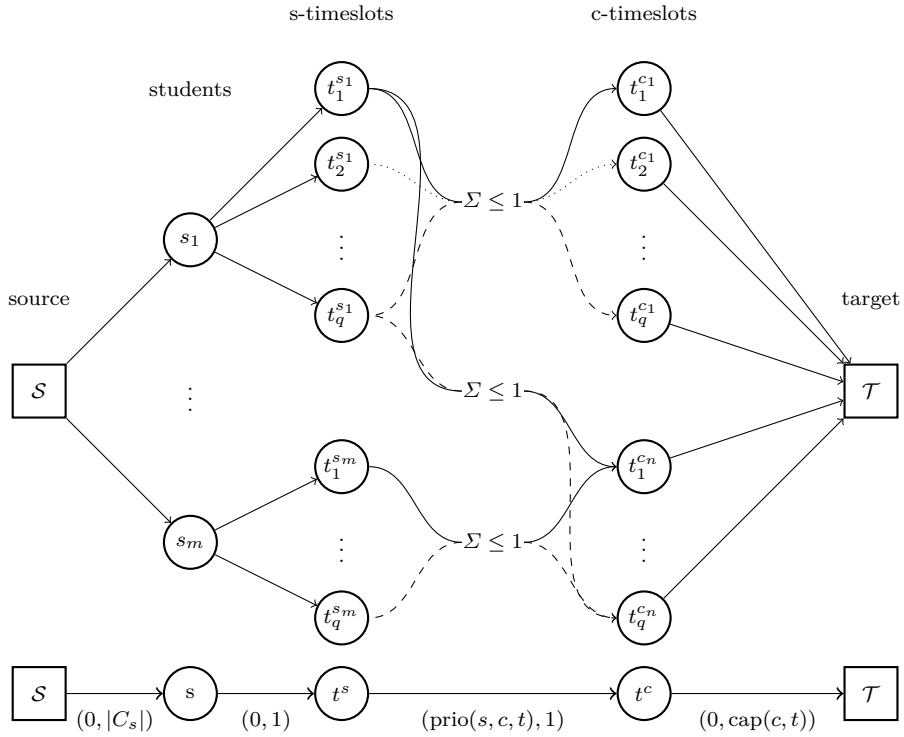


Fig. 1 Graph of the *min-cost-flow* formulation of the PECT problem stated in Section 2.2. Sample costs and capacities are given below the actual graph where the first value is the cost and the second the capacity of the edge. Image adapted from [5].

3.1 Time model

In order to allow tutorials to occupy more than one timeslot, we introduce a new entity called *period*. A period is a subset of the set of all timeslots. In a period we can combine as many timeslots in one entity as we need. The set P of all periods is a subset of the power set of the set of all timeslots: $P \subset \mathcal{P}(T)$. Typically, P contains all timeslots and some subsets of T with more than one element. Furthermore, for a timeslot t, let $P_t \subset P$ be the set of all periods that contain timeslot t:

$$P_t := \{p \in P : t \in p\}$$

With this new time model, we can reformulate the constraints of (IP-OLD) as follows: We define

$$\hat{\mathcal{X}} := \{(s, c, p) : s \in S, c \in C_s, p \in P\}$$

A vector $X \in \{0,1\}^{|\hat{\mathcal{X}}|}$ is said to be feasible if it satisfies the following constraints:

$$\sum_{p \in P} X_{(s,c,p)} = 1 \qquad \forall s \in S, c \in C_s$$

$$\sum_{c \in C_s, p \in P_t} X_{(s,c,p)} \leq 1 \qquad \forall s \in S, t \in T$$

$$\sum_{s \in S} X_{(s,c,p)} \leq \text{cap}(c,p) \qquad \forall c \in C, p \in P$$

Note that not only are the three constraints changed but also the space over which we optimize.

3.2 Variable room assignment

In order to let the algorithm handle the assignment of courses into rooms, we introduce a new set of variables:

$$Y_{c,p,r} = \begin{cases} 1 & \text{if course } c \text{ takes place in room } r \text{ at time } p \\ 0 & \text{else} \end{cases}$$

With these new variables Y, we have to change some of the old constraints and add a couple of new ones to our model.

First, the number of students that a course can take at a certain time ($\text{cap}(c,p)$) is no longer a fixed parameter, but depends on the size or capacity of the rooms where the course takes place at that time. Second, the following six constraints must be satisfied by the room assignment Y:

1. Each course has to offer exactly the required amount of tutorials.
2. Tutorials need to take place in suitable rooms.
3. At each time, there cannot be more than one tutorial per room.
4. Existent timetables (which room is available at what time) have to be taken into account.
5. The capacity of a room must not be exceeded.
6. The maximum number of tutorials that a course can offer at a time and the maximum number of students that are permitted in a tutorial of a course must be taken into account.

To be able to model all these new requirements, we introduce the following list of parameters:

S	Set of m students
C	Set of n courses
T	Set of q timeslots
P	Set of $\tilde{q}$ periods needed
R	Set of ℓ rooms
C_s	Set of all courses that student s wants to take
R_c	Set of all rooms that are suitable for course c
P_r	Set of all periods in which room r is available
P_c	Set of all periods in which course c can at least offer one tutorial

P_s	Set of all periods in which student s can take courses
$\text{prio}(s,p)$	Priority of student s for period p
n_c	Total number of tutorials offered by course c
$\text{cap}(r)$	Capacity (number of seats) of room r
$m(c,p)$	Maximum number of tutorials that can be offered by course c in period p
l_c	Maximum number of students that are permitted in one tutorial of course c

3.3 Integer program formulation

With these parameters, and the variables X and Y, it is possible to formulate the extended problem of finding an optimal timetable as an integer program. To keep the space over which we optimize as small as possible, we exclude some infeasible assignments a priori. For example, $Y_{c,p,r}$ is infeasible if room r is unavailable at time p or if it is unsuitable for courses c. Let $\mathcal{X}$ and $\mathcal{Y}$ be defined by $\mathcal{X} := S \times C \times P$ and $\mathcal{Y} := C \times P \times R$. Then, we define the sets of all *possible* assignments as:

$$\hat{\mathcal{X}} := \{(s,c,p) \in \mathcal{X} : c \in C_s, p \in P_c \cap P_s\}$$
$$\hat{\mathcal{Y}} := \{(c,p,r) \in \mathcal{Y} : r \in R_c, p \in P_r \cap P_c\}$$

The objective function is the same as in (IP-OLD) and the constraints can be modeled in a straightforward manner; they result in the following integer program:

$$\min \quad \sum_{(s,c,p)\in\hat{\mathcal{X}}} X_{(s,c,p)} \cdot \text{prio}(s,p)$$

s.t.

$$\sum_{(s,c,p)\in\hat{\mathcal{X}}} X_{(s,c,p)} = 1 \qquad \forall s \in S, c \in C_s \quad (2)$$

$$\sum_{p\in P_t,(s,c,p)\in\hat{\mathcal{X}}} X_{(s,c,p)} \leq 1 \qquad \forall s \in S, t \in T \quad (3)$$

$$\sum_{(c,p,r)\in\hat{\mathcal{Y}}} Y_{(c,p,r)} = n_c \qquad \forall c \in C \quad (4)$$

$$\sum_{p\in P_t,(c,p,r)\in\hat{\mathcal{Y}}} Y_{(c,p,r)} \leq 1 \qquad \forall r \in R, t \in T \quad (5)$$

$$\sum_{(c,p,r)\in\hat{\mathcal{Y}}} Y_{(c,p,r)} \leq m(c,p) \qquad \forall c \in C, p \in P \quad (6)$$

$$\sum_{(s,c,p)\in\hat{\mathcal{X}}} X_{(s,c,p)} - \sum_{(c,p,r)\in\hat{\mathcal{Y}}} Y_{(c,p,r)} \cdot \max(\text{cap}(r), l_c) \leq 0 \qquad \forall c \in C, p \in P_c \quad (7)$$

$$Y \in \{0,1\}^{|\hat{\mathcal{Y}}|}$$
$$X \in \{0,1\}^{|\hat{\mathcal{X}}|}$$

Constraint (2) and (3) are taken over from the old model and ensure that each student takes exactly one tutorial in each of the courses he or she has to take as well

as that no student has to attend more than one tutorial at a time. Constraint (7) replaces the last constraint of the old model (IP-OLD) and ensures that the number of attending students never exceeds the maximal tutorial size or the room capacity. The remaining constraints ensure that the correct number of tutorials is offered for each course (4) but does not exceed the given maxima for each time and course (6) and that there is, at most, one tutorial at a time in each room (5).

3.4 Graphical interpretation

As for the old model in Section 2, we found a strong connection to network-flow problems for the new one as well. We take the graph from Figure 1, substitute all timeslot nodes with period nodes, and add a third category of nodes: the r-periods. R-periods are located between the target and the c-periods all connected to the target and with an edge to a c-period if the corresponding room is suitable for the corresponding course. The costs of all new edges are zero and the capacity of edges from r-periods to the target are defined by the room's capacity $\text{cap}(r)$, while edges between c- and r-periods are given by the maximum number of students per tutorial l_c. The resulting graph is depicted in Figure 2. Again, the *min-cost-flow* model is not fully capable of modeling all the needed constraints. Therefore, we define some subsets of edges

$$\begin{aligned}
\Omega &:= \bigcup_{c \in C} \omega_c && \text{with} && \omega_c := \bigcup_{p \in P, r \in R_c} (p^c, p^r) \\
\Theta &:= \bigcup_{c \in C, p \in P} \theta_{c,p} && \text{with} && \theta_{c,p} := \bigcup_{r \in R_c} (p^c, p^r) \\
\Sigma &:= \bigcup_{s \in S, c \in C} \sigma_{s,c} && \text{with} && \sigma_{s,c} := \bigcup_{p \in P} (p^s, p^c)
\end{aligned}$$

and then formulate the PECT problem with variable room assignment as a *min-cost-flow* problem with some additional constraints:

$$\begin{aligned}
\min \quad & \sum_{e \in W} \text{cost}(e) x_e \\
\text{s.t.} \quad & \sum_{e \in \delta^+(s)} x_e = \sum_{e \in \delta^-(t)} x_e = n \\
& \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 && \forall v \in V \setminus \{s, t\} \\
& \sum_{e \in \sigma} x_e \leq 1 && \forall \sigma \in \Sigma \\
& \sum_{e \in \omega} y_e \leq n_c && \forall \omega \in \Omega \\
& \sum_{e \in \theta} y_e \leq m(c, p) && \forall \theta \in \Theta
\end{aligned}$$

$$\sum_{e \in \delta^-(P_t^r)} y_e \leq 1 \qquad \forall r \in R, t \in T \qquad (8)$$

$$\sum_{e \in \delta^-(P_t^s)} x_e \leq 1 \qquad \forall s \in S, t \in T \qquad (9)$$

$$c(e)y_e - x_e \geq 0 \qquad \forall e \in E$$

$$0 \leq x_e \leq c(e) \qquad \forall e \in E$$

$$x \in \mathbb{Z}$$

$$y \in \{0,1\}$$

4 Computing a timetable

To compensate for inconsistencies in the data that would result in an infeasible IP, we introduce penalty variables for students who cannot be assigned to their courses ($a_{s,c}$) as well as for courses that cannot be assigned to rooms (b_c). Using these new variables, we get the final IP that we used for computing timetables with; this led to a significant improvement in the solution, as compared to the old model.

$$\min \quad \sum_{(s,c,p) \in \hat{\mathcal{X}}} X_{(s,c,p)} \cdot \text{prio}(s,p) + \sum_{c \in C} b_c \cdot \beta + \sum_{s \in S, c \in C_s} a_{s,c} \cdot \alpha$$

s.t.

$$\sum_{(s,c,p) \in \hat{\mathcal{X}}} X_{(s,c,p)} + a_{s,c} = 1 \qquad \forall s \in S, c \in C_s$$

$$\sum_{p \in P_t, (s,c,p) \in \hat{\mathcal{X}}} X_{(s,c,p)} \leq 1 \qquad \forall s \in S, t \in T$$

$$\sum_{(c,p,r) \in \hat{\mathcal{Y}}} Y_{(c,p,r)} + b_c = n_c \qquad \forall c \in C$$

$$\sum_{p \in P_t, (c,p,r) \in \hat{\mathcal{Y}}} Y_{(c,p,r)} \leq 1 \qquad \forall r \in R, t \in T$$

$$\sum_{(c,p,r) \in \hat{\mathcal{Y}}} Y_{(c,p,r)} \leq m(c,p) \qquad \forall c \in C, p \in P$$

$$\sum_{(s,c,p) \in \hat{\mathcal{X}}} X_{(s,c,p)} - \sum_{(c,p,r) \in \hat{\mathcal{Y}}} Y_{(c,p,r)} \cdot \max(\text{cap}(r), l_c) \leq 0 \qquad \forall c \in C, p \in P_c$$

$$Y \in \{0,1\}^{|\hat{\mathcal{Y}}|}$$

$$X \in \{0,1\}^{|\hat{\mathcal{X}}|}$$

(IP-FINAL)

In the graphical model, you can see the introduction of penalty variables in the following way. First, the supply of the source and the demand of the target are reduced iteratively until a feasible flow exists. Thereafter, the flow is minimized with respect to its costs.

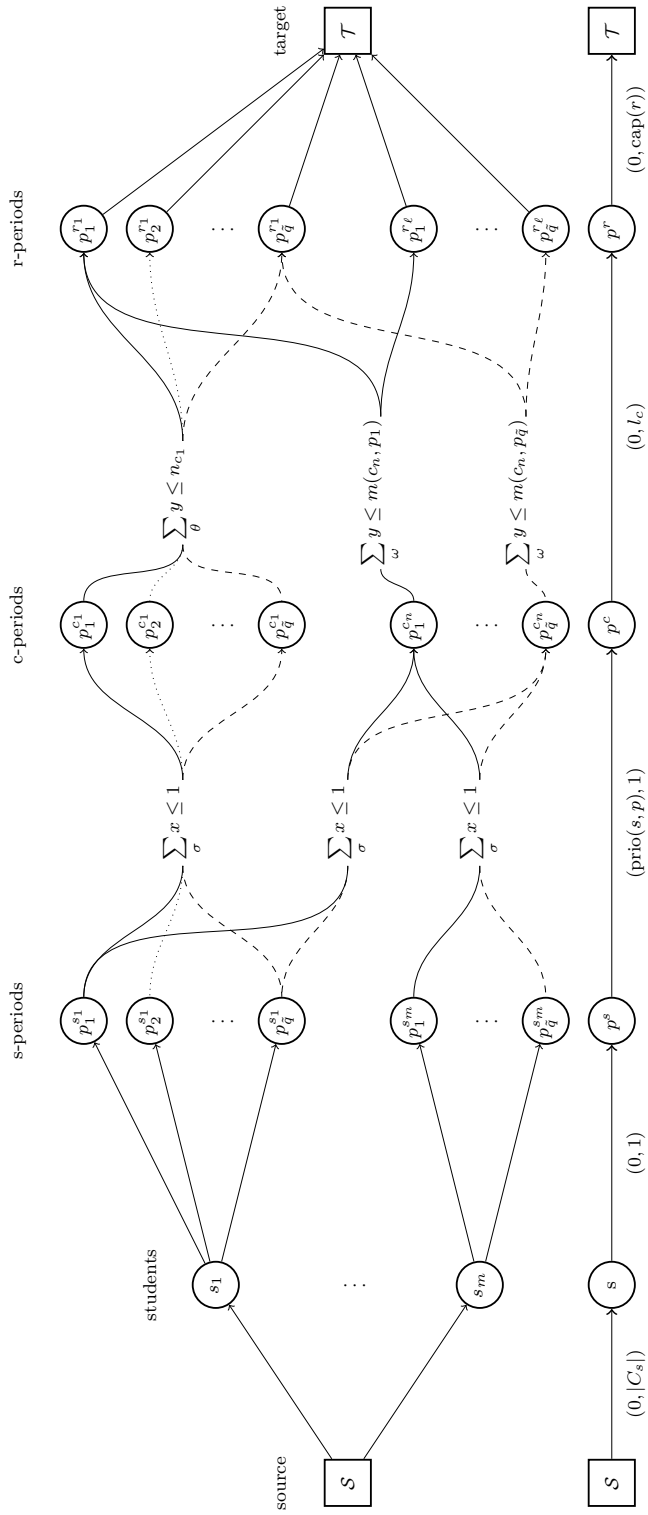


Fig. 2 Graph of the *min-cost-flow* model for the variable room assignment. The two constraints (8) and (9) are not modeled in the graph itself. Costs and capacities of edges are given beneath the actual graph.

4.1 Complexity

Similar to most of the timetabling problems, the one defined in (IP-FINAL) is NP-hard, and we will show this by reducing it to the problem of finding the maximum 3-dimensional matching in a given hypergraph.

Theorem 1 *The PECT problem defined by* (IP-FINAL) *is NP-hard.*

Proof We show the NP-hardness by a reduction to the *maximum 3-dimensional matching* problem

Definition 1 *Let A, B, and C be three disjoint finite sets and $T \subset (A \times B \times C)$ be the set of edges. A subset $M \subset T$ is called a* 3-dimensional matching *if, for any two distinct triples $(a_1, b_1, c_1), (a_2, b_2, c_3) \in M$, the following holds:*

$$a_1 \neq a_2 \wedge b_1 \neq b_2 \wedge c_1 \neq c_2 \tag{10}$$

Finding such a 3-dimensional matching of maximum size is a common NP-hard problem [3] and we reduce (IP-FINAL) as follows:

Let (A, B, C, T) be an instance based on which we try to find the *maximum 3-dimensional matching* M, i.e., it maximizes the cardinality of M. We then set A, B, and C to be the sets of students, courses, and rooms in (IP-FINAL), respectively. The subsets C_s and R_c are chosen in such away that the set $\{(s, c, r) \in S \times C \times R : c \in C_s, r \in R_c\}$ matches the set of edges in T. The set of all periods and the subsets for each room $P = P_r = \{p\}$ are all set to the same single element. The timeslots do not matter and can be chosen arbitrarily. By setting $\text{cap}(r) = m(c, p) = l_c = n_c = 1$, we enforce that each course offers one tutorial at time p and that each of these tutorials can take exactly one student.

If there exists a *feasible timetable* of this problem, it is an assignment of students into tutorials and rooms at a single time p in such a way that, at most, one student is attending one tutorial at a time, and, at most, one tutorial takes place in one room. In other words, the resulting timetable will consist of assignment triples (s, c, r) that satisfy the condition of a 3-dimensional matching (pairwise different in each component, see equation (10)). These triples will correspond to a 3-dimensional matching $M = T$ that covers all edges.

If no such *feasible timetable* exists, solving (IP-FINAL) will lead to a set of triples (s, c, r) that has maximum cardinality.

In both cases, we get a solution to the *maximum 3-dimensional matching* □

5 Results

The final model (IP-FINAL) was implemented in the modeling language OPL and solved using CPLEX on an i7 quad-core machine. The data sets were real data from the *MosesKonto*, collected during the period 2011–2013. The smaller set includes 64 different courses and about 16,200 student-tutorial assignments. The larger one contains 80 courses and appoximately 24,700 student-tutorial assignments. We set a time limit of 500 seconds for the CPLEX solver for both data sets. For the smaller set, we were able to compute an optimal timetable in the given time, whereas, for the larger set, the found solution differs less than 1% from the optimum.

Table 1 Comparison of the solutions of the new model and the old model.

	summer term 2011	winter term 2012/13
courses	64	80
tutorials	869	1,014
rooms	136	166
students	5,756	8,564
assignments	16,198	24,686
average priority (old model)	1.23	1.27
average priority (new model)	1.13	1.16
distance to optimum	0	$< 1\%$

To compare our new solution with the old one (i.e. the actual timetable that was used at TU Berlin), we examined the average incorporated priority of the student-tutorial assignments (see Table 1). For the smaller data set, this value decreased from 1.23 to 1.13, which corresponds to a single-priority improvement in about 1,500 cases (more than 9%). For the larger sample with a total of 24,686 assignments, the average decreased from 1.27 to 1.16, which corresponds to a single-priority improvement in about 2,900 cases (more than 12%).

6 Conclusion

We have presented a new model for the post-enrollment-based timetabling problem at TU Berlin. The new model involves the assignment of students to courses and of courses to timeslots and rooms as well. Thus, these two separate assignments of the old model are now included in one optimization process. The new model yields better results and enables an automated process to construct timetables.

7 Outlook

One way to gain even more flexibility and automation is by managing all tutors in some sort of a tutor pool. The assignment of a tutor to a tutorial will then be maintained automatically by the algorithm as well. Tutors can prioritize certain timeslots in the same manner as students, and the algorithm tries to take these into account. Of course, there will be some new constraints that need to be satisfied (e.g. the qualification of a tutor must match the required qualification of the tutorial, and no tutor can give more than one tutorial at the same time). Integrating this step into the optimization process would possibly lead to better results and the only work that still has be done manually (finding the parameter $m(c, p)$) would become superfluous.

References

1. Ayob, M., Jaradat, G.: Hybrid ant colony systems for course timetabling problems. In: Data Mining and Optimization, 2009. DMO'09. 2nd Conference on, pp. 120–126. IEEE (2009)
2. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Annals of Operations Research **194**(1), 111–135 (2012)
3. Crescenzi, P., Kann, V.: A compendium of NP optimization problems (1995)
4. Daskalaki, S., Birbas, T., Housos, E.: An integer programming formulation for a case study in university timetabling. European Journal of Operational Research **153**(1), 117–135 (2004)
5. Jeschke, S., Lach, G., Luce, R., Pfeiffer, O., Zorn, E.: Management and optimal distribution of large student numbers. In: Automation, Communication and Cybernetics in Science and Engineering 2009/2010, pp. 71–84. Springer (2011)
6. Jeschke, S., Luce, R., Pfeiffer, O., Zorn, E.: An optimized algorithm for distributing large numbers of students to small exercise groups. In: Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on, pp. 232–234. IEEE (2007)
7. Lach, G., Lübbecke, M.E.: Optimal university course timetables and the partial transversal polytope. In: Experimental Algorithms, pp. 235–248. Springer (2008)
8. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR spectrum **30**(1), 167–190 (2008)
9. Lewis, R., Paechter, B., McCollum, B., et al.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. http://www.cs.qub.ac.uk/itc2007/postenrolcourse/report/Post Enrolment based CourseTimetabling.pdf (2007). Last visited 28.04.2015
10. Nothegger, C., Mayer, A., Chwatal, A., Raidl, G.R.: Solving the post enrolment course timetabling problem by ant colony optimization. Annals of Operations Research **194**(1), 325–339 (2012)
11. Schaerf, A.: Local search techniques for large high school timetabling problems. systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **29**(4), 368–377 (1999)
12. Schimmelpfeng, K., Helber, S.: Application of a real-world university-course timetabling model solved by integer programming. OR Spectrum **29**(4), 783–803 (2007)
13. Schrijver, A.: Combinatorial Optimization Polyhedra and Efficiency. Springer, Berlin (2003)

MISTA 2015

Simulation Based Cause and Effect analysis of Input Variables in Wafer Fabrication

Rashmi Singh . M. Mathirajan

Abstract: Controlling a re-entrant flow line typically found in the semiconductor wafer fabrication industry is complex. To obtain a fundamental understanding of the system and to evaluate the effects of different input variables on selected parameters such as cycle time, WIP (work-in-process) level and throughput, we construct a simulation model of Intel Mini-Fab using Arena simulation software. The Intel Mini Fab has been selected for this study as it captures the challenges involved in scheduling the highly re-entrant semiconductor wafer fabrication flow lines. The input variables include arrival rate, arrival distribution, processing time, maintenance schedule, operator's schedule, batch size, dispatching rule and lot release control. Instigating this experimentation brings the major influencing variables and the most desirable lot release pattern.

Keywords – Simulation, lot release control, re-entrant flow, semiconductor, wafer fabrication

1 Introduction

Semiconductor industry has long been a driving force behind major advances in computing and electronics. It is one of the fastest growing and most significant industries due to its effect on accelerating the advance in technology and the resulting effect on world's economy. Semiconductor industry differs from many other manufacturing environments by the complexity and the variability of its processes. The entire semiconductor industry is very sensitive to the economic and trade climates. It is characterized by extremely short product life cycles, frequently decreasing profit margins and intense competition. In addition to this, the cycle time in semiconductor manufacturing is long in general due to the trade-off of waiting time in exchange for high equipment utilization in a factory of unreliable equipment. In general semiconductor manufacturing flow can be divided into four stages: wafer fabrication, wafer probe, assembly or packaging and final testing.

Wafer fabrication is arguably the most technologically complex, competitive and capital intensive stage of semiconductor manufacturing. This process is highly re-entrant and involves hundreds of machines, restrictions, and processing steps.

Rashmi Singh
Department of Management Studies
Indian Institute of Science
E-mail: rashmi@mgmt.iisc.ernet.in

Dr. M. Mathirajan
Department of Management Studies
Indian Institute of Science
E-mail: msdmathi@mgmt.iisc.ernet.in

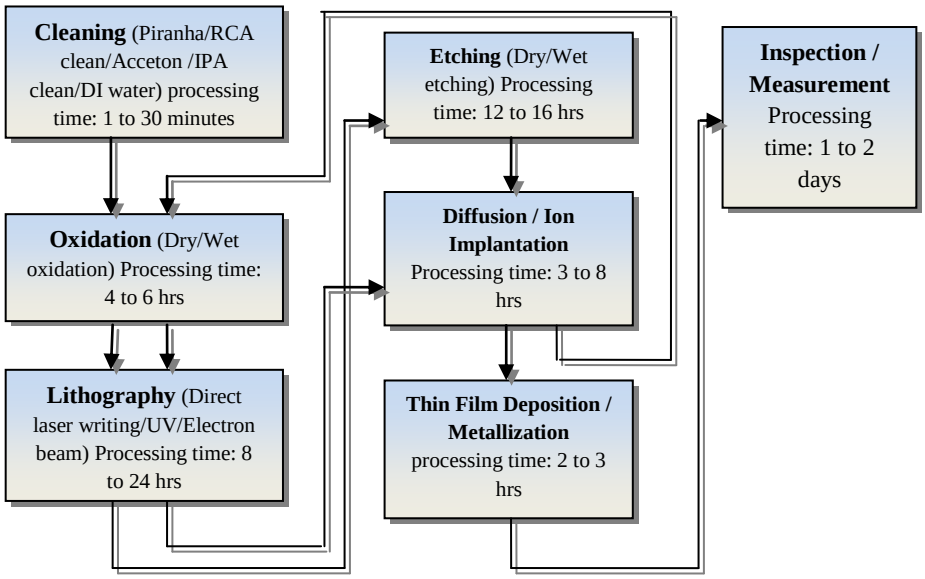


Figure 1: Basic Operation in Wafer Fab

In wafer fabrication, the layers of the ICs are fabricated onto raw silicon wafers. The manufacturing procedure for each layer involves a complex sequence of processing steps, with the number of operations typically in hundreds that includes: cleaning, oxidation/thin film deposition, photolithography, etching, diffusion/Ion implantation, inspection and measurement. These operations must be performed in a very clean environment known as wafer fab. The basic sequence of operations in wafer fabrication is presented in Figure 1. Sequence of operations may vary considerably for different products. In addition, random yield, rework, diverse equipment, availability of data, maintenance and unbalanced production facilities put together further intricacy in wafer fabrication process. Interested readers are referred to [22] for detail description on wafer fabrication complexities.

Each wafer lot visits the same workstation several times at different stages of processing, before exiting the system. This type of flow is known as re-entrant flow which is a distinguishing characteristic of semiconductor wafer fabrication manufacturing systems. In addition, the machines used for processing jobs are extremely expensive, some as high as US $40 million, and thus are scarce resources. In such a volatile scenario, maintaining a competitive advantage and remaining profitable, in operational terms necessitates the development of a control paradigm that will allow the effective and efficient deployment and operation of contemporary fabs.

In this paper we provide an overall cause and effect analysis of the behaviour of some key parameters in semiconductor wafer fabrication, which include cycle time, WIP level and throughput in response to changes in the Intel Mini Fab environment. For this purpose a simulation model is constructed for the Intel Mini-Fab using Arena simulation software. The Intel Mini Fab has been selected for this study as it captures the challenges involved in scheduling the highly re-entrant semiconductor wafer fabrication flow lines. Instigating this experimentation brings the major influencing variables and the most desirable lot release pattern.

The remainder of this paper is organized as follows: Section 2 summarises the literature involving shop floor control strategies and simulation model in wafer fabrication industry. Section 3 presents a proposed simulation model for Intel Mini Fab. The experimental performance evaluation and the results are presented and discussed in section 4. Finally, the conclusions and the plan for future research are presented in section 5.

2 Literature Review

In semiconductor manufacturing wafer fabrication is the most technologically complex and capital intensive process found today in existence. Therefore, a lot of dynamic shop floor control techniques have been stated in publications over the last years to improve the performance of semiconductor wafer fabrication. Given the complexity of semiconductor wafer fabrication, simulation emerges as powerful technique that describes the detail interactions among elements of such a manufacturing environment [13]. In addition, traditional techniques through mathematical models or even deterministic models are simply not adequate to analyse these complex manufacturing environments [1]. Consequently, discrete event simulation is used in evaluating such a complex system. Moreover, simulation offers the advantage of developing a feasible and accurate schedule in shorter computation times compared to some of the other techniques [15 and 10].

In Wafer fabrication industry several simulation studies are reported by different authors in the literature [3, 24, 8, 4, 9, 2, 12, 17, 7 and 21]. A lot of researcher's have proposed a new job release control rule to improve the wafer fabs performance in terms of cycle time, WIP level and throughput. Furthermore, they compared empirically through simulation their proposed job release control rule with other input control rules on several semiconductor wafer manufacturing job shops with favourable results [3, 24, 8 and 4]. Kim et al. [9] have recently focuses on lot release control, mask scheduling and batch scheduling collectively to improve the average cycle time, WIP level and throughput rate of wafer fabs. In addition to this a new input control rule is proposed by Chern et al. [2] to decrease setup time and increase throughput in the photolithography area without increasing cycle time of the wafer fabs. Furthermore, Lin et al. [12] presents a dynamic releasing scheme (D-Roll), which determines when and which wafer lot should be released into the shop floor using the concept of rolling correction. Several studies [17, 7 and 21] have utilized an input control and sequencing rule or both in wafer fabs to improve its performance in terms of cycle time, WIP level and throughput. A point on which many authors seem to agree is that the improvement in wafer fabs performance can be achieved with input control rule or sequencing rule or both.

Although to improve the performance of wafer fabs system**,** it is important and necessary to analyse the impact of different decision variables on main production process parameters, such as cycle time, WIP level and throughput. This has been ignored in the previous research studies. Though, there is one study reported in the literature by Chao and Shivakumar [16] that analyses the effect of arrival distribution, batch size, downtime pattern and lot release control on selected parameters such as cycle time, WIP level and equipment utilization rates. However, the simulation experimentation is not done in a realistic setting of wafer fabs. Furthermore, the breadth of input decision variables which has been investigated is very limited. Therefore, the objective of this study is to evaluate the effects of different input variables on selected parameters, such as cycle time, WIP level and throughput in realistic settings of wafer fabs. The selected variables include arrival rate, arrival distribution, processing time, maintenance schedule, operator schedule, batch size, dispatching rule and lot release control. For each variable, different magnitude of their parameters would be modelled to analyse the effects on the processes. In this way, it could be evaluated which variable would be the most influential, and thus possible measures could be suggested to improve the performance of the fab.

There are three aspects to keep in mind in order to appreciate the effort put forth in this study. The first is the way we modelled Intel Mini Fab to mirrors the complexity of the actual semiconductor fabrication in terms of including factors such as re-entrant flow, batching, operators, transporters and set-up operations. The second is the breadth of the input decision variables that has been investigated in this study. Finally, the findings that can benefit semiconductor wafer fabs.

3 A Simulation model for Intel Mini Fab

The Intel Mini Fab is selected for the study because it exhibits all the characteristic features of real semiconductor wafer fabs such as re-entrant loops, operators, transporters, batching, failures, preventive maintenance, setups, disparate processing, loading and unloading. Moreover, it is a benchmark system for all kinds of semiconductor research and also is a popular one used by several authors for evaluating different sequencing rules [7, 21 and 25].

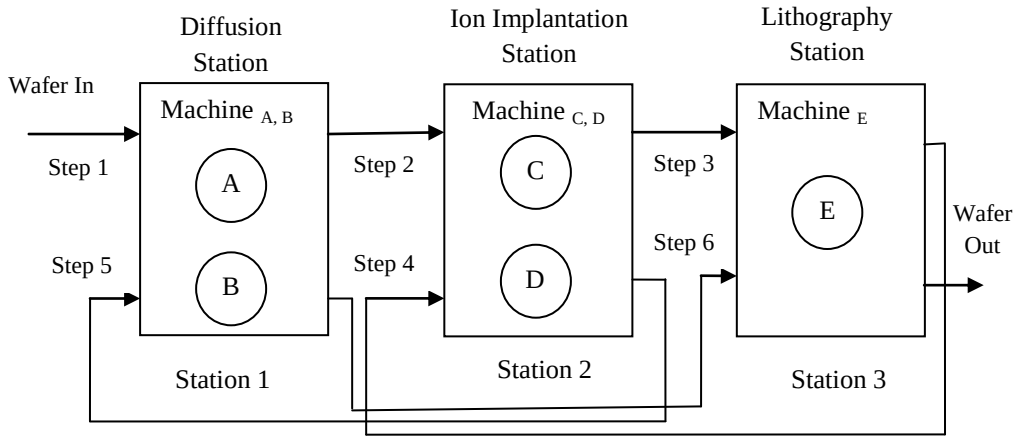


Figure 2: Intel five-machine six steps Mini-Fab

Intel Mini-Fab includes six processing steps and five machines distributed in three work stations as presented in Figure 2. The three processing workstation includes diffusion, ion-implantation and lithography respectively. Diffusion workstation consists of two machines A and B that can process batch of 3 lots at a time and it serves step 1 and 5. Ion-implantation workstation consists of two machines C and D and it can process one lot at a time and serves step 2 and 4. Lithography workstation consists of one machine E that process one lot at a time and serves step 3 and 6. The three critical operations performed per layer, namely diffusion, ion-implantation and lithography are considered in the Mini Fab [4].

In each week, the nominal arrival rate is 84 jobs per week, with 51 of the arrivals being product A, 30 are product B and the rest are test wafers. All product types follow the same predefined route, illustrated in Figure 2. For detail description of Intel Mini-Fab, the interested readers are referred to [6].

Individual characteristics such as step number, processing time, loading and unloading time, batch size and number of machines per station for three workstations are presented in Table 1. The operation of loading and unloading before and after the processing at each machine is done by operators. In addition, for processing steps 3 and 6 setup is provided by the operators depending upon changes of processing step to be performed and type of product to be processed.

The Mini-Fab operates the 24 hours of the day, 7 days of week. Each day of operations is composed of two shifts of 12 hours. In the Mini-Fab machine failures and emergency repairs occur as random events. In particular, only machines C and D can have failures that occur every 50 ±26 hours and the repair time requires 420 ± 60 minutes. Preventive maintenance (PM) operations are included in the model by the specification of PM tasks for each machine in the system. The specification of the PM tasks for each machine of the Mini-Fab is presented in Table 2.

Table 1: Individual Characteristics for Workstation in Mini-Fab

S.No	Workstation	Individual Characteristic's					
		Step No.	Processing Time (minutes)	Loading Time (minutes)	Unloading Time (minutes)	Batch Size	No. of Machines per station
1.	Diffusion	S1	225	20	40	3	2
		S5	255	20	40	3	
2.	Ion Implantation	S2	30	15	15	1	2
		S4	50	15	15	1	
3.	Lithography	S3	55	10	10	1	1
		S6	10	10	10	1	

*****For Detail Description of Intel Mini Fab *aar.faculty.asu.edu/research/**intel**/papers/**fab**spec.html*

Table 2: Preventive Maintenance Specification for Machines in Mini-Fab

Machine's	Preventive Maintenance Specification
A and B	75 minutes every day
C and D	120 minutes every shift
E	30 minutes every shift

*****For Detail Description of Intel Mini Fab *aar.faculty.asu.edu/research/**intel**/papers/**fab**spec.html*

In our simulation model, each lot entering the segment of the fab has a process flow that consists of 6 total operations at three different workstations. In model, the arrival of jobs is considered as deterministic with inter-arrival mean of 120 minutes, which is equivalent to 84 jobs per week. In addition, the three different product types (P_a, P_b and test wafers (TW)) are produced that follow an empirical distribution with probabilities 0.61, 0.36 and 0.03 respectively. It is assumed that the lot size is constant throughout the study and one lot is equal to one job. Rework is not considered. In addition, two operators referred as PO1 and PO2 are modelled for loading and unloading operations as well as to provide setups at respective workstations. However, operators travelling time is not considered. To exclude blocking from the model, the buffers are modelled to have infinite capacity. Technician is not considered. For batching similar type of products are batched together for the same production step at diffusion workstation. Machine restriction is not considered for the test wafers to avoid the complexity. Moreover, each operator is equally efficient at performing their tasks. The machines have different processing times and it determines correct processing time via the production step of the product it has received. The schedules for preventive maintenance are deterministic and repeated every day or shift. The unscheduled or random breakdowns for machines C and D at Ion-Implantation station are uniformly distributed with a minimum of 24 hours and maximum of 76 hours. The repair time is uniformly distributed with a minimum of 6 hours and maximum of 8 hours. Hence, it can be concluded that the developed simulation model is stochastic in nature.

Verification and Validation for Proposed Simulation Model

The simulation model is build starting with a simple version that contain the basic production processes and subsequently extending it with additional complexity until every aspect of the Mini Fab has been included in the model. At each and every level, the computer program is

Table 3: Partial Trace Data for the Verification of Mini Fab

Entity Number	Event Start Time	Event End Time	Time (hrs)	State	Step	Workstation
Lot_1_1	13/01/2015 00:00:00	13/01/2015 05:36:00	5.6	Waiting	**1**	Diffusion
Lot_1_1	13/01/2015 05:36:00	13/01/2015 09:41:00	4.0833	Processing	1	Diffusion
Lot_1_1	13/01/2015 09:41:00	13/01/2015 10:21:00	0.6667	Waiting	1	Diffusion
Lot_1_1	13/01/2015 10:21:00	13/01/2015 10:36:00	0.25	Waiting	2	Ion Implantation
Lot_1_1	13/01/2015 10:36:00	13/01/2015 11:06:00	0.5	Processing	2	Ion Implantation
Lot_1_1	13/01/2015 11:06:00	13/01/2015 11:21:00	0.25	Waiting	2	Ion Implantation
Lot_1_2	13/01/2015 02:48:00	13/01/2015 05:36:00	2.8	Waiting	1	Diffusion
Lot_1_2	13/01/2015 05:36:00	13/01/2015 09:41:00	4.0833	Processing	1	Diffusion
Lot_1_2	13/01/2015 09:41:00	13/01/2015 10:21:00	0.6667	Waiting	1	Diffusion
Lot_1_2	13/01/2015 10:21:00	13/01/2015 10:36:00	0.35	Waiting	2	Ion Implantation
Lot_1_2	13/01/2015 10:36:00	13/01/2015 11:21:00	0.5	Processing	2	Ion Implantation

checked and debugged in steps to verify the simulation model. In addition to this, the validation is performed further by comparing the theoretical value of cycle time from the dataset and the cycle time obtained from the simulation. The theoretical value of cycle time for one job as calculated from the dataset is 14.95 hour, and the result from simulation showed a cycle time of 15.0 hour, a slight increase of 0.05 hour is there which is negligible. However, the slight increase of 0.05 hour in cycle time is due to the operator non availability that cannot be captured in theoretical calculation of the cycle time. Therefore, it can be concluded that the model reflects sufficiently accurately the actual cycle time of the processes.

Furthermore, the technique of "trace" is employed to verify the proposed simulation model. This is one of the most powerful technique that can be used to debug a discrete event simulation program Law and Kelton [2009]. For instance, considering the single part type case, few lots of part type 1 are simultaneously released into the system at the beginning of simulation run, and then the operation events on these lots are traced. The event tracing record is compared to the expected event schedule, which can be derived from the route illustrated in Figure 2. The Table 3, shows the partial trace data of the release lots. The model verification is confirmed since the achieved event schedule is consistent with the intended one.

4 Experimentation

In this section, we will investigate the effects of altering the different input decision variables on cycle time, WIP level and throughput. The input variables include arrival rate, arrival distribution, processing time, maintenance schedule, operator's schedule, batch size, dispatching rule and lot release control. For each variable, different magnitude of their parameters are tested by varying only one variable at a time. The detail experimental environment is summarised in Table 4.

Table 4: Summary of Experimental Environment

Input Variable's	No of levels	Levels
Arrival Rate	8	Inter-arrival mean with (2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2, 3.4)
Arrival Distribution	7	Deterministic (2.8), Random, Poisson(2.8), Normal(2.8, 0.5), Normal (2.8, 2), Uniform(2.66, 2.94) & Uniform(2.52, 3.08)
Processing Time	2	Deterministic, Random
Preventive Maintenance Schedule	6	Schedule 1, Schedule 2, Schedule 3, Schedule 4, Schedule 5, Schedule 6
Operator's Schedule	4	Schedule 1, Schedule 2, Schedule 3, Schedule 4
Batch Size	5	2, 3, 4, 5, 6
Dispatching Rule	5	FIFO, LIFO, EST, SPT, LPT
Lot Release Control	2	Push, CONWIP

Arena simulation software is used to build the model and conduct these experiments. In the course of the simulation, each scenario is tested for 200 replications of 9600 hours length each. The first 4800 hours are discarded to avoid the influence of transient state behaviour. This number and length of replications provided uniformly good statistical precision across the outputs (95% confidence interval half widths within 3% of the respective sample means). Machine reliability is the main factor behind variability, and different random stream seed increments were used in each run. The data obtained after the 4800 hours are used for the performance analysis. The performance indices considered in this experiment are cycle time, WIP level and throughput.

The significance of cycle time performance in semiconductor manufacturing is a well-known fact. Cycle time (also known as throughput time, flow time, or sojourn time) is the time elapsed between a job entering the facility and leaving the facility as a finished product, consisting of processing time, transportation time between workstations, set-ups time, loading and unloading time and waiting time in queues. The average WIP level is defined as the average number of jobs present in the Mini Fab during the non-transient run time period and throughput is the number of jobs that came out after the last step in the process. In this study, throughput is measured on weekly basis to facilitate the comparison with the target throughput of 84 jobs per week. The weekly throughput is calculated for the analysis by dividing the throughput value with 200 and then multiplying the value by 7. This is because, the simulation run length is 400 days or 9600 hours and the warm-up period is selected as 200 days or 4800 hours to avoid the effect of transient bias.

4.1 Impact of Arrival Rate

In this section, we investigated the effects of altering the arrival rate on cycle time, WIP level and weekly throughput. The arrival rate is the number of arrivals per unit of time. It can be measured as the arrival rate or the inter-arrival time (time between arrivals). In reality arrival rates are highly variable and difficult to predict. In this model we choose a mean arrival rate as 0.5 lots per hour, to achieve a throughput of 84 lots per week. The steady state behaviour is observed which indicates that the lithography workstation is utilized heavily among all the workstations. Furthermore, it is observed that even though lithography workstation is utilized heavily but still there are many jobs waiting in the queue to get hold of lithography machine for processing. Subsequently, it can be concluded that the bottleneck workstation of the flow line is lithography workstation. It can be observed from Table 5 that

the achieved throughput is 50.81 products per week or 60.48 % of the required throughputs with inter-arrival mean of 2.0 hours. Moreover, it is observed that the mean cycle time of the products is increasing, because the work in process is increasing. The processes have much idle time and the machines have to wait for loading and unloading operations. Therefore, we can conclude that the control of the flow line is poor.

Table 5: Performance of Mini Fab with different Arrival Rate

Inter-arrival means	Average Cycle Time (hr)	Average WIP	Weekly Throughput
Deterministic (2.0)	2839.11	1419.80	50.81
Deterministic (2.2)	2151.55	978.50	53.45
Deterministic (2.4)	1376.05	574.27	56.69
Deterministic (2.6)	566.71	219.20	59.70
Deterministic (2.8)	45.92	17.80	60.04
Deterministic (3.0)	40.36	14.73	56.01
Deterministic (3.2)	38.67	13.27	52.50

The implementation of improved control in the flow line is required to increase the achieved throughput of the line by 39.52% to meet the target throughput. To determine the maximum throughput that can be achieved with FIFO control, the input of the flow line is decreased until the work in process is just not increasing in steady state during the simulation. The ratio of the product types is kept identical to the ratio in the Mini Fab. It can be observed from Table 5, that the maximum throughput with FIFO control is achieved at inter-arrival mean of 2.8 hours and it is equal to the 60.04 products per week or 71.47% of the target throughput. This is because no work is wasted on the semi-finished products that form the increasing work in process. Moreover, it can be observed further that the parameters of average value of both cycle time and WIP level are reasonably good for inter-arrival mean of 2.8 hours. The performance measure of interest is calculated as average of 200 replications, one per run.

4.2 Impact of the Arrival Distribution

In this section, we will investigate the effects of altering the distribution of arrival on cycle time, WIP level and weekly throughput. In real life, there is variability in the inter-arrival times of the incoming jobs due to the uncertainty in multiple external factors such as demand, customer order schedules and work in process inventory and transportation irregularities. However, we consider a scenario whereby the fab manager could decide the arrival rate based on maximum throughput that can be achieved with FIFO control in Mini Fab. Assuming the mean inter-arrival time equals to 2.8 hours, we have to decide the characteristic for lot release. Consequently based on the frequently used arrival or inter-arrival distributions in the literature, we have experimented six distributions to release the job on the shop floor. These six distributions include Poisson (2.8), normal (2.8, 0.5), normal (2.8, 2), uniform (2.66, 2.94), uniform (2.52, 3.08) and random. The parameters for uniform distribution are selected with an offset of 5% and 10% of the inter-arrival mean value on both sides. For comparison purpose, these six arrival distributions are compared with deterministic arrival rate with inter-arrival mean of 2.8 hours. The behaviour of these release patterns were tested based on FIFO control and the results are presented in Table 6.

Table 6: Performance of Mini Fab with Different Arrival Distribution

Arrival Distribution	Average Cycle Time (hr)	Standard Deviation of Cycle Time	Average WIP	Weekly Throughput
Deterministic (2.8)	45.92	26.85	17.80	60.04
Random	113.57	51.67	42.51	60.02
Poisson (2.8)	69.98	34.71	26.53	60.09
N (2.8, 0.5)	47.68	27.01	18.45	60.05
N (2.8, 2)	55.83	31.84	20.88	58.56
U (2.66, 2.94)	45.85	26.98	17.78	60.03
U (2.52, 3.08)	46.08	27.08	17.86	60.04

The performance of random release distribution is obviously the worst. This result has been proven mathematically by Shivakumar [20]. The result of deterministic arrival rate is best in terms of each performance measure value because there is less variability in input rate. Furthermore, it can be observed from the Table 6, that under the Poisson distribution, the parameters of average value of both cycle time and WIP level are the worst except for that of random scenario. It is interesting to note the difference between the standard deviation of the two results using normal distribution, although they have identical mean frequency. The one with the larger deviation produce longer cycle time and also much higher standard variation in cycle time. This result corresponds to the theory that variability would increase the cycle time of production [5, 19 and 20]. The results shown in the uniform distribution have the same trend as that of the normal distribution. However, given an opportunity the fab manager should go for uniform distribution to release the job on the shop floor because it is giving the best results in terms of cycle time and WIP level. Moreover, less variability is being observed in uniform distribution because we are getting close results with an offset of 5% and 10% of the inter-arrival mean value on both sides. Therefore, we can conclude that the uniform distribution is more robust in nature. The similar observation is reported by Chao and Shivakumar [16].

4.3 Impact of Processing Time

It is well known fact that the wafer fabrication typically involves different types of products. These product mix environment plays an important role in adding the variability to the processing time. In order to address this issue we considered two possibilities of processing time one is deterministic and the other one is random.

Table 7: Performance of Mini Fab with Different Processing Time

Processing Time	Average Cycle Time (hr)	Average WIP	Weekly Throughput
Deterministic	45.92	17.80	60.04
Random (expo)	53.64	20.87	60.00

Two different simulation experiments were performed under two different scenarios for processing time that is deterministic and random. The random processing times were assumed to be exponentially distributed with means equal to the processing times indicated in Table 1. It can be observed from Table 7 that under random processing time, the parameters of average value of both cycle time and WIP level are the worse than deterministic processing

time. This result again corresponds to the theory that variability would increase the cycle time of production [5, 19 and 20].

4.4 Impact of Schedules for Maintenance

The production equipment used in semiconductor manufacturing is technologically sophisticated. Therefore, it requires extensive preventive maintenance (PM) to ensure that the equipment would operate in optimum conditions. Preventive maintenance is the scheduled process of intentionally taking a tool offline for routine maintenance. The machines will not be available during the preventive maintenance and this kind of downtimes is referred to as scheduled downtimes. It is used to increase the tool reliability and availability. Six different simulation experiments were performed under six different and arbitrary PM schedules. The Table 8 provides the details of the six PM schedules utilized daily in the experiments where the start times for the PM tasks are given in the usual time format hours:min:secs and assuming that the first work shift starts at 00:00:00.

Table 8: Description of PM Schedules for Machines in Mini Fab

Machine	PM Task Start Time					
	Schedule 1	**Schedule 2**	**Schedule 3**	**Schedule 4**	**Schedule 5**	**Schedule 6**
A	06:00:00	06:00:00	06:00:00	06:00:00	00:00:00	03:00:00
B	12:00:00	08:00:00	10:00:00	06:00:00	02:00:00	05:00:00
C	00:00:00	06:00:00	04:00:00	02:00:00	04:00:00	00:00:00
D	08:00:00	04:00:00	08:00:00	10:00:00	06:00:00	08:00:00
E	03:00:00	03:00:00	02:00:00	04:00:00	08:00:00	10:00:00

Table 9: Performance of Mini Fab with Different PM Schedules

PM Schedules	Average Cycle Time (hr)	Average WIP	Weekly Throughput
Schedule 1	45.92	17.80	60.04
Schedule 2	46.02	17.83	60.03
Schedule 3	45.93	17.81	60.03
Schedule 4	45.84	17.77	60.03
Schedule 5	46.21	17.89	60.02
Schedule 6	46.05	17.84	60.03

It would seem intuitive that the more sophisticated the maintenance schedule, the higher the asset availability would be and therefore the performance would be better. However, it can be observed from Table 9 that the parameter of the performance metrics for schedule 1 is the best among all the schedules and it is worst for schedule 5. Though, changes in parameters of the performance metrics are not very significant with the change in maintenance schedule. Furthermore, it is observed that the schedule should be prepared intuitively according to the machine processing time and their maintenance need. Specifically, the preventive maintenance should be provided to the machines when they are least needed.

4.5 Impact of Operator's Schedule

The operators are required to load and unload the jobs in the machine at the beginning and at the end of process. However, operators will not be available all the time as they have three off times of 60 minutes each. Subsequently a schedule is prepared for the operators in four different ways as given in Table 10 to provide details of their accessibility.

Table 10: Description of Operator Schedules for Mini Fab

Operators	Operator's Break Start Time			
	Schedule 1	**Schedule 2**	**Schedule 3**	**Schedule 4**
OP0	(12:00:00, 3:00:00, 09:00:00)	(12:00:00, 02:00:00, 08:00:00)	(02:00:00, 07:00:00, 10:00:00)	(03:00:00, 07:00:00, 11:00:00)
OP1	(01:00:00, 05:00:00, 10:00:00)	(04:00:00, 09:00:00, 1:00:00)	(12:00:00, 06:00:00, 11:00:00)	(12:00:00, 04:00:00, 08:00:00)

Table 11: Performance of Mini Fab with Different Operator's schedules

Operator's Schedules	Average Cycle Time (hr)	Average WIP	Weekly Throughput
Schedule 1	45.98	17.89	60.01
Schedule 2	41.69	16.34	60.01
Schedule 3	62.11	23.62	60.02
Schedule 4	45.92	17.80	60.04

It can be observed from Table 11, that the parameters of average value of both cycle time and WIP level are almost same for schedule 1 and schedule 4. This is because there is not much difference in their respective schedules as well. It is interesting to note that the performance of schedule 3 is really worst among all the schedules. The reason for this is that the operators off time are scheduled independently of the maintenance need of the machine, whereas it should be scheduled when operators are least needed. The parameters of average value of both cycle time and WIP level are best for schedule 2. The reason for this is simple that the operator off time is scheduled when operators are least needed. Therefore, we can conclude that the operator schedule significantly influence the parameters of the system performance metrics. Furthermore, it is observed that the operators off time should be scheduled based on the maintenance need of machines.

4.6 Impact of Batch Size

The predominance of batch processing systems in a semiconductor wafer fabrication facility is a well-known fact. The processes of the wafer fabrication involve a number of batch processing operations such as oxidation, diffusion and deposition which are performed by the batch processing machines. For instance if batch sizes are too small, then the batch processing machines may run out of capacity. On the other hand, if batch sizes are too large, then the waiting times to form batches may increase. Consequently, there is no specific rule to set batch sizes of all products simultaneously so that the performance of the system is improved. A thorough batch size analysis requires extended study and is beyond the scope of this paper.

Table 12: Performance of Mini Fab with Different Batch Size

Batch Size	Average Cycle Time (hr)	Average WIP	Weekly Throughput
Batch (2)	718.56	380.68	54.26
Batch (3)	45.92	17.80	60.04
Batch (4)	53.48	20.10	60.04
Batch (5)	60.35	22.35	60.03
Batch (6)	67.90	24.90	60.04

To determine the optimal batch size we have experimented with five different batch sizes in the range of two to six as given in the Table 12. It can be observed that the batch size of two gives the worst results. This might be because the batch size of two is very small and the processing time of diffusion furnace is large that is 4.25 hours. It is interesting to note that the batch size of three is giving the best result among all the batch sizes tested for Intel Mini Fab. Furthermore, it is noticed that as we are increasing the batch size more beyond three the flow time is increasing because the work in process is increasing. Consequently, we can conclude that batch size adds variability into a system because jobs wait to form batch and upon service completion jobs are released to downstream machines.

4.7 Impact of Dispatching

Dispatching in wafer fabrication is enviable task. In reality, dispatching rule determines the sequence in which jobs in front of machine are processed according to some variable. Since, the choice of dispatching rule is not the focus of this study. Therefore, most commonly used dispatching rules such as First-In-First-Out (FIFO), Last-In-First-Out (LIFO), Earliest Start Time (EST), Shortest Processing Time (SPT) and Longest Processing Time (LPT) found in the literature are experimented in simulation. The lithography is found to be bottleneck workstation in Intel mini fab so; dispatching rules are applicable only at lithography workstation.

Table 13: Performance of Mini Fab with Different Dispatching Rules

Dispatching Policies	Average Cycle Time (hr)	Average WIP	Weekly Throughput
FIFO	45.92	17.80	60.04
LIFO	57.30	21.85	60.07
EST	50.30	19.45	60.03
SPT	44.79	17.40	60.02
LPT	50.69	19.52	60.05

It can be observed from Table 13 that the LIFO rule is giving worst performance as compared to others because waiting time is more for jobs which are arriving earlier at lithography. Henceforth, the flow time and work in process is more for this dispatching rule. EST and LPT rule is giving similar results. Actually, in case of EST, the job will be coming for production step 3 which is having longer processing time. Henceforth, the result of EST dispatching rule is very close to the LPT dispatching rule. Among all the dispatching rules, SPT is the best dispatching rule in terms of cycle time and work in process because it is giving preference to the job having the shortest processing time.

4.8 Impact of Input Control

To evaluate the effect of release control in Mini Fab Push and CONWIP release method is discussed. The PUSH release control loads the job shop uniformly (or with any other distribution) based on demand. Job is moved through the facility in a first-come-first-serve manner. No feedback on the status of the shop floor is employed. It is an open loop method. In this section, we will discuss the scenario of CONWIP as push is already being discussed in impact of arrival distribution and the same is presented in Table 15. In CONWIP release policy, the number of jobs in the system is held constant. A new job is released into the system whenever a job finishes processing at the final workstation. The results are shown below in Table 5 under different WIP levels. One of the distinct benefits of CONWIP is its simplicity to adjust and to implement. This property becomes more advantageous when the system environmental conditions are constantly changing. It can be observed from Table 14 that input control has a significant impact on wafer fabrication performance measures.

Table 14: Performance of Mini Fab with CONWIP Release Control

WIP levels	Average Cycle Time (hr)	Standard deviation of Cycle time (STD)	Weekly Throughput
WIP 10	37.63	37.33	44.61
WIP 12	39.37	32.39	51.24
WIP 14	42.20	29.25	55.73
WIP 15	43.71	28.22	57.65
WIP 16	45.32	27.46	59.29
WIP 17	44.24	26.40	60.03
WIP 18	43.74	26.97	60.02
WIP 20	43.84	26.93	60.02
WIP 22	43.76	26.88	60.01
WIP 24	43.74	26.78	60.01

Table 15: Comparison of CONWIP and other Arrival Distribution (Push)

Release Control	Arrival Distribution	Average Cycle Time (hr)	Standard Deviation of Cycle Time	Weekly Throughput
Push	**Deterministic (2.8)**	45.92	26.85	60.04
	Random	113.57	51.67	60.02
	Poisson (2.8)	69.98	34.71	60.09
	N (2.8, 0.5)	47.68	27.01	60.05
	N (2.8, 2)	55.83	31.84	58.56
	U (2.66, 2.94)	45.85	26.98	60.03
	U (2.52, 3.08)	46.08	27.08	60.04
CONWIP	**WIP (17)**	44.24	26.40	60.03

It can be observed from the Table 14, that initially average cycle time is increasing with the increase in WIP level. However, it is interesting to note that the average cycle time started decreasing with WIP (17) level. Moreover, the values of standard deviation of cycle time and weekly throughput is best at WIP (17) level. Furthermore, it can be observed that

there is not much change in the values of average cycle time, standard deviation of cycle time and throughput rate beyond WIP (18) level. Hence we can conclude that in our simulation model the best results are obtained for WIP (17). In reality, most common release control used in the wafer fabrication is the open loop control such as uniform start. However, it can be observed from Table 15 that the performance of CONWIP is much better than that of the other arrival distribution. This result corresponds to the conclusion of many authors in the literature that the input control has the greatest impact on cycle time and WIP level [24, 3 and 8]. In addition, we can conclude that any reasonable closed loop control performs better than open loop control. This result corresponds to the conclusion of Glassey and Resende [3]. This is because all the closed loop control rules adjust the arrival rate to the shop so that it is negatively correlated with the queue length at the bottleneck.

5 Conclusions

In this paper we constructed a simulation model of Intel Mini Fab using the Arena simulation software. The objective of the study is to evaluate the effects of different input variables on selected parameters, such as cycle time, WIP level and throughput rate. The input variables include arrival rate, arrival distribution, processing time, maintenance schedule, operator's schedule, batch size, dispatching rule and lot release control. A simple model of Intel Mini Fab is selected because it captures the challenges involved in scheduling re-entrant manufacturing line.

Most importantly, it is noticed in case of arrival distribution and processing time that the system performance decreases with the increase in system variability. This result corresponds to the theory that variability would increase the cycle time of production [5, 19 and 20]. Furthermore, the results show that the relationship between input variables and system performance metrics are extremely complex due to the complexity of semiconductor wafer fabrication. However, it is important to stress few facts that the operator off time should be scheduled according to the maintenance need of the machines rather independently. Preventive maintenance should be provided intuitively to the machines based on their processing time and maintenance need. Moreover, it has been observed that the choice of batch size and dispatching rule also affects the system performance. Therefore, a watchful decision is required by the fab managers while choosing batch size and dispatching rules on the shop floor.

But the most important conclusion drawn from the analysis is that the input control has the greatest impact on cycle time and WIP level in wafer fabrication. This conclusion is consistent with the argument by Wein, Glassey and Resende and Kim et al. [24, 3 and 8]. Moreover, it is noticed that closed loop release method works better than an open loop method such as uniform release rule. This is mainly because a closed loop release method is capable of adjusting the release decision responding to the dynamic events, which happen in the system due to stochastic factors.

The future research will focus on closed loop release methods, considering the real time status and uncertainties in the system. Furthermore, we would like to test all the previously developed closed loop release methods in realistic settings of wafer fabrication.

References

1. Arisha, Amr, Young Paul, Baradie, M.El and Hashmi, M.S.J, "Intelligent shop scheduling for semiconductor manufacturing," PhD Thesis, School of Mechanical and Manufacturing Engineering, Dublin City University, 2003.
2. Chern, ching-chin and Haung, Kwei-long, A heuristic control method for a single product, high volume wafer fabrication process to minimize the number of photomask changes. Journal of Manufacturing Systems; 23, 1; ABI/INFORM Global, pg 30, 2004.

3. Glassey, C.R., and Resende, M.G.C., "Closed-loop job release control for VLSI circuit manufacturing," IEEE Transactions on Semiconductor Manufacturing, Vol. 1,No. 1, pp. 36-46, 1988.
4. Glassey, C. R., Shanthikumar, J. G., and Seshadri, S., Linear control rules for production control of semiconductor fabs, IEEE Transactions on Semiconductor Manufacturing, vol. 9, no. 4, pp. 536-549, 1996.
5. Hopp, Wallace J., L. Spearman, Factory Physics: Foundations of Manufacturing Management, Irwin/McGraw-Hill, Inc, 2000.
6. Kempf, K, "Intel five-machine six-step Mini-Fab description," http://aar.faculty.asu.edu/research/intel/papers/fabspec.html, accessed on 17th Feb 2015.
7. Khouly, Ingy A. El, Kilany, Khaled S. El and Sayed, Aziz E. El, "Effective scheduling of semiconductor manufacturing using simulation," World Academy of Science, Engineering and Technology Vol: 5, 2011.
8. Kim Jongsoo, Leachman Robert.C and Suhn Byungkyoo: Dynamic Release Control policy for the Semiconductor Wafer Fabrication Lines. Journal of the Operational Research Society 47, 1516-1525, 1996.
9. Kim Dae, Yeong, Lee Ho, Dong and Kim Ug, Jung: A simulation study on lot release control, mask scheduling, and batch scheduling in semiconductor wafer fabrication facilities. Journal of manufacturing systems, vol. 17, No. 2, 1998.
10. Kiran, A.S., Simulation and Scheduling, In Handbook of Simulation, ed. J. Banks, 677-717, New York: John Wiley & Sons, Inc, 1998.
11. Law, Averill M., and Kelton, W. David, Simulation Modelling and Analysis,McGraw-Hill, Inc, 1991.
12. Lin Hsin-Yu, Tsai Hung Chih, Lee En Ching, Liu Kaung Sheng: A dynamic releasing scheme for wafer fabrication. International Journal of the Computer, the Internet and Management Vol. 15#1, pp 33 – 42, 2007.
13. Lou, Sheldon X. C. and Kager, Patrick W, "A robust production control policy for VLSI wafer fabrication," IEEE Transactions on semiconductor manufacturing, Vol. 2, No 4, 1989.
14. Matthias Thürer, Mark Stevenson, Cristovao Silva, Martin Land and Moacir Godinho Filho: Workload control and order release in two-level multi-stage job shops: an assessment by simulation, International Journal of Production Research,pg 1-14, 2012.
15. Mazziotti, B.W., and Horne, Jr., R.E., Creating a flexible, simulation-based finite scheduling tools. In Proceeding of the Winter Simulation Conference, 1997.
16. Qi, Chao and Sivakumar, A. I, Simulation based cause and effect analysis in semiconductor wafer fabrication, Journal of the Institution of Engineers, Singapore Vol. 44, Issue 4, 2004.
17. Qi, Chao, Sivakumar, A. I. And Gershwin, Stanley .B: An efficient new job release control methodology. International Journal of Production Research, Vol. 47, No. 3, 1, pg 703–731, 2009.
18. Sivakumar, A.I., "Optimization of cycle time and utilization in semiconductor Test manufacturing using simulation based, on-line, near-real-time scheduling system," Proceedings of the Winter Simulation Conference, 1999.
19. Sivakumar, A.I., "Simulation based cause and effect analysis of cycle time Distribution in semiconductor backend," Proceedings of the Winter Simulation Conference, 2000.
20. Sivakumar, A.I., Chong, C.S., "A simulation based analysis of cycle time Distribution and throughput in semiconductor backend manufacturing," Computers in Industry, Vol. 45, pp. 59-78, 2001.
21. Tabatabaei, R.A and Salazar, Carlos F. Ruiz, "Effective wip dependent (EWD) lot release policies: a discrete event simulation approach," Winter Simulation Conference, 2011.
22. Uzsoy, R., Lee, C.Y., and Martin-Vega, L.A., "A review of production planning and scheduling models in the semiconductor industry – Part I: system characteristics, Performance evaluation and production planning," IIE Transactions, Vol. 24, No. 4, pp. 47-60, 1992.
23. Uzsoy, R., Lee, C.Y., Martin-Vega, L.A., "A review of production planning and Scheduling models in the semiconductor industry – Part II: shop-floor control," IIE Transactions, Vol. 26, No. 5, pp. 44-55, 1994.
24. Wein, L.M., "Scheduling semiconductor wafer fabrication," IEEE Transactions on Semiconductor Manufacturing, Vol. 1, No. 3, pp. 115-130, 1988.

MISTA 2015

Optimisation of Staff Absences

C. Klöcker · J. Ostler · P. Wilke

Abstract Most timetabling problems to date focus on the *presence* of employees like nurses or teachers or, in general, resources like rooms or classes. Although, for example, in nurse rostering attention is paid to time intervals in which nurses are on a holiday, it seems that – to the best of our knowledge – no fundamental approach to pure *absence planning* exists. In order to fill this gap, we introduce a novel approach to staff absence optimisation through leave request approval or rejection: the Absence Scheduling Problem (ASP). Using the Erlangen Advanced Time Tabling Software (EATTS) framework we implemented a very flexible absence request model that includes *alternatives* to *first choice* requests, multiple *periods* for a single request, and sophisticated possibilities to specify the requested time slots within each period. In this paper we describe our data model and the corresponding problem *constraints*, like fulfilling minimal staff or absence quota conditions, including a mathematical model for both.

Additionally, we compare the performance of Tabu Search (TS) and Simulated Annealing (SA) paired with two different *move pools*, one of them including *repair moves*, on the ASP. For this purpose, we first describe our test data generator and present test results for problem sizes of 100 and 250 employees afterwards. We show that the 'advanced' move pool with repair moves on the one hand helps TS to find slightly better solutions but, on the other hand, actually hinders SA's optimisation process for the smaller problem size while having a positive effect for the 250 employees problem.

1 Introduction

In the broad field of timetabling problems, two big problem classes have been receiving most of the scientific attention to date: nurse rostering and school timetabling. We introduce a new problem class which has certain similarities with these well-established problems, but – which makes it worth being investigated – some unique characteristics, too: optimisation of staff absences, or, in short, the Absence Scheduling Problem (ASP). The two problems

Multi Criteria Optimisation Group
Pattern Recognition Lab
Computer Science Dept.
University Erlangen-Nuernberg, Martensstrasse 3, 91058 Erlangen, Germany
E-mail: Peter.Wilke@FAU.DE

mentioned first have fixed time slots for *shifts* or *classes* with relatively constant interdependent time scopes for each assignable resource; a nurse must have a certain resting time between two shifts and should work the same shift for a certain number of days; school timetables may have certain restrictions for parallel events, i.e. during the same class time slot, or requirements targeting the whole week.

By contrast, our absence planning model, described in detail later on, potentially has a great freedom of choice when it comes to granting leave. A leave request may have *alternatives* from which the one to be granted has to be chosen, and each request is also very variable in terms of specifying the fitting time slots, which may be very numerous and diverse in duration: consisting of just a single day or, for instance, a two to three week leave with 21 possible starting days. Additionally, we define problem specific constraints that also have very diverse time scopes ranging from one day to the whole planning period and, besides, cause very high interdependencies between requests.

We implemented our absence planning system using the Erlangen Advanced Time Tabling Software (EATTS), a flexible framework with an own XML based description language for general timetabling problems [6].

2 The Problem

Timetabling problems usually consist of assigning resources to certain time slots. In nurse rostering, for example, nurses are assigned to shifts while in high school course scheduling teachers, rooms, and other resources are assigned to certain fixed time slots. In the ASP, the according approach is to allocate the employees that will be working to each day of the planning period , i.e. are not scheduled for a leave. An equivalent problem model, which we will be using, is the assignment of time slots to absence requests, wherein the time slots represent a time period in which the corresponding employee is on leave. This model is outlined in the following.

The time slots suitable for a certain absence request are specified by its *periods*. Let $P = \{p_1, \ldots, p_n\}$ be the set of days in the planning period where $\forall k \in \{1, \ldots, n-1\} : p_k, p_{k+1}$ are consecutive. A single period has a start date, p_s, and an end date, p_e, $1 \leq s \leq e \leq n$, a minimum duration, $d_{\min} \in \mathbb{N}$, and a maximum duration, $d_{\max} \in \mathbb{N}$, $1 \leq d_{\min} \leq d_{\max} \leq e - s + 1$, and a list of possible start days $S = \{p_{s_1}, \ldots, p_{s_m}\}, \forall k \in \{1, \ldots, m\} : s \leq s_k \leq e - d_{\min} + 1$. From this, the fitting time slots T result by all continuous time intervals that have a duration between the given minimum and maximum, start on one of the given start days and do not end after the end date of the period, i.e.:

$$T = \{(p_a, p_b) \mid a \leq b, d_{\min} \leq b - a + 1 \leq d_{\max}, p_a \in S, b \leq e\}$$

The planning problem thus is discrete having a granularity of one day, so a period with x days could have a maximum of $x(x+1)/2$ fitting time slots. Our period scheme could be used, for instance, to request a one week leave (duration) sometime in October (start and end date) starting on Tuesday or Thursday (possible start days) because on those days especially low priced flights would depart.

Each absence request has at least one period and, optionally, a number of other absence requests as *alternatives*; the base request will then be called the *first choice*. The *approval* of a request is equivalent to the assignment of one time slot to each period. Since periods of a request may overlap with periods of one of its alternatives, it is possible that a time slot assignment is ambiguous in terms of which request it approves – an example is given in

figure 1 – and the constraints, described later on, thus independently choose for every time slot the alternative where the lowest costs arise.

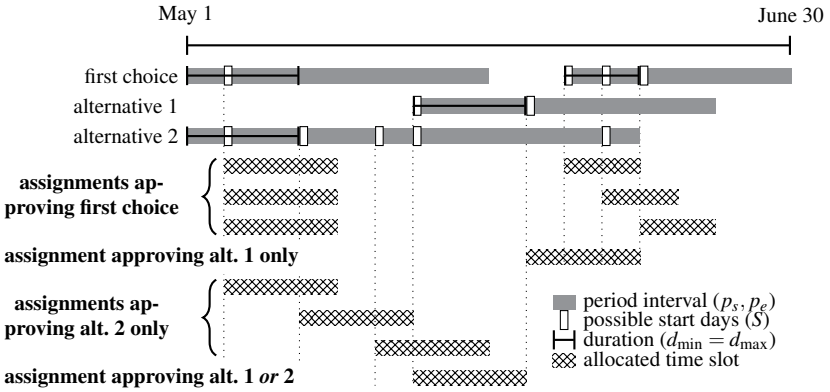


Fig. 1 Absence request with alternatives and all of its approving time slot assignments. A valid time slot has to start on a 'possible start day', match the given duration, and must not exceed the period interval. Note that the first choice can only be approved by assigning two time slots and that the first possible starting day does not have to correspond with the first day of the period.

3 Problem Specific Constraints

The first conflict of interests in the planning process in most cases arises from the fact that, on the one hand, all employees want their first choice requests to be approved and, on the other hand, only a certain number of employees may be on leave at once. With these two simple premisses the planning problem already becomes considerably hard. Lets say, that on each day of the planning period at least one employee must be present and that the rejection of an absence request yields costs of 1. For a single employee, the costs for them to be working at a time interval that is included in one of their requests would therefore be 1, otherwise 0. The (work) days of the planning period are thus fully covered by time intervals, i.e. sets of days, with cost 1 or 0 and the objective is to cover all days with at least one subset as to satisfy the minimal manning of 1. This optimisation problem is a weighted set covering problem which has been proven to be NP-hard [4].

3.1 Minimal Manning Constraint

In our planning framework, the 'minimal manning constraint' was implemented by defining *groups* and the number of employees required for each group on each day of the planning period. Employees may be members of an arbitrary number of groups and increment the actual manning of each of their groups by one when they are at work. The applications of the minimal manning constraint and the requirements themselves can be very versatile: the constraint can state, for instance, that each company division always needs at least one appointed first aider to be present, who, simultaneously, could fulfil another manning requirement for his actual job role; that in a special two week period 95% of staff have to be present for a corporate event; or that on every day at least one barista, three waiters and two

chefs have to be at work. If the minimal manning on one day is violated by a difference of x employees, the costs for this group are increased by $x^p, p > 1$, so the constraint will yield more costs if the manning is x employees short on one day, than if it is short one employee on x days respectively.

Lets consider a single group with manning requirements $R = \{r_1, \ldots, r_n\}, \forall k \in \{1, \ldots, n\}$: $r_k \geq 0$ for each day of the planning period. Let $A = \{a_1, \ldots, a_n\}$ be the actual number of employees not on leave in a certain leave schedule. The minimal manning constraint's costs, c_{man}, for *a single group* are then given by

$$c_{\text{man}} = \sum_{k=1}^{n} (c_k)^r, c_k = \max\{0, r_k - a_k\}$$

For our test setup we used $r = 1.2$.

3.2 Approval Constraint

The penalty for an absence request rejection, or for approving an alternative rather than the first choice, is given by the 'approval constraint'. We use the concept of *bonus points* to enable employees to prioritise certain first choice requests over others – their own as well as requests of different employees. All employees, at the start of each planning period, receive a fixed number of bonus points, β_{max}, and distribute them among their requests; in our model all bonus points are used up, but it could also make sense to let employees 'bank' bonus points for use in a later planning period; furthermore, long-term employees or the severely-disabled could receive more points than others. While bonus points define the priority between absence requests, an applicant may specify an additional *priority value* π, $0 \leq \pi \leq 1$, for each alternative of a first choice request, to weight them differently among each other. The accepted constraint then computes costs using the request's bonus point value β_{req}, in case of a complete rejection, and the respective priority π_{alt}, in case that an alternative is approved rather than the first choice.

In mathematical terms, at first the influence of the bonus points is given by

$$\gamma_{\text{bonus}} = c_1 \cdot \frac{\beta_{\text{req}}}{\beta_{\text{max}}} + c_2$$

and is used to compute the costs of the complete rejection: $\gamma_{\text{full}} = \gamma_{\text{bonus}} + c_3$. The costs of a rejected first choice whose k-th alternative is approved then result from the difference to the full rejection costs:

$$\gamma_{\text{alt}_k} = \gamma_{\text{full}} - (\pi_{\text{alt}} \cdot \gamma_{\text{bonus}} + c_4)$$

The constants $c_1, \ldots, c_4$ are used to control certain aspects of the cost computation. c_2 for example represents fixed minimum costs the rejection of an absence request has, independent from its bonus points. We used the following values: $c_1 = 30, c_2 = 2, c_3 = 2, c_4 = 1$.

A single time slot that is allocated to an absence request may match several alternatives at once, as shown earlier in figure 1. Accordingly, only the alternative which causes the lowest costs must be used as reference for the approval constraint. Let T_k be all the fitting time slots of alternative k; then the minimal costs of an allocated time slot (p_a, p_b) are

$$c_{(p_a,p_b)} = \min_{k|(p_a,p_b)\in T_k} \{\gamma_{\text{alt}_k}\}$$

and the approval constraint's final costs, c_{app}, for *a single absence request* with allocated time slots $\tilde{P}$:

$$c_{\text{app}} = \begin{cases} \gamma_{\text{full}} & \tilde{P} = \emptyset \\ \max_{(p_a,p_b)\in\tilde{P}}\{c_{(p_a,p_b)}\} & \text{else} \end{cases}$$

3.3 Quota Constraint

Additionally, the granted absence days of an employee, for instance for holiday leaves, must not sum up to more than a certain *quota* the employee can use. This could be avoided by preventing beforehand that employees apply for more days than they are entitled to, but this would also mean that they could only utilise the quota fully if no request was rejected. For this reason, we use the 'quota constraint' that, for every employee, checks each quota for under- and over-usage. An application example for lower and upper quota bounds is the German labour law that requests the usage of at least 12 out of 24 paid vacation days for recreational purposes, with 24 days being the minimal granted annual amount for full-time employees. As in the minimal manning constraint, the costs are higher for a single severe violation than for several light violations.

Since our period model, as described before, allows time slots with variable durations to grant the same absence request, and thus causing the same costs in the approval constraint, planning algorithms would tend to assign time slots with minimal duration as longer ones would cause more manning violations. At this point, the quota constraint is of great importance because it counteracts such tendencies since short time slots alone would most likely not sum up to the minimum quota usage.

Every quota has a validity period $v = (p_i, p_j), i \leq j$, and a minimum, $q_{\min} \in \mathbb{N}_0$, and maximum, $q_{\max} \in \mathbb{N}$, of days to use within this period (to simplify matters, we here only consider quotas whose validity periods do not overlap). Let T be *all* allocated time slots for a certain employee, not only those belonging to a single request. The number of days these time slots overlap the quota's validity period is thus given by:

$$u = \sum_{(p_a, p_b) \in T} (\max\{0, \min\{b, j\} - \max\{a, i\}\} + 1)$$

and the quota constraint's costs for *a single employee* and *a single quota type* by:

$$c_{\text{quota}} = \left(\max\left\{0, c_1 \cdot \frac{q_{\min} - u}{q_{\min}}\right\}\right)^{c_3} + \left(\max\left\{0, c_2 \cdot \frac{u - q_{\max}}{q_{\max}}\right\}\right)^{c_4}$$

We used $c_1 = 20, c_2 = 20, c_3 = 2, c_4 = 2$.

3.4 Time Clash Constraint

Finally, it is not only sufficient to allocate fitting time slots (or none) for each absence request period to construct a feasible solution, but it is also required to avoid overlaps between the granted absence periods of one employee. We postulate that feasible problem input data has no possible intersections between first choice periods. However, request alternatives are allowed to possibly intersect each other. The 'time clash constraint' produces relatively high costs for any such overlap to guide solutions to a feasible region without prohibiting them altogether to visit 'clash regions' as intermediate results.

4 Move Pools

To visit the neighbourhood of a certain planning solution, we implemented two different sets of possible next moves: a 'naïve' move pool which just randomly changes time slot assignments and an 'advanced' pool that additionally takes the current constraint violations

into account. The naïve pool contains the moves necessary to explore the whole search space: a rejection move to deallocate all time slots from an absence request and a change move to make a new random time slot assignment. It also contains a move allocating first choice time slots and a move that only assigns time slots to a rejected request, i.e. presently without any time slots. Those four moves each operate on a randomly chosen target request and are all executed with equal probability.

The second, advanced move pool additionally works with *repair moves*. A repair move tries to mitigate a specific conflict in the current planning solution. Since conflicts are defined by the corresponding constraints, each constraint has its repair move. It would be unwise to try and find the correction which solves a local conflict best as a specific constraint's repair move is independent from the other constraints: a minimal manning conflict, for example, is locally solved best by rejecting all involved requests which directly leads to high costs in the approval constraint. For this reason, our repair move implementations randomly try to change affected requests and return the first new assignment that causes less violations in the constraint being repaired. In addition to the repair moves, the advanced move pool also includes the basic reject and change moves also used by the naïve pool. A repair move is chosen over one of the two basic moves with a rate of 2 : 1.

5 The Algorithms

Our first focus lay on getting a general idea of the challenges in absence optimisation and the influence of the two different move pools. Consequently, we at first used well-established metaheuristics which we adapted to our specific problem only to a small extent as to capitalise on the performance of the basic approaches. The two metaheuristics we employed were Tabu Search (TS) [2] and Simulated Annealing (SA) [5], which both are not population-based and strongly influenced by their neighbourhood definition given by the move pool.

Our implementation of TS follows basic design paradigms described, for example, in [3]. The *tabu list length* is equal to the problem dimension, that is, the total number of first choice requests; it is kept fixed throughout the optimisation. For our problem model, the full neighbourhood of a current solution would consist of all other solutions visitable by a single move. Since the size of this neighbourhood is too large for an efficient TS implementation, we artificially limited the *neighbourhood size* by choosing a fixed number of random moves as candidates representative for the complete neighbourhood. A *tabu list entry* is generated by computing the delta of newly allocated/deallocated absence days from the last executed move. An exemplary computation is shown in figure 2: Only those days that have not been allocated before are prohibited to be deallocated and those that have newly been deallocated are prohibited to be allocated again. We did not incorporate any major changes to the stan-

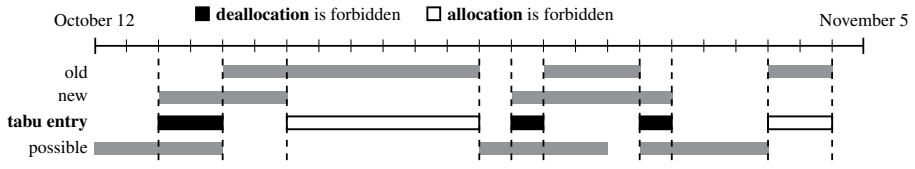


Fig. 2 Example of tabu list entry generation. New and old time slots are compared and the tabu list is filled accordingly. An assignment that would still be allowed afterwards is given as 'possible'.

dard algorithm design [1] in the SA either. The only deviation consists of an adaptive cooling scheme which pays attention to a predefined maximum iteration count and thus incorporates slower cooling for more iterations and vice versa.

We carried out preliminary experiments to determine good parameters for our algorithms and ended up with a cooling factor of 0.95 for SA and 20 as the neighbourhood size for TS; these values delivered the best *expected* – as we are dealing with randomised algorithms and initialisation – solution quality.

6 Results

Due to data privacy protection issues, real-world leave application data unfortunately was not accessible for testing at the time of writing this paper. We therefore designed a randomised test data generator according to typical real-world conditions:

- The planning period spans over a single calendar year.
- Employees have a quota of 30 days for holiday leave, of which 14 days should be granted at least.
- The probability that an employee applies for a leave is higher on bridge days and during (German) school holidays.
- There is a group hierarchy with, for a total of n employees, $n/10$ 'base groups' and a probability of $10/n$ that a certain employee is in a certain base group. There are also k additional groups with probability 0.3^k for an employee to be member of additional group k. This can be interpreted as a hierarchy of superiors with three superiors per base group on the first level, 0.3 of those on the second, and so forth.
- The minimal manning requirement for all groups is increased around mid-year.
- Employees are randomly assigned to groups according to the respective group membership probability but are member in at least one group. Paired with the hierarchical group model, the absence or presence of an employee has a high probability to influence the headcount of more than one group at once.
- Employees use an average of 95% of their leave quota for absence requests.

Exemplary test data generated thus with 100 employees had more than 10^{343} potential solutions[1], other data with 500 employees had 10^{1951} solutions. In figure 3 an example of a generated absence request distribution and the corresponding minimal manning requirements is shown (for the sake of simplicity, all employees are in one single base group only). It is impossible in our test setup to find a solution with no constraint violations, because on several days more first choice requests overlap than would be allowed to fulfil the minimal group manning, and the rejection of a first choice always causes costs > 0.

All optimisation runs were executed on a machine with an Intel® Core™ i5-3570K processor (4 cores, maximum clock rate 3.5GHz) under a 64-bit Ubuntu Linux version 14.04. Solutions were initialised randomly, i.e. by assigning random feasible time slots to each absence request or rejecting it. The maximum optimisation time was chosen arbitrarily only with respect to providing TS with a sufficiently generous time to converge.

In table 1 the averaged results of ten optimisation runs for problem sizes of 100 and 250 employees are shown. It is interesting to note that the advanced move pool could only slightly improve some results and even caused significantly worse final costs in some cases. Figure 4 shows the costs of the best solution found so far over time for single representative

[1] approx. 700 first choice requests with an average of one alternative each, slightly more than one period per request, and about 5 possible time slots per first choice/alternative

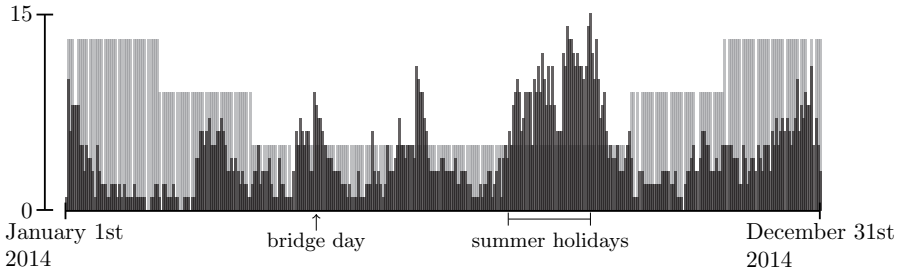


Fig. 3 Plot of generated test data with 50 employees. For each day of the planning period the accumulated number of overlapping first choice absence requests (dark grey) and the maximum allowed number of absent employees (light grey) are shown. Gaps in the latter result from days where the minimal manning is 0 (Sundays and public holidays).

		naïve move pool		advanced move pool	
		simulated annealing	**tabu search**	**simulated annealing**	**tabu search**
100	**costs at start**	5254.96	5343.54	5425.37	5268.26
	final costs	522.9	575.9	529.57	540.84
	time [s]	440	440	430	430
250	**costs at start**	13333.84	13258.21	12921.56	13273.32
	final costs	1067.56	1197.44	1074.68	1118.5
	time [s]	1034	1034	1101	1101

Table 1 Averaged minimal final costs after ten test runs found by TS and SA for randomly generated problems with 100 and 250 employees.

algorithm executions. Although for the 100 employees problem, the advanced move pool caused SA to find better solutions faster within the first half minute of execution, it in fact led to a slower convergence speed that did not even result in lower costs. This may be due to the fact that the computation of repair moves is more complex and takes thus more time such that, on the one hand, more algorithm iterations with the naïve move pool can be executed in the same time compared to using the advanced pool. On the other hand, the solely random moves do not reduce the search space by concentrating on specific current constraint conflicts and thereby allow more different solutions to be visited and possibly better ones to be found. However, the advanced move pool did improve SA's optimisation process for the larger – 250 employees – problem. It seemed like the higher problem size did relativise the computational overhead of the advanced move pool such that the additional effort proved to be worthwhile in the end.

Besides the influences of the two move pools, the final results of SA were a little better than those of TS while the latter reached good solutions a lot earlier.

7 Conclusion and Future Work

We have shown that the optimisation of staff absences, at least the way we modelled it, is no easy task: the ASP has a considerably large search domain, constraints with high interrelation among each other, and diverse factors that cause high variability in the input data, like school and public holidays or different general workload throughout the planning period. It remains to be determined whether, and to what extent, the modelling possibilities,

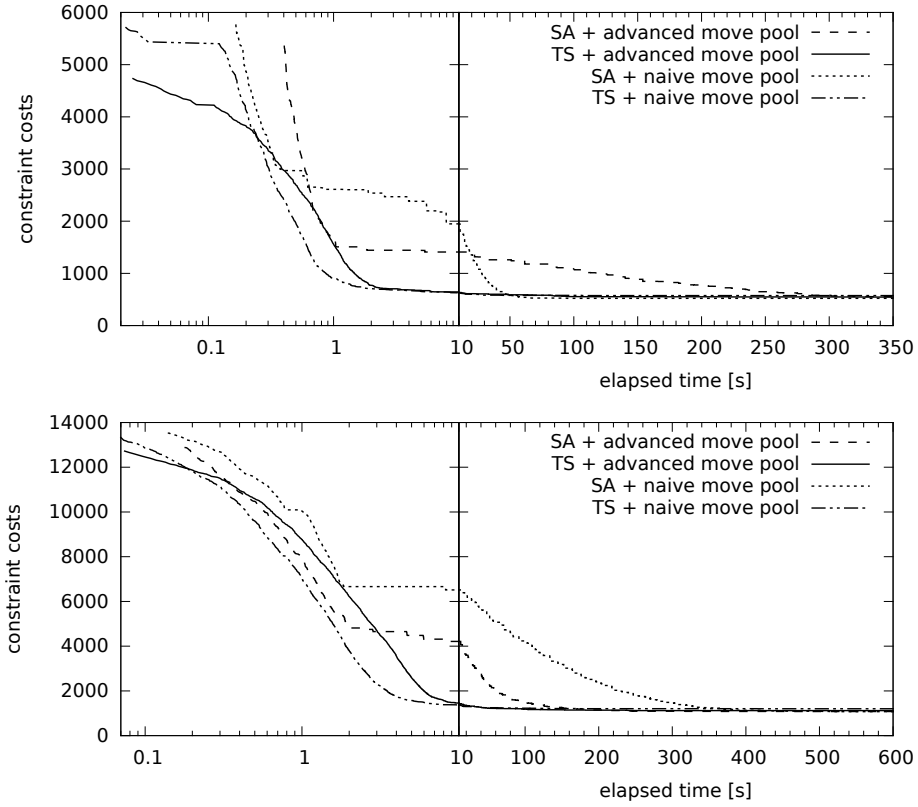


Fig. 4 Exemplary optimisation runs on the 100 employee (top) and 250 employee (bottom) problem. The different starting costs are the result of randomised initialisation and first optimisation step of the respective algorithm.

for instance multiple periods per request, will be incorporated in real-world applications or whether they cause negative feedback by overcomplicating the leave application process.

Additionally, we demonstrated how the 'usual suspects' TS and SA can improve a randomly initialised solution by a great deal, but that special attention has to be paid on the possible negative effects of more sophisticated move pools as is strongly suggested by our simulation runs.

Our next steps will include optimisation runs on anonymised real-world data and closer examination of the acceptance of the generated vacation schedules, both by the applicants and those responsible for planning.

References

1. E. Aarts, J. Korst, and P. Laarhoven. Simulated Annealing. In E. Aarts and J. K. Lenstra, editors, *Local Search and Combinatorial Optimization*, pages 91–120. John Wiley & Sons Ltd., 1997.
2. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
3. A. Hertz, E. Taillard, and D. de Werra. A Tutorial on Tabu Search. In *Proceedings of Giornate di Lavoro AIRO*, volume 95, pages 13–24, 1995.

4. R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104, New York, 1972. Plenum Press.
5. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.
6. J. Ostler and P. Wilke. The Erlangen Advanced Timetabling System (EATTS) Unified XML File Format for the Specification of Timetabling Systems. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, pages 447–464. Patat2010 – Queen's University Belfast, 2010.

MISTA 2015

Solving Huge Real-World Timetabling Instances

Gerald Lach · Mirjana Lach · Erhard Zorn

Abstract In this paper, we present a new IP-based model for the university course timetabling problem. It has been applied successfully to huge real-world instances. We have been able to generate timetables at RWTH Aachen University and Technische Universität Berlin—two of the largest technical universities in Germany—with more than 30,000–40,000 students, serious room limitations, and several additional constraints.

1 Introduction

Before the start of a new semester, universities are confronted with the problem of coordinating courses, rooms, lecturers, and student groups in such a way that the generated course timetable satisfies a multitude of needs. This problem is called the *university course timetabling problem (UCT)*—and known to be NP-complete. Owing to the complexity of the problem as well as many organizational issues, most universities do not generate automated timetables. Due to the lack of adequate tools, the majority of German universities do not calculate suitable timetables. As a result, the old course timetables are maintained year after year. This way of coordinating the resources often leads to serious problems: Changes in the courses of study—for example, changes due to the *Bologna process*—result in new challenging demands on the timetable which are often impossible to satisfy by just manually adapting the old one. Furthermore, the hoarding of room capacity by lecturers causes an enormous waste of room resources and results in costs—which are avoidable.

Gerald Lach
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: lach@math.tu-berlin.de

Mirjana Lach
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: mlach@math.tu-berlin.de

Erhard Zorn
Technische Universität Berlin, Institute of Mathematics/innoCampus
E-mail: erhard@math.tu-berlin.de

University course timetabling remains a challenging issue. Especially large universities with more than 30,000 students often get overstrained owing to the creation of timetables that satisfy all requirements. Due to the high complexity of the problem, manually creating timetables seems impossible. On the other hand, although the university course timetabling problem has been of great interest over the past few years, there has been little research on solution methods that focus on such huge problems. With our work, we take a first step to close this gap, and present a solution to this problem. So far, our method has been successfully implemented at two of the largest technical universities in Germany: RWTH Aachen University and Technische Universität Berlin.

1.1 History

In 2003, a team of innoCampus, a department of TU Berlin, started research on new ways to deal with university timetabling problems. Over the decade, the IT system *MosesKonto* was developed as a technical basis, consisting of a variety of components: IP solver, database, graphical user interface, and web services. After having successfully introduced new solution methods for the *post-enrollment timetabling problem* in 2003 ([5],[4]) and the *examination timetabling problem* in 2010 at TU Berlin, the innoCampus team and the department of operations research at RWTH Aachen University were contacted by the administration of RWTH Aachen University. Owing to an increasing number of first-year students in 2013 and a delay in constructing new lecture halls, RWTH Aachen University had to improve the coordination of their resource 'room'. In 2012, innoCampus and the department of operation research started the *carpe diem project*. The aim of the project was to extend the MosesKonto by a tool—including an IP solver—that is capable to automatically generate a conflict-free course timetable for all courses taught at RWTH Aachen University—a university with more than 42,000 students, 1,200 lecturers, and 500 rooms. Furthermore, the generated timetable had to be accepted by all users. After one year of researching, implementing and collecting data, the first automatically generated timetable for fall semester 2013 was published in June 2013. One and a half years later, TU Berlin also stopped copying the old course timetables and started to create them automatically with the MosesKonto.

1.2 Our contribution

In the last 10 years, scheduling in general and timetabling in particular have been a point of interest in research. The conference series *Practice and Theory of Automated Timetabling (PATAT)* only addresses timetabling problems. Every four years, as part of the *International Timetabling Competition (ITC)*, best solution methods for standardized problem formulations are looked for. In all the problem instances considered in the ITC, the problem size had been significantly smaller than the problem sizes we had to deal with at RWTH Aachen University or TU Berlin. While in ITC 2007 and 2011 no instance with more than 400 courses and 50 lecturers had been included in the test sets, our solution methods had to solve approximately 5–8 times larger instances.

In this paper, we introduce our approach for solving huge university course timetabling problems. The problem is decomposed into two parts; depending on their type, courses are scheduled in the first step or in a subsequent second step. If a course consists

of a multitude of lectures, and every attending student has only to participate in one of them, we categorize the course as a 'tutorial' and consider it in the second step, otherwise in the first one. Though the problems of the first and the second step appear to be similar, there is one difference that impacts the model significantly. Although the definition of a 'conflict' between non-tutorial courses is straightforward, this could not be applied to tutorials. This difference results in different problem formulations that are solved consecutively. Both steps are solved using integer programming (IP) techniques.

We tested our solution method on various real-world problem instances from RWTH Aachen University and TU Berlin, and were able to find good feasible solutions for all instances in reasonable running times. Differences in the structure of the universities lead to a discrepancy in the quality of the solutions. While it generally took one day to find a solution with an optimality gap of 15 % for the problem instances of RWTH Aachen University, problems of TU Berlin could be solved nearly optimally in less than two hours. To the best of our knowledge, we are the first to solve such huge timetabling problems, and whose methods have been successfully implemented at universities.

1.3 Related work

Over the past few years, university course timetabling problems have received a lot attention. Many heuristic (e.g. [10], [11]) or logic programming ([1]) solution methods have been published. In the last six years, integer programming techniques ([2], [8]) have also been used. Although we do not use a two-stage decomposition in *firstly assigning timeslots and secondly rooms*, the presented model is based on the work of Lach and Lübbecke [7]. Due to the requirements of RWTH Aachen University and TU Berlin—mainly a variety of different spans of the course timeslots—the model described in [7] was not applicable. Other time-room decomposition approaches as discussed in [12] are also not suitable. Due to tight room resources, we did not consider neglecting the room resources in the first step.

As mentioned before, the majority of the published solution methods for the university course timetabling problem focus on solving complex, combinatorial but small problem instances. One project *UniTime* also focuses on solving larger timetabling instances. In [10], a detailed explanation of an approach for solving timetabling instances for up to 7,000 students is given. In contrast to our approach, here heuristics methods, as opposed to integer programming techniques, are used.

2 Problem formulation

In the following sections, we introduce our IP formulation for both steps of the decomposition model. For ease of exposition, we do not introduce a bunch of soft constraints. The soft constraints were mainly added to the IP formulation in order to be able to model deductive requirements stipulated by the lecturers. Neglecting these would enormously decrease the user acceptance and result in a failure of the project. A detailed research on the soft constraints will be presented in an upcoming article.

We denote $C = C^T \cup C^N$ to be the set of all courses, and L to be the set of all lectures. $C^T \subset C$ is the set of all courses c which are of the type *tutorial*, and $C^N \subset C$ the set of all *non-tutorial courses*. Each course c comprises multiple lectures

$L_c \subset L$, and for every lecture l, we denote $l_c \in C$ the course belonging to l. The set of all lectures belonging to a tutorial course is denoted by L^{C^N}, and the set of lectures belonging to a non-tutorial course by L^{C^T}. There is a structural difference between a course $c^T \in C^T$ and a course $c^N \in C^N$: Every student who participates in course c^N has to attend all lectures $l \in L_{c^N}$, whereas a student participating in the course c^T has to attend only one lecture $l \in L_{c^T}$.

Furthermore, let D be the set of the days of a week, H be a set of pairwise disjoint times of day intervals and $T = \{(d, h) : d \in D, h \in H\}$ the set of all possible timeslots. Assuming for all $h_1, h_2 \in H$ to be disjoint, consequently all $t_1, t_2 \in T$ are disjoint. Most of the problems known in the literature—e.g. [9] or [8]—assume that all lectures are of the same duration. We found that this formulation is not sufficient to model the situation at German universities. Therefore, we introduced the following set of periods:

$$P = \{(d, \bigcup_{i=1}^{m} h_i) : d \in D, 1 \leq n \leq |H|, h_i, h_{i+1} \text{ are consecutive}\} \tag{1}$$

For every lecture l, we denote the set of all eligible periods $P_l \subset P$, and for every course c, we denote all eligible periods $P_c = \bigcup_{l \in L_c} P_l$.

For ease of presentation, we define a timeslot $t = (d^t, h^t) \in T$ to be an element of a period $p = (d^p, \bigcup_{i=1}^{n} h_i^p)$ $(t \in p)$ if $d^t = d^p$ and $h^t \in \bigcup_{i=1}^{n} h_i^p$.

Moreover, we extended the commonly used model of the resource room in order to fit needs of the real world. For a non-negligible number of courses, we had to offer the ability to be held in multiple rooms at the same time—some lectures have to be live-streamed in another room, or laboratories are to be held simultaneously. Therefore, for the set of rooms R, we define the set of room groups in the following manner:

$$RG = \{rg \in 2^R : \exists l \in L \text{ such that } l \text{ is to be held in all } r \in rg \text{ at once}\} \subset 2^R \tag{2}$$

With $RG_r \subset RG$, we denote all room groups which room r is a part of, with $RG_l \subset RG$ all for lecture l appropriate room groups and $T_r \subset T$ all timeslots for which room r is available. Based on the availability of the rooms, we deduce the (period) availability of a room group $P_{rg} \subset P$ as follows:

$$P_{rg} = \{p \in P : \forall r \in rg \forall t \in p \;\; t \in T_r\} \tag{3}$$

Time conflicts are represented by two conflict graphs $G_{\text{conf}} = (V_{\text{conf}}, E_{\text{conf}})$ and $\hat{G}_{\text{conf}} = (\hat{V}_{\text{conf}}, \hat{E}_{\text{conf}})$. While $G_{\text{conf}} = (V_{\text{conf}}, E_{\text{conf}})$ is used to model the time conflicts to be considered in the first decomposition step, $\hat{G}_{\text{conf}} = (\hat{V}_{\text{conf}}, \hat{E}_{\text{conf}})$ is suitable for the time conflicts in the second step. Each node in $V_{\text{conf}} = \{(l, p) : l \in L \wedge p \in P_l \wedge l_c \in C^N\}$ and $\hat{V}_{\text{conf}} = \{(l, p) : l \in L \wedge p \in P_l \wedge l_c \in C^T\}$ represents a possible period p where lecture l should take place. Two nodes $v_1 = (l_1, p_1), v_2 = (l_2, p_2)$ are adjacent if l_1 held at p_1 prohibits l_2 to be held at p_2. There can be a number of reason for adding a conflict to G_{conf}, for example some student groups may be compelled to take l_1 and l_2 at the same time, or l_1 and l_2 have the same instructor, or for all student groups at least one lunch slot per day should be free. $\hat{G}_{\text{conf}}$ has less edges. At most, an edge $v_1 = (l_1, p_1), v_2 = (l_2, p_2)$ is added to the graph if $p_1 = p_2$ and l_1 and l_2 are both instructed by the same lecturer.

2.1 Integer program of the first step

In the first step of our decomposition approach, we disregard all lectures $l \in L^{C^T}$. In our opinion—due to the ability of the students to opt for just one of the many possible lectures, the conflicts for all lectures l belonging to a tutorial are soft and can be coordinated in a secondary way—this approach is suitable.

Before creating the model based on the first part, we preprocess and calculate an inclusion minimal clique cover $U \subset 2^{V_{\text{conf}}}$ of G_{conf} using the algorithm introduced by Kou, Stockmeyer, and Wong introduced in [6].

The IP formulation is a modified version of the well-known tree-index formulation presented in [7]. With the modification, the model turned out to be suitable to solve huge timetabling instances. For all $v = (l, p) \in V_{\text{conf}}$, we introduce a binary variable $x_{l,p}$ which is set to one if lecture l takes place at period p. Furthermore, we define

$$LRGP = \{(l, p, rg) : (l, p) \in V_{\text{conf}} \wedge p \in P_{rg} \wedge rg \in RG_l\} \tag{4}$$

to be the set of all feasible lecture-period-room-group assignments. For each element $(l, rg, p) \in LRGP$, we define a variable $y_{l,rg,p}$ which is set to one if lecture l is scheduled in room group rg at period p, otherwise it is zero. Adding an objective coefficient $u_{l,rg,p}$ to $y_{l,rg,p}$ representing time respectively room preferences of the lecturer and dummy variable d_l, we formulate the model as follows:

$$\min \sum_{(l,rg,p)\in LRGP} u_{l,rg,p} \cdot y_{l,rg,p} + \sum_{l \in L^{C^N}} 1,000 \cdot d_l \tag{5}$$

$$s.t. \sum_{p \in P_l} x_{l,p} + d_l = 1 \qquad \forall l \in L^{C^N} \tag{6}$$

$$\sum_{(l,p)\in U_i} x_{l,p} \leq 1 \qquad \forall U_i \in U \tag{7}$$

$$x_{l,p} - \sum_{(l,rg,p)\in LRGP} y_{l,p,rg} = 0 \qquad \forall (l, p) \in V_{\text{conf}} \tag{8}$$

$$\sum_{rg\in RG_r, t\in p, (l,rg,p)\in LRGP} y_{l,rg,p} \leq 1 \qquad \forall r \in R, t \in T \tag{9}$$

$$x_{l,p} \in \{0, 1\} \qquad \forall (l, p) \in V_{\text{conf}} \tag{10}$$

$$y_{l,rg,p} \in \{0, 1\} \qquad \forall (l, rg, p) \in LRGP \tag{11}$$

Constraint (6) guarantees that every lecture is assigned a period. (7) guarantees a conflict-free coordination of the lectures. In (8), it is ensured that every lecture takes place in an appropriate room group and finally in (9) it is ensured that no room is booked twice at the same time. For a clear arrangement of the formulation, we skip some specific needs of TU Berlin and RWTH Aachen University which could be easily integrated into the model. For example, there are some lectures that do not need any room to be assigned to but have to be considered in the formulation in order to ensure conflict-free coordination of the lectures.

2.2 Integer program for the second step

The basic idea in the second step of our decomposition approach is to simulate a *post-enrollment course timetabling problem* by taking into account the created timetable of the first decomposition step—for a detailed description of the *post-enrollment course timetabling problem*, see [3]. We base our formulation on the availability of the study plans of all programs of study at the university and define a *student group* as a set of students of a specific term pursuing a particular program of study. By denoting SG the set of all student groups, we define $C_{sg} \subset C$ as the set of all courses the student group sg should take based on their study plan and $SG_c \subset SG$ the set of student groups participating in course c. In addition, we define the *average lecture capacity* (alc) of a $c \in C^T$:

$$\text{alc}(c) = \frac{\sum_{sg \in SG_c} |sg|}{|L_c|} \tag{12}$$

In order to find a globally feasible solution, we have to consider the solutions of the first part in the second part. Especially it is important to block slots for student groups that have been assigned compulsory courses. Assuming $U_{sg} \subset C$ to be the set of compulsory courses of student group sg, we define remaining student group timeslots (RSGT) as follows:

$$RSGT = \{(sg,t) : \forall c \in C^N \cap U_{sg} \ \nexists l \in L_c \text{ such that } t \in p \text{ and } x_{l,p} = 1 \ \} \tag{13}$$

Based on the remaining student group timeslots, we claim the set remaining student group periods (RSGP) as follows:

$$RSGP = \{(sg,p) : sg \in SG, \ \ p \in P \ \wedge \ \nexists t \in p \text{ such that } (sg,t) \notin RSGT\} \tag{14}$$

The set student group course periods (SGCP) represents all periods P when members of a student group sg are able to attend a lecture l of the (tutorial) course c:

$$SGCP = \{(sg,c,p) : sg \in SG \ \ c \in C_{sg} \ \ p \in P \ \text{ such that } (sg,p) \in RSGP \ \ \wedge p \in P_c\} \tag{15}$$

To avoid overbooking, we make allowance for the room occupation plan calculated in the first step and define $\hat{P}_{rg}$:

$$\hat{P}_{rg} = P_{rg} \backslash \{p \in P : \exists l \in L \ \ \exists r \in rg \ \ \exists \hat{rg} \in RG_r \ \ \exists t \in p \ \ \exists \hat{p} \in P \ \ t \in \hat{p} \wedge y_{l,\hat{rg},\hat{p}} = 1\} \tag{16}$$

Based on $\hat{P}_{rg}$, we set the lecture-period-room-group-assignment $\widehat{LRGP}$ of the second step as follows:

$$\widehat{LRGP} = \{(l,rg,p) : (l,p) \in \hat{V}_{\text{conf}} \wedge p \in \hat{P}_{rg} \wedge rg \in RG_l\} \tag{17}$$

Similar to the model of the first part, every $v \in \hat{V}_{\text{conf}}$ is represented by a binary variable $\hat{x}_{l,p}$ set to one if lecture l takes place at period P and every $(l,rg,p) \in \widehat{LRGP}$ by another binary variable $\hat{y}_{l,rg,p}$ set to one if lecture l takes place at period P in room group rg. Additionally, for every triple $(sg,c,p) \in SGCP$, we introduce an integer

variable z, which implies the number of member sg participating at period p in a lecturer of course c. Finally, adding dummy variables $d_{sg,c}$, which are penalizing if a student group member is not able to visit any lecture of a tutorial course he should have to, we get the following integer program:

$$\min \sum_{(l,rg,p)\in L\widehat{RG}P} u_{l,rg,p}\cdot \hat{y}_{l,rg,p} + \sum_{sg\in SG}\sum_{c\in C_{sg}} 10\cdot d_{sg,c} + \sum_{l\in L^{C^T}} 1,000\cdot d_l \tag{18}$$

$$s.t. \sum_{p\in P_l} x_{l,p} + d_l = 1 \qquad \forall l \in L^{C^T} \tag{19}$$

$$\sum_{(sg,c,p)\in SGCP} z_{sg,c,p} + d_{sg,c} = |sg| \qquad \forall sg \in SG, \forall c \in C_{sg} \tag{20}$$

$$\sum_{(sg,c,p)\in SGCP, t\in p} z_{sg,c,p} \leq |sg| \qquad \forall sg \in SG, \forall t \in T \tag{21}$$

$$\sum_{(sg,c,p)\in SGCP} z_{sg,c,p} - \sum_{l\in L_c} \mathrm{alc}(c)\cdot \hat{x}_{l,p} \leq 0 \qquad \forall c \in C^T, \forall p \in P_c \tag{22}$$

$$\hat{x}_{l_1,p_1} + \hat{x}_{l_2,p_2} \leq 1 \qquad \forall((l_1,p_1),(l_2,p_2)) \in \hat{E}_{\mathrm{conf}} \tag{23}$$

$$\hat{x}_{l,p} - \sum_{(l,rg,p)\in L\widehat{RG}P} \hat{y}_{l,rg,p} = 0 \qquad \forall (l,p) \in \hat{V}_{\mathrm{conf}} \tag{24}$$

$$\sum_{rg\in RG_r, t\in p, (l,rg,p)\in L\widehat{RG}P} \hat{y}_{l,rg,p} \leq 1 \qquad \forall r \in R, t \in T \tag{25}$$

$$\hat{x}_{l,p} \in \{0,1\} \qquad \forall (l,p) \in \hat{V}_{\mathrm{conf}} \tag{26}$$

$$\hat{y}_{l,rg,p} \in \{0,1\} \qquad \forall (l,rg,p) \in L\widehat{RG}P \tag{27}$$

$$z_{sg,c,p} \in \mathrm{N} \qquad \forall (sg,c,p) \in SGCP \tag{28}$$

Constraint (20) and (21) ensure every student group member is assigned to the lectures of all (tutorial) courses without conflict. Constraint (22) guarantees that for all possible periods the capacity of the lectures is sufficient for the assigned students. All remaining constraints can be easily deduced from the integer program (5) – (11).

3 Results

Our model has been successfully applied to problem instances at TU Berlin and RWTH Aachen University. Both universities are among the largest in Germany, and for both we were able to find feasible and nearly optimum solutions in reasonable running time. In Table 3, basic conditions of TU Berlin and RWTH Aachen University are presented.

Some problem characteristics turned out have an enormous influence on the performance of the solver. In particular, the size of the set H significantly affected the running time. The finer the discretization of the set *periods-per-day*, the purer the performance of the solver. Therefore, two different definitions of H were used in our test set: $H_1 = \{08-10, 10-12, 12-13, 13-14, 14-15, 15-16, 16-18, 18-20\}$ and $H_2 = \{08-10, 10-12, 12-14, 14-16, 16-18, 18-20\}$. Moreover, the number of

Table 1 Problem statistics

Uni	Semester	$\|C\|$	$\|C^N\|$	$\|C^T\|$	$\|L\|$	$\|SG\|$	U	$\|R\|$
RWTH	WS 13/14	2,354	2,312	42	2,915	1,345	1,448	192
RWTH	SS 14	2,540	2,487	53	3,601	1,114	1,480	214
RWTH	WS 14/15	2,932	2,865	67	3,892	1,382	1,527	227
RWTH	SS 15	3,179	2,677	57	3,702	1,135	1,516	276
TUB	SS 15	1,970	1,884	86	3,773	704	860	337

conflicts—the number of edges in G_{conf}—heavily influenced the running time of the solver. For ease of exposition in Tables 2 and 3, we denote with conf $\subset C \times C$ all conflicting course tuples, which means if $(c_1, c_2) \in$ conf all lectures of c_1 and c_2 should not be scheduled at the same time. Table 2 shows the running time statistics for the first step of the decomposition model of two representative problem instances, while Table 3 shows the running time statistics for the second step of the decomposition model of the two problem instances.

Table 2 Problem statistics for the first step

Instance	H	|conf|	Relaxation	Root-Node	Gap ≤ 20
RWTH WS 14/15	H_1	56,647	5,874 sec.	9,770 sec	9,000 sec.
RWTH WS 14/15	H_1	25,401	1,380 sec.	21,562 sec.	17,871 sec.
RWTH WS 14/15	H_2	56,647	3,583 sec.	8,520 sec.	8,126 sec.
RWTH WS 14/15	H_2	25,401	380 sec.	5,064 sec.	2,247 sec.
TUB SS 15	H_1	16,170	90 sec.	500 sec.	752 sec.
TUB SS 15	H_1	6,467	23 sec.	130 sec.	320 sec.
TUB SS 15	H_2	16,170	25 sec.	209 sec.	258 sec.
TUB SS 15	H_2	6,467	10 sec.	100 sec.	123 sec.

Table 3 Problem statistics for the second step

Instance	H	|conf|	Relaxation	Root-Node	Gap ≤ 15
RWTH WS 14/15	H_1	10	8 sec.	50 sec.	30 sec.
RWTH WS 14/15	H_1	10	64 sec.	170 sec.	170 sec.
RWTH WS 14/15	H_2	10	12 sec.	12 sec.	135 sec.
RWTH WS 14/15	H_2	10	30 sec.	265 sec.	879 sec.
TUB SS 15	H_1	0	140 sec.	3,304 sec.	3,304 sec.
TUB SS 15	H_1	0	50 sec.	750 sec.	460 sec.
TUB SS 15	H_2	0	132 sec.	1,894 sec.	1,894 sec.
TUB SS 15	H_2	0	71 sec.	1,000 sec.	3,200 sec.

4 Outlook

It was a great success for us to be able to to solve such huge timetabling instances and implementing our software at RWTH Aachen University and TU Berlin. Our future

research will focus on ways to improve the quality of the timetable. One main issue will concern robust timetabling algorithms. Taking into account the old timetable and the new basic conditions to come up with a new feasible timetable that largely matches the old one seems to be an interesting point of view.

References

1. Banbara, M., Soh, T., Tamura, N., Inoue, K., Schaub, T.: Answer Set Programming as a Modeling Language for Course Timetabling. Theory and Practice of Logic Programming (2013)
2. Burke, E.K., Mareek, J., Parkes, A.J., Rudov, H.: Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research **37**(3), 582–597 (2010)
3. Ceschia, S., Di Gaspero, L., Schaerf, A.: Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Computers & Operations Research **39**(7), 1615–1624 (2012)
4. Gora, W., Jeschke, S., Lach, G., Lübbe, J., Pfeiffer, O., Zorn, E.: Management and optimal distribution of large student numbers. In: Proceedings of the Education Engineering 2010, pp. 1891–1896 (2010)
5. Jeschke, S., Luce, R., Pfeiffer, O., Zorn, E.: Study Management and Allocation of Exercise Classes for Large Lectures at TU Berlin. In: Proceedings of the CISSE 2007, pp. 235–248. Springer, Berlin (2008)
6. Kou, L.T., Stockmeyer, L.J., Wong, C.K.: Covering edges by cliques with regard to keyword conflicts and intersection graphs. Commun. ACM **21**(2), 135–139 (1978)
7. Lach, G., Lübbecke, M.E.: Optimal university course timetables and the partial transversal polytope. In: C. McGeoch (ed.) 7th International Workshop on Efficient and Experimental Algorithms (WEA08), *LNCS*, vol. 5038, pp. 235–248. Springer, Berlin (2008)
8. Lach, G., Lübbecke, M.E.: Curriculum based course timetabling: new solutions to Udine benchmark instances. Annals of Operations Research **194**(1), 255–272 (2012)
9. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing **22**(1), 120–130 (2010)
10. Müller, T., Rudov, H.: Real-life curriculum-based timetabling with elective courses and course sections. Annals of Operations Research pp. 1–18 (2014)
11. Murray, K., Schluttenhofer, S.: University course timetabling & student sectioning system
12. Phillips, A.E., Waterer, H., Ehrgott, M., Ryan, D.M.: Integer programming methods for large-scale practical classroom assignment problems. Computers & OR **53**, 42–53 (2015)

MISTA 2015

Scheduling a conference to minimize attendee preference conflicts

Jeffrey Quesnelle · **Daniel Steffy**

Abstract This paper describes a conference scheduling (or timetabling) problem where, at the time of registration, participants indicate preferences for events within the conference that they would like to attend. Based upon these preferences, an assignment of events to rooms and time slots should be determined that minimizes the number of attendee preference conflicts and satisfies a number of hard constraints. Ideally the schedule should be constructed so that for most, or all, participants the events that they would like to attend are assigned to different time slots. We show that our problem, and several variants of it, are NP-hard. An integer programming model is developed to solve the problem and a computational study of this model is performed on instances generated from real data. Improvements to the model, including a symmetry breaking reformulation and a dualization of some hard constraints, are shown to significantly improve solution times, making the problem tractable for the desired real world application.

1 Introduction

This project was motivated by the problem of scheduling events within PenguiCon [5], a conference organized by the open-source community in Michigan. The conference typically includes approximately 250 events such as lectures, demonstrations and panel discussions, all of which must be scheduled into rooms and time slots. Many of the events involve multiple presenters/panelists, and many presenters participate in more than one event; it is a hard constraint that no speaker can be multi-booked during a given time period. Furthermore, the registration website will give conference participants the ability to indicate preferences for events before the schedule is generated, giving the extra complication of trying to generate the schedule based on these responses that minimizes participant schedule conflicts. Our problem is related to previously studied conference and class scheduling problems but

Jeffrey Quesnelle
University of Michigan-Dearborn
E-mail: jfquesne@umich.edu
This paper was written while the author was an undergraduate at Oakland University

Daniel Steffy
Oakland University
E-mail: steffy@oakland.edu

includes what we believe is a novel and difficult combination of constraints and objectives. The goal of this paper is to model our problem, relate it to previous work, and implement solution methods that can be deployed to schedule the PenguiCon conference in practice.

In Section 2 we discuss some related work. In Section 3 we formally describe our problem and relate it to previous work, we also show that our scheduling problem, and some variants, are all NP-hard. Section 4 describes an integer programming model for the problem. Various improvements to the model are also described and evaluated computationally. Section 5 provides concluding remarks.

2 Background and related work

One of the oldest scheduling problems that has been studied is the *timetable design problem* (TTD). Given a set of time slots, a set of teachers and their available teaching hours, and a matrix describing which courses each teacher is required to teach, the TTD problem is the problem of determining if there exists a schedule that satisfies the constraints. TTD was shown to be NP-Complete in 1976 via reduction from 3-SAT [1]. However it is notable that certain variants of the TTD problem are known to be polynomial time solvable. For example, if each teacher is only available for up to two hours, or each teacher is able to teach any class, then the problem is solvable in polynomial time [2].

The basic TTD model often doesn't map well onto several common problems such as scheduling courses for a university. Specifically, the requirement that a teacher *must* teach certain classes may be relaxed to describing those classes they are *willing* to teach. This is know as the Basic Course Scheduling problem (BCS); it was shown to be solvable in polynomial time by Lovelace [3]. Extensions of the BCS, for example including the requirement that courses are assigned to rooms, results again in an NP-hard problem.

The scheduling problem considered in this paper more closely resembles the TTD problem, before introducing it we will give a precise formulation of the TTD. Here we denote the decision variables as a function f, which gives the assignment of presenters to talks and hours. We henceforth refer to the courses, or events as *talks*. We also assume that all talks have the same length and introduce a set of *hours* which is used to represent the set of time slots in which talks can be scheduled.

TIMETABLE DECISION PROBLEM
INSTANCE:

1. a finite set H of hours and numbers n and m indicating the number of presenters and talks, respectively;
2. a collection $P = \{P_1, P_2, \cdots, P_n\}$, where $P_i \subseteq H$ (there are n presenters and P_i is the set of hours during which the ith presenter is available for presenting);
3. a collection $T = \{T_1, T_2, \cdots, T_m\}$, where $T_j \subseteq H$ (there are m talks and T_j is the set of hours during which the jth talk can be given);
4. an $n \times m$ matrix G of nonnegative integers (G_{ij} is the number of hours (times) which the ith presenter will give the jth talk).

QUESTION: Does there exist a function

$$f(i,j,h) : \{1,\cdots,n\} \times \{1,\cdots,m\} \times H \to \{0,1\}$$

(where $f(i,j,h) = 1$ if and only if presenter i gives talk j during hour h) such that
(a) $f(i,j,h) = 1 \Rightarrow h \in P_i \cap T_j$ (the presenter and talk are both available to be scheduled at hour h);

(b) $\sum_{h \in H} f(i,j,h) = G_{ij}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$ (the ith presenter was scheduled for the jth talk the required number of times);
(c) $\sum_{i=1}^{n} f(i,j,h) \leq 1$ for all $1 \leq j \leq m$ and $h \in H$ (no talk has more than one presenter at a time);
(d) $\sum_{j=1}^{m} f(i,j,h) \leq 1$ for all $1 \leq i \leq n$ and $h \in H$ (no presenter is giving more than one talk simultaneously).

3 Conference scheduling problem

We now consider extensions and modifications of the TTD problem that incorporate requirements arising from our application.

3.1 Conference TTD Problem

Although the TTD problem is related to our target problem, it does not capture all of the decisions and constraints involved. One requirement is that some talks may involve multiple presenters, each of which may have additional differing scheduling conflicts. In the TTD model, constraint (c) ensures that each talk is scheduled to exactly one presenter. In the case where multiple presenters are allowed we most likely wish to add a different constraint: that for each talk *every* presenter that can be scheduled is scheduled. For example, if Alice is giving talks A, B, and C, and Bob is giving talks B, C, and D, all scheduled instances of B and C should include both Alice and Bob. We call this variant the Conference Timetable Decision problem (CTTD).

CONFERENCE TIMETABLE DECISION PROBLEM
Same as TTD, but with constraint (c) changed to
(c) $(G_{ij} > 0) \wedge (G_{i'j} > 0) \Rightarrow f(i,j,h) = f(i',j,h)$ for all $1 \leq i,i' \leq n$, $1 \leq j \leq m$ and $h \in H$ (all presenters that are required to give a talk must be present at all instances of that talk);

We now show that CTTD is NP-complete via reduction from the Graph k-colorability problem, the decision problem of determining whether or not a graph admits a k-coloring, which is well known to be NP-complete [4]. As we will see in the proof, the CTTD problem is NP-complete even without the inclusion of the availability constraints and even if each entry G_{ij} is equal to zero or one.

Proposition 1 *CTTD is NP-Complete.*

Proof We first note that CTTD is clearly in NP. Given a graph $G = (V,E)$ and a positive integer k, we will show how to construct an instance of CTTD that is feasible if and only if G is k-colorable. For simplicity of presentation we assume that G contains no isolated nodes (coloring of such nodes is trivial). Essentially, a talk is created to correspond to each vertex in G, each of the k colors corresponds to an hour in which talks can be scheduled and speakers are created to correspond to the edges in G. More formally, define $H = \{1,2,\cdots,k\}$ and for each vertex $v_j \in V = \{v_1, v_2, \cdots, v_{|V|}\}$ let $T_j = H$. For each edge $e_l \in E = \{e_1, e_2, \cdots, e_{|E|}\}$ we let $P_l = H$. For each $e_l = (v_i, v_j) \in E$, where $v_i, v_j \in V$, we let $\hat{G}_{li} = \hat{G}_{lj} = 1$, and let $\hat{G}_{lm} =$

0 for each $m \neq i, j$. We now have an instance $(H, P, T, \hat{G})$ of CTTD (whose construction was easily computed in polynomial time).

We now observe that $(H, P, T, \hat{G})$ has a feasible schedule f if and only if G is k-colorable. Given a feasible schedule f, each vertex v_a in G is assigned color h, where h is the timeslot in which talk a is assigned. For any edge $e_l = (v_i, v_j) \in E$ we note that since a speaker l was created to give talks i and j they will not be scheduled in the same time slots, and thus v_i, v_j are assigned different colors, leading to a valid k-coloring of G. Conversely, given a k-coloring of G it is easy to construct a feasible schedule f for $(H, P, T, \hat{G})$ using the same idea. Finally we conclude that CTTD is NP-Complete.

3.2 Basic Conference TTD Problem

Lovelace showed that a relaxed version of TTD (called BCS for "Basic Course Scheduling") can be solved in polynomial time using a network flow model [3]. The principal differences between BCS and TTD are the reduction of many "hard" requirements such as those insisting that presenters give *exactly* a certain number of talks of a certain type to simply saying they may give *at most* the number of talks for which they are *willing* to give. BCS does maintain a hard requirement that all presentations must be scheduled, but offers flexibility in which speaker makes each presentation. We give a formulation of BCS using notation consistent with our description of the Basic Timetable Decision Problem (BTTD).

BASIC TIMETABLE DECISION PROBLEM
INSTANCE:

1. a finite set H of hours and numbers n and m indicating the number of presenters and talks, respectively;
2. a collection $P = \{P_1, P_2, \cdots, P_n\}$, where $P_i \subseteq H$ (there are n presenters and P_i is the set of hours during which the ith presenter is available for presenting);
3. a collection $T = \{T_1, T_2, \cdots, T_m\}$, where $T_j \subseteq H$ (there are m talks and T_j is the set of hours during which the jth talk can be given);
4. a function $L : \mathbb{Z}^+ \to \mathbb{Z}_0^+$, where $L(n)$ is the maximum number of talks that the nth presenter can give;
5. a function $S : \mathbb{Z}^+ \to \mathbb{Z}_0^+$, where $S(m)$ is the desired number of instances of the mth presentation;
6. a function $WTP : \{1, 2, \cdots, n\} \times \{1, 2, \cdots, m\} \to \{0, 1\}$, where $WTP(i, j)$ indicates if the ith presenter is Willing To Present the jth talk.

QUESTION: Does there exist a function

$$f(i, j, h) : \{1, \cdots, n\} \times \{1, \cdots, m\} \times H \to \{0, 1\}$$

(where $f(i, j, h) = 1$ if and only if presenter i gives talk j during hour h) such that

(a) $f(i, j, h) = 1 \Rightarrow h \in P_i \cap T_j$ (the presenter and talk are both available to be scheduled at hour h);

(b) $\sum_{h \in H} f'(j, h) = S(j)$ for all $1 \leq j \leq m$ where $f'(j, h) = 1 \iff \exists i$ with $1 \leq i \leq n$ such that $f(i, j, h) = 1$, and 0 otherwise (the jth talk is given the required number of times);

(c) $\sum_{j=1}^{m} f(i, j, h) \leq 1$ for all $1 \leq i \leq n$ and $h \in H$ (there is no more than one presenter scheduled for each instance of a talk);

(d) $f(i,j,h) = 1 \Rightarrow WTP(i,j) = 1$ (only presenters willing to give a talk are scheduled for it);
(e) $\sum_{j=1}^{n} \sum_{h \in H} f(i,j,h) \leq L(i)$ for all $1 \leq i \leq n$ (the total number of talks that the ith presenter is scheduled for is at most their maximum number of presentations)
(f) $\sum_{j=1}^{m} f(i,j,h) \leq 1$ for all $1 \leq i \leq n$ and $h \in H$ (no presenter is giving more than one talk simultaneously).

Remark 1 BTTD $\in$ P [3].

The difference between TTD and CTTD is the ability for presentations to have multiple presenters, and the requirement that all presenters be scheduled for all instances of a talk. Likewise, we can formulate a modified version of BTTD that incorporates this new constraint which we will call the Basic Conference Timetable Decision problem (BCTTD).

BASIC CONFERENCE TIMETABLE DECISION PROBLEM
Same as BTTD, but with constraint (c) changed to
(c) $WTP(i,j) = WTP(i',j) \Rightarrow f(i,j,h) = f(i',j,h)$ for all $1 \leq i,i' \leq n$, $1 \leq j \leq m$ and $h \in H$ (all presenters that are required to give a talk must be present at all instances of that talk);

We observe that after this constraint is introduced, we may apply the same reduction used in Proposition 1 and thus we have the following.

Proposition 2 *BCTTD is NP-Complete.*

3.3 Extended Conference TTD Problem

We now present a modification to CTTD that introduces room assignment decisions and room compatibility constraints. The CTTD problem assigns speakers to talks and time slots but, as in many other applications, we also require that talks are assigned to suitable rooms. Furthermore, rooms also may only be available during certain times or suitable for certain talks and this information must be factored into the problem. This leads us to the Extended Conference Timetable Decision problem (ECTTD).

EXTENDED CONFERENCE TIMETABLE DECISION PROBLEM
INSTANCE: Same as CTTD, but with the additional parameters:
5. a finite set R of rooms;
6. a collection $\{A_1, A_2, \cdots, A_r\}$, where $A_k \subseteq H$ (there are $r = |R|$ rooms and A_k is the set of hours during which the kth room is available);
7. a collection $\{S_1, S_2, \cdots, S_m\}$, where $S_l \subseteq R$ (there are m talks and S_l is the set of rooms that the lth presentation may be given in)

QUESTION: Does there exist a function

$$f(i,j,h,r) : \{1,\cdots,n\} \times \{1,\cdots,m\} \times H \times R \to \{0,1\}$$

(where $f(i,j,h,r) = 1$ if and only if presenter i gives talk j during hour h in room r) such that
(a) $f(i,j,h,r) = 1 \Rightarrow h \in P_i \cap T_j \cap A_r \wedge r \in S_j$ (the ith presenter, jth presentation and room r are all available to be scheduled at hour h and room r is suitable for the jth presentation);

(b) $\sum_{r\in R}\sum_{h\in H} f(i,j,h,r) = G_{ij}$ for all $1 \le i \le n$ and $1 \le j \le m$ (the ith presenter was scheduled for the jth presentation the required number of times);
(c) $G_{ij} > 0 \wedge G_{i'j} > 0 \Rightarrow f(i,j,h,r) = f(i',j,h,r)$ for all $1 \le i,i' \le n$, $1 \le j \le m$, $h \in H$, and $r \in R$ (all presenters that are required to give a talk must be present at all instances of that talk);
(d) $\sum_{r\in R}\sum_{j=1}^{m} f(i,j,h,r) \le 1$ for all $1 \le i \le n$ and $h \in H$ (no presenter is giving more than one talk simultaneously);
(e) $\sum_{j=1}^{m} f'(j,h,r) \le 1$ for each $h \in H$ and $r \in R$ where $f'(j,h,r) = 1 \iff \exists i$ with $1 \le i \le n$ such that $f(i,j,h,r) = 1$, and 0 otherwise (room r is scheduled for at most one talk at hour h).

We also note that since this is a clear generalization of CTTD, and in the class NP, it is also NP-Complete.

Proposition 3 *ECTTD is NP-Complete.*

3.4 Preference Conference Optimization Problem

We have examined several different variations of scheduling problems as they relate to conferences; we now offer a final variation that will be the subject of study for the rest of the paper. In particular we are interested in not only finding a schedule that is feasible with respect to speaker and room logistics, but one that also minimizes attendee preference conflicts. Formally, an *attendee preference conflict* is a tuple (e, j, j') where e is an attendee, j, j' are two events for which e has indicated an interest to attend, and j, j' are scheduled to occur during the same time slot. Namely, given the set of conference attendees and their preferences for talks they would like to attend, we want to minimize the total number of times that a given attendee has shown preference for a pair of talks that are scheduled in the same time period. We call the resulting optimization problem the Preference Conference Optimization problem (PCO).

PREFERENCE CONFERENCE OPTIMIZATION PROBLEM
INSTANCE: Same as ECTTD, but with the additional parameters:
8. a finite set $E = \{e_1, e_2, \cdots, e_t\}$ of attendees;
9. a $t \times m$ 0-1 matrix W (W_{ej} indicates if the eth attendee would like to attend the jth talk).

GOAL: Find a function

$$f(i,j,h,r) : \{1,\cdots,n\} \times \{1,\cdots,m\} \times H \times R \to \{0,1\}$$

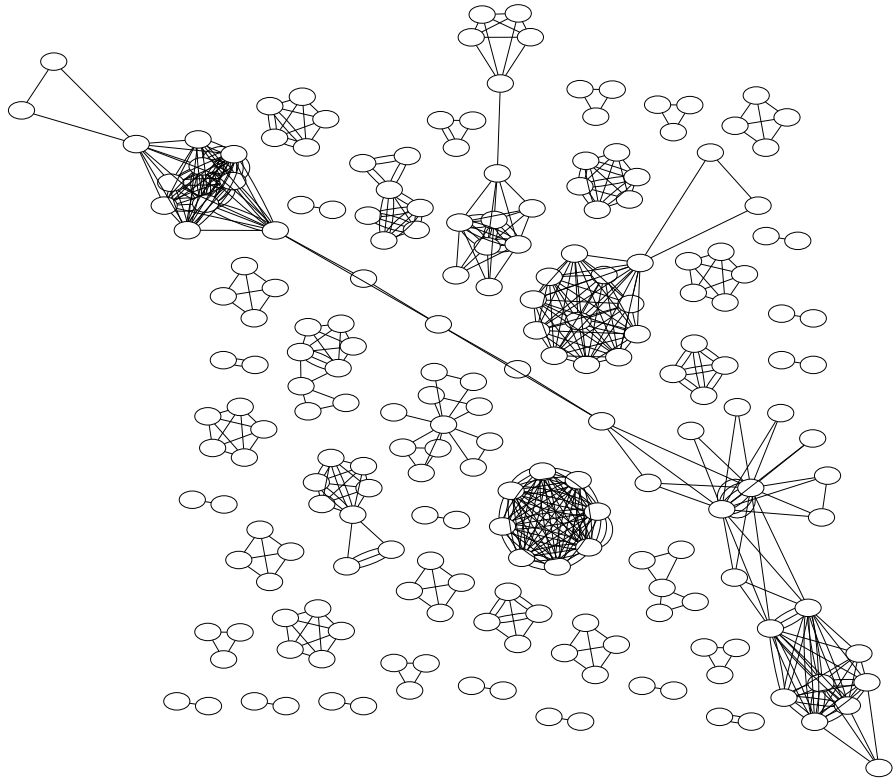
that satisfies all the constraints of the ECTTD problem while minimizing the sum of the attendee preference conflicts where, as described above, an attendee preference conflict is any tuple (e, j, j') such that there exist i, i', h, r and r' such that $f(i,j,h,r) = f(i',j',h,r') = 1$ where $W_{ej} = 1$ and $W_{ej'} = 1$.

4 Integer programming models

We will present integer programming models that can solve PCO and ECTTD. The data used to measure the models comes from a real conference held in 2013, which we shall

refer to as PC2013. PC2013 had 195 presenters giving a total of 253 talks. Figure 1 helps illustrate the data we worked with: each vertex represents a talk and adjacent vertices share a common presenter and cannot be scheduled at the same time. By Proposition 1, solving CTTD is equivalent to asking if this graph admits an h-coloring (where h is the number of time slots available at the conference).

Fig. 1 Presenter conflicts that must be scheduled around in PC2013



4.1 Model for the Extended Conference Timetable Decision Problem

Before describing the model for the PCO problem, we give an integer programming model where feasible solutions determine values of functions f which correspond directly to schedules that satisfy the constraints laid out in the ECTTD decision problem. We note that the size of f, i.e. the number of variables in our model, can be very large when building models corresponding to our application data.

$$\text{size of } f = \#\text{ of presenters } \times \#\text{ of talks } \times \#\text{ of hours } \times \#\text{ of rooms}$$

For PC2013, size of $f = 193 \times 253 \times 37 \times 15 = 27{,}421{,}440$. This number of variables could be problematic computationally, however, we know that many (nearly all) of these variables will be zero based on information we have at formulation time. For example, if a presenter

i doesn't give talk j, then $f(i,j,h,r) = 0$ for all $h \in H, r \in R$. Although integer programming solver may automatically fix such variables to zero in the preprocessing phase, we exclude these variables from the model at the time of construction. We create an index set $\mathscr{F} \subseteq \{1,\cdots,n\} \times \{1,\cdots,m\} \times H \times R$ where $f_{i,j,h,r} \in \mathscr{F}$ only if presenter i gives talk j; the talk j, presenter i and room r are available at hour h; and room r is suitable for the jth talk. In addition to this condition, we restrict the inclusion of variable indices in $\mathscr{F}$ to the intersection of the available hours of all *co-presenters* (pairs of presenters that give the same talk), e.g. if co-presenters i, i' have availability sets $\{h_2, h_3\}$ and $\{h_3, h_4\}$ (assuming room and talk availability is at least $\{h_2, h_3, h_4\}$) then only variables with $h = h_3$ for these co-presenters and talk will be included. In practice, the reduction of our solution space to only $\mathscr{F}$ gives a massive performance gain. For PC2013, this immediately reduced the number of variables down to 91,514 (a reduction of 99.997%).

The variables indexed by $\mathscr{F}$ can be thought of as a sparse representation of the interesting elements of the domain of f. In addition to $\mathscr{F}$ we will use the index set $\mathscr{G}$ to represent tuples (j,h,r) for which talk j can be given by any presenter at hour h in room r, and variables $g_{j,h,r}$ will indicate whether or not this occurs. We will now describe a formulation that implements each of the constraints on f in ECTTD.

ECTTD formulation

$$\text{minimize: } 0 \tag{1}$$

$$\text{subject to:} \tag{2}$$

$$\sum_{h,r:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r} = G_{ij} \qquad \text{for every presenter } i \text{ and talk } j \tag{3}$$

$$f_{i,j,h,r} - f_{i',j,h,r} = 0 \qquad \text{for every talk } j \text{ with co-presenters } i,\ i' \tag{4}$$

$$\sum_{j,r:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r} \leq 1 \qquad \text{for every presenter } i \text{ and hour } h \tag{5}$$

$$\left(\sum_{i:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r}\right) - U \times g_{j,h,r} \leq 0 \qquad \text{for each } g_{j,h,r} \in \mathscr{G} \tag{6}$$

$$\sum_{j:(j,h,r)\in\mathscr{G}} g_{j,h,r} \leq 1 \qquad \text{for each hour } h \text{ and room } r \tag{7}$$

$$\text{binary: } f_{i,j,h,r}, g_{j,h,r} \tag{8}$$

The first requirement (a) of ECTTD merely enforces all availability and suitability sets. We implicitly enforce this in our model by considering only the variables indexed over $\mathscr{F}$. As such, no specific constraints are needed in our model.

The second requirement (b) ensures that every presenter is scheduled for all of their talks, which we receive as parameter G to ECTTD where G_{ij} is the number of times presenter i should give talk j. For each presenter i and talk j, the sum of the times they are scheduled (over all hours and rooms) should be G_{ij}; this is constraint (3).

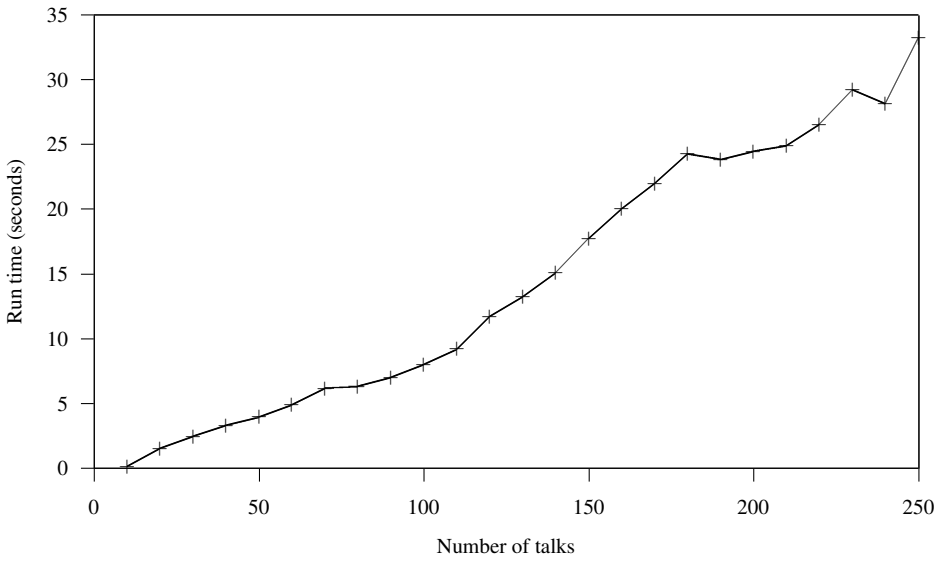
To ensure requirement (c) we force that all co-presenters have the same schedule for their shared talk (constraint (4)).

Requirements (a)-(c) guarantee that all presenters are scheduled for their talks and that co-presenters are scheduled together. Requirement (d) ensures that if a presenter has multiple talks, then these talks are scheduled during different hours. For each presenter i and hour h, the sum of their schedule variables for their talks in all rooms must be less than or equal to one (constraint (5)).

The final requirement (e) ensures that room scheduling is exclusive. We would like to simply iterate over $\mathscr{F}$ for a particular pair of hour h and room r, summing all of these together. If we didn't allow co-presenters (like TTD) then we could simply make this sum less than or equal to one. But, for talks with co-presenters this sum varies. To overcome this we create indicator variables $g_{j,h,r}$ where $g_{j,h,r} = 1$ whenever talk j is scheduled at hour h in room r; this is modeled in constraint (6), where U represents a sufficiently large number. Finally, constraint (6) ensures that no room is multi-booked by checking the sum of $g_{j,h,r}$ for each pair h, r.

Solving this feasibility problem proved tractable. We solved this formulation with the open source IP solver CBC on PC2013 with varying numbers of talks pruned out to see how the model scales. The results are given in Figure 2.

Fig. 2 Run time of ECTTD model



4.2 Model for the Preference Conference Optimization Problem

We now turn our attention to the Preference Conference Optimization (PCO) problem. PCO adds an additional layer of complexity to ECTTD by including a matrix of preferences for attendees with the goal of minimizing the number of conflicts caused by concurrent talks. Through experimentation we have found that considering these preferences significantly increases the difficulty of solving our conference scheduling problem. We now present an integer programming model for PCO.

The PCO model we present builds on our previous ECTTD model. In addition to the G variables which collapse the four dimensional f function down to three dimensions (talk $\times$ hour $\times$ room), all the while considering only those variables for which a feasible schedule is even possible given the different availability constraints, we will introduce three new classes

of variables for PCO. The first is z which will collapse g to two dimensions (talk $\times$ hour). Next, we will expand z to c which will have a three dimensional range from talk $\times$ talk $\times$ hour; it will indicate if two talks j, j' are given concurrently at hour h. As with previous models, the corresponding script (e.g. $\mathscr{Z}$ for z) will represent the index set of variables that are possible given availability constraints. The parameter w is constructed from W as a talk $\times$ talk $\times$ hour matrix where each entry is the number of attendees who wish to attend both talks j, j' for all h when j, j' can be given based on talk, presenter, and room availability. Formally, $w_{j,j',h} = |\{e_k \in E : e_k \text{ has } W_{kj} = 1 \text{ and } W_{kj'} = 1\}|$ for each h where $(h,j) \cup (h,j') \subseteq \mathscr{Z}$. The objective function that is minimized in the model is the sum of elements in c.

PCO formulation

$$\text{minimize:} \sum_{(j,j',h)\in\mathscr{C}} w_{j,j',h} \times c_{j,j',h} \tag{9}$$

subject to: (10)

$$\sum_{h,r:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r} = G_{ij} \qquad \text{for every presenter } i \text{ and talk } j \tag{11}$$

$$f_{i,j,h,r} - f_{i',j,h,r} = 0 \qquad \text{for every talk } j \text{ with co-presenters } i,\ i' \tag{12}$$

$$\sum_{j,r:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r} \leq 1 \qquad \text{for every presenter } i \text{ and hour } h \tag{13}$$

$$\left(\sum_{i:(i,j,h,r)\in\mathscr{F}} f_{i,j,h,r}\right) - U \times g_{j,h,r} \leq 0 \qquad \text{for each } (j,h,r) \in \mathscr{G} \tag{14}$$

$$\sum_{j:(j,h,r)\in\mathscr{G}} g_{j,h,r} \leq 1 \qquad \text{for each hour } h \text{ and room } r \tag{15}$$

$$\left(\sum_{j,h:(j,h,r)\in\mathscr{G}} g_{j,h,r}\right) - U \times z_{j,h} \leq 0 \qquad \text{for each room } r \tag{16}$$

$$\sum_{h:(j,h)\in\mathscr{Z}} z_{j,h} = G_{ij} \qquad \text{for each talk } j \text{ and some presenter } i \tag{17}$$

$$z_{j,h} + z_{j',h} - c_{j,j',h} \leq 1 \qquad \text{for each } (j,j',h) \in \mathscr{C} \tag{18}$$

$$\text{binary: } f_{i,j,h,r},\ g_{j,h,r},\ z_{j,h},\ c_{j,j',h} \tag{19}$$

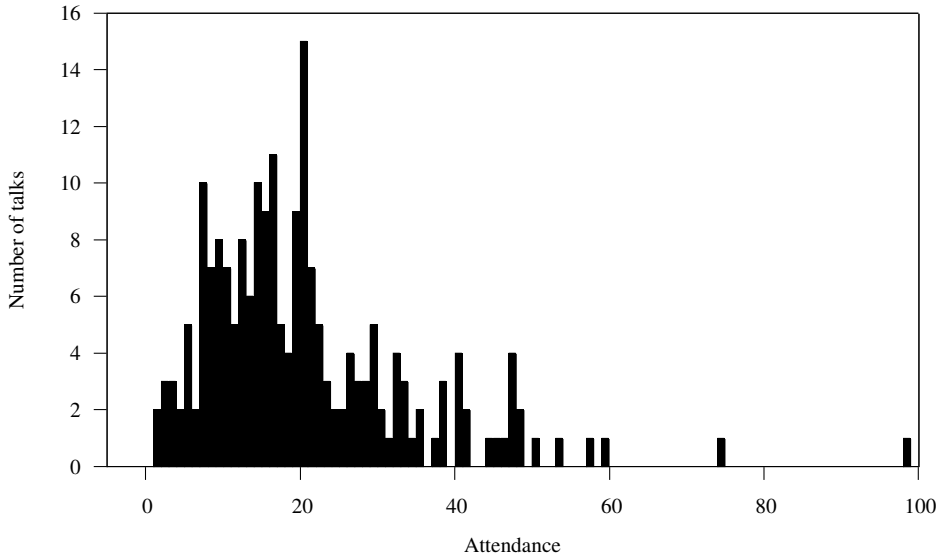
Constraints (11) - (15) are the same as in our model for ECTTD. Constraint (16) begins to build the z variables which will be 0-1 indicators of talk j being given at hour h via the same boolean cast mechanism described previously by collapsing the room entries for j,h in g. To ensure that only the correct number of zs are set to one, constraint (17) sums all hours h for each talk j and sets it equal to the number of times that talk j was set to be given in the problem instance (the matrix G). It is of note that we pick *any* presenter i's entry in G for talk j; although it is possible that a co-presenter i' may have a different value for $G_{i'j}$ requirement (c) of PCO explicitly forbids this since it would be impossible for all co-presenters to be at all instances of a talk if they had different entries for their shared talk j, thus we can pick any presenter i.

The z variables will now be used to generate c, which indicates if a pair of talks j, j' are being given concurrently at hour h. Specifically, constraint (18) , enforces that $z_{j,h} = 1$ and $z_{j',h} = 1 \implies c_{j,j',h} = 1$. The objective (9) of the model is to minimize the sum of attendee preference conflicts. Each variable $c_{j,j',h}$ appears with coefficient of $w_{j,j',h}$, which is the number of attendee preference conflicts generated by talks j and j' being scheduled in the

same time period. Note that if the coefficient $w_{j,j',h}$ is nonzero, then the minimization nature of the problem will force the corresponding $c_{j,j',h}$ to take zero value whenever possible.

To measure the efficiency of our model we tested it on our sample set PC2013. This data set included information about talks, speakers and attendance, but did not include attendee preferences (as they were not solicited that year), however we may use this data to generate reasonable instances by taking the historical attendance data as a basis for generating hypothetical attendee preferences. Attendance figures we recorded for each talk given at PC2013; a distribution of the attendance per talk is shown in Figure 3. The sum of all atten-

Fig. 3 Distribution of attendance at PC2013



dance counts was 4101 [6] for around 1000 unique attendees. For the purposes of testing our model created a W such that $\sum_{i=0}^{n} W_{ij}$ was equal to the attendance count for that talk j, i.e. we created an indicated attendance preference for each individual talk attendance at PC2013. Since individual attendee attendance wasn't tracked (only totals were) we took some liberties in distributing the preference responses across the attendees in our model. We first randomly spread the preference responses over the attendees using a uniform distribution; that is, if talk j had an attendance of 24 in PC2013 then 24 attendees were randomly chosen to express a preference for attending this talk. We solved our model with commercial solver Gurobi which returned a solution with an objective value of 0 after 64 seconds, i.e. a schedule with absolutely no attendee conflicts. All computations were run on a machine with 4 12-core Intel Xeon E5-2695 CPUs running at 2.4 GHz with 96 GB of RAM

After finding a non-conflicting schedule for uniformly distributed random attendees that followed the attendance counts in PC2013 we turned our attention to how the solver would react when the random attendees were not distributed uniformly. Our intuition was that, like the actual attendance figures, the distribution of attendance per attendee would not be evenly spaced out; there would be some attendees who went to many talks, and some who went to

only a few. We chose a normal distribution with $\mu = 500, \sigma = 100$. For each attendance in the PC2013 distribution (Figure 3) a random integer from the normal distribution was chosen. Since $\mu = 500$ those attendees with index around 500 were much more likely to be chosen to indicate a preference to the talk then those with indices 50 or 950, which implemented our intuition that a small percentage of attendees will indicate attendance preferences for many talks while the bulk of those remaining will indicate preferences for relatively few. In addition, we added a check to ensure that no attendee was selected to attend more talks than were hours available; such a scenario would automatically preclude a zero objective value. Even with an extremely powerful computer to run our model on and a state-of-the-art commercial solver we were unable to solve this instance after 24 hours. To help understand this phenomenon we created a half-sized problem (half as many talks and hours) and ran the model with decreasing values of σ and found that the solving time exploded exponentially as σ decreased; the average running time for $\sigma = 200$ was 22 seconds but increased to 22,683 seconds for $\sigma = 100$.

4.2.1 Performance considerations

After including attendee preferences in our test models, the integer programming models became significantly more difficult to solve. One possible cause of this is symmetry present in the models, a property that often leads to increased solution time [7]. An integer programming model is said to be *symmetric* if some of its variables can be permuted (nontrivially) without changing the structure of the problem [8]. Our model exhibits high amounts of symmetry in relation to the scheduling of talks in rooms. If two rooms have the same availability and suitability set then permuting talk assignments among them in each hour produces no discernible change to the objective. It may be, however, that the solver will choose to branch early on in its branch-and-bound tree on these room assignments, leading to lots of unnecessary computation. In general it is difficult for the solver to detect that such variables "really" represent the same thing, although there are several mechanisms for determining and avoiding symmetry in solvers [9]. However, it is easy for us to identify this symmetry and avoid it.

Two rooms will be said to be *symmetric* if they have the same suitability and availability sets, i.e. for rooms R_α and R_β we have that R_α and R_β are symmetric if and only if

$$A_\alpha = A_\beta \quad \text{and}$$
$$\{i \mid R_\alpha \in S_i \text{ for all } 1 \leq i \leq m\} = \{i \mid R_\beta \in S_i \text{ for all } 1 \leq i \leq m\}.$$

To break the symmetry we create room *classes* which will represent several rooms with the same attributes. First, we make a new room set $R' = \{r' = \{r_1, r_2, \cdots, r_p\} \subseteq R \mid \text{all } r \in r' \text{ are symmetric with each other}\}$. The corresponding new availability set A' has simply the common availability set for each new $r' \in R'$. For the new suitability set S' we replace each instance of r with the room class r' that r is a member of. When we solve our model talks will be booked to room classes, avoiding the symmetry that arises by having to consider two essentially "equal" rooms separately. When our model is solved we will have bookings in room classes, and we can arbitrarily assign the talk to any room in that class. We must make only one adjustment to our model: Constraint (15) in the PCO model ensures that each room has only one talk booked in it per hour. For our room classes we wish to relax this, requiring only that the number of bookings be at most the number of rooms in the class; this way, when we assign actual rooms from the solved model we can match talks to rooms in a

one-to-one way. Formally, we will change constraint (15) to

$$\sum_{j \in \mathscr{G}_{h,r}} g_{j,h,r} \leq |r|. \tag{20}$$

It is easy to see that this model degenerates to our regular PCO model when no rooms are symmetric; in this case each room class would contain only one room. In practice the removal of the room symmetries increased performance by roughly a factor of 5x for our solver on PC2013, which contained only three room classes but had fourteen rooms (see Figure 4).

The next step we took to improve performance was to *dualize* one class of constraints. In this technique, these requirements are moved from being hard constraints in the model, to appearing in the objective function with a sufficiently large penalty to ensure their satisfaction. We chose to dualize constraint (d), namely that no presenter is scheduled for more than one talk per hour. Intuitively, this seemed like a promising adjustment because these constraints are similar in structure to the attendee preference conflicts that are minimized in the objective function. We first created 0-1 indicator variables $d_{i,h} = 1 \iff$ presenter i is doubly (or more) booked at hour h by changing (13) (which enforced (d)) to

$$\left(\sum_{j,r \in \mathscr{F}_{i,h}} f_{i,j,h,r} \right) - U \times d_{i,h} \leq 1. \tag{21}$$

Where U is a sufficiently large number. The left hand side of (21) is the number of times that presenter i is scheduled at hour h, and $d_{i,h}$ may be 0 or 1 if this sum is less than 2, but must be 1 if the sum is 2 or greater. We then changed the objective (9) to

$$\text{minimize:} \sum_{(j,j',h) \in \mathscr{C}} w_{j,j',h} \times c_{j,j',h} + \sum_{(i,h) \in \mathscr{D}} U \times d_{i,h}. \tag{22}$$

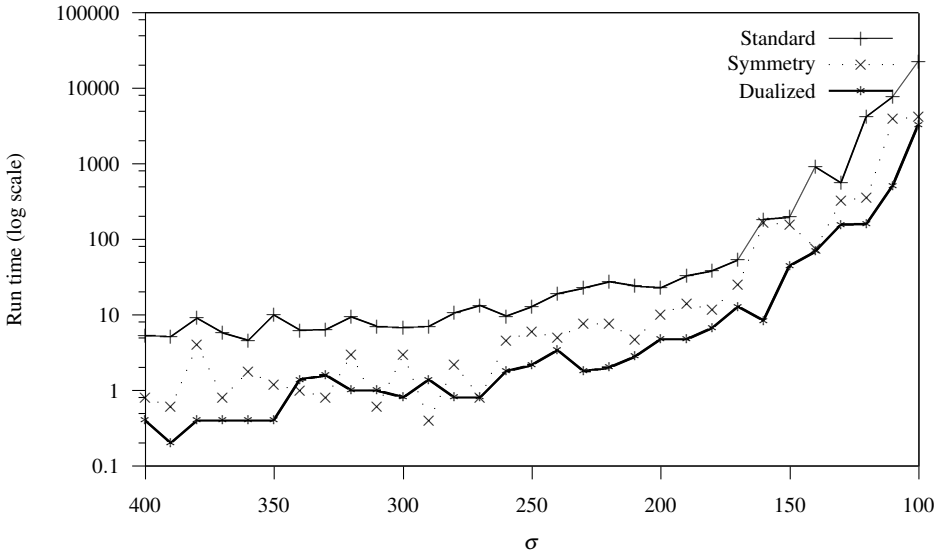
Our new objective places a penalty of U on presenters being multiply booked. We should choose U sufficiently large so d is identically zero, otherwise the model can be resolved with a larger value of U. In our experiments the dualized constraints were always satisfied after solving the model. In practice, dualizing PCO led to moderate performance increases of roughly 75% faster.

For a summary of solution times comparing the original model with the improved models discussed in this subsection see Figure 4. The three models compared are: the Standard model, which corresponds to the PCO formulation given by (9)-(15); the Symmetry model, which incorporates the symmetry breaking reformulation described above; and finally the Dualized model, which incorporates both the symmetry breaking reformulation, and the dualization of constraint (d) as described above. The graph plots the average running time for solving 10 randomly generated instances for values of σ between 100 and 400 with increments of 10. The table shows the same information, only listing times for instances where σ is a multiple of 50. All times are listed in seconds. We also note that it turned out that for the generated instances solved in these experiments, the optimal solutions had an objective value of zero.

5 Conclusion

Conference scheduling represents an important class of timetabling problems. This paper studies a conference scheduling problem where attendee preference conflicts are minimized,

Fig. 4 Run time of PCO model with decreasing σ



(σ)	Standard	Symmetry	Dualized
400	5.4	0.8	0.4
350	10.0	1.2	0.4
300	6.8	3.0	0.8
250	12.8	6.0	2.2
200	22.8	10.2	2.0
150	196.2	156.8	45.2
100	22683.7	4179.2	3282.6

subject to a collection of hard constraints. We have demonstrated integer programming to be an effective solution technique, especially after incorporating symmetry breaking and other improvements.

References

1. Even, S., A. Itai, and A. Shamir. 1976. On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing 5, (4) (12): 691-13
2. Garey, M., and D. Johnson. 1976. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman
3. Lovelace, A. 2010. On the complexity of scheduling university classes. M.S. in Computer Science Thesis. California Polytechnic State University: U.S.A.
4. Garey, M., D. Johnson, and L. Stockmeyer. 1976. Some simplified NP-Complete graph problems. Theoretical Computer Science 1: 237-267
5. Penguicon Conference, `http://www.penguicon.org/`, Accessed: January, 2015.
6. Penguicon Programming Ops. `http://penguicon.info/doku.php/programmingops?s=attendance`. Accessed: January, 2015.
7. Sherali, H.D., and J.C. Smith. 2001. Improving Discrete Model Representations via Symmetry Considerations. Managements Science 47: 1396-1407.
8. Margot, F. 2009. Symmetry in Integer Linear Programming. 2010. 50 Years of Integer Programming 1958-2008, Chapter 16: 647-681. Springer.
9. Ostrowski, J. 2008. Symmetry in Integer Programming. Ph.D. Thesis. Lehigh University: U.S.A.

MISTA 2015

Bounding schemes for the parallel processors scheduling problem with release date, delivery time and with no-idle time constraint

Lotfi Hidri • Achraf Gazdar

Abstract: In this paper, we address the parallel processors scheduling problem with release date, delivery time and no-idle time. In this problem we are given a system composed of parallel processors intended to treat a family of tasks. These tasks are characterized by a release date (arrival time), processing time and delivery time. In addition, the idle time between consecutive tasks is not allowed during the treatment on each processor. The no-idle time constraint is intended to avoid wasting the consumed energy while treating tasks. The objective is to provide a feasible schedule that minimizes the completion time of the last treated task (makespan). In order to solve this optimization scheduling problem we propose a tight and new lower bound, which is based on the optimal solution of a relaxed parallel processors scheduling problem. In addition a family of two-phase heuristics, providing a near optimal solution is presented. More precisely, Phase 1 is intended to construct an initial feasible schedule that is improved in Phase 2. Finally, we present the results of extensive computational experiments in order to evaluate the performance of the proposed procedures.

1 Introduction

The parallel computing consists in treating several tasks simultaneously, using many processors instead of one ([1], [5]). Therefore, large problems can be divided into small ones treated concurrently (in parallel) [14], [2]). This kind of treatment allows speeding up the processing of complex problems. Consequently, the parallel computing attracted the attention of the computer science community and a lot of literature was provided.

Several real life applications owe their spectacular progress to the parallel computing advances, as in aerospace engineering, mechanical engineering, civil engineering, mathematical optimization, medicine, biology , chemistry, high performance computing, transportation, management, etc. ([11], [15], [10]). These spectacular technological progresses are due to the simulation of a lot of phenomena which becomes possible thanks to high performance computing offered by the parallel computing.

The costs of the computing hardware (computer, network, and storage) are decreasing dramatically, however the power consumption, cooling and buildings facilities required by the

Lotfi Hidri
Department of Industrial Engineering,
College of Engineering, King Saud University,
E-mail: lhidri@ksu.edu.sa

Achraf Gazdar
Department of Software Engineering,
College of Computer Sciences and Information Systems, King Saud University,
E-mail: agazdar@ksu.edu.sa

parallel computing deployment are becoming more and more expensive. Thus, the costs of the needed infrastructure and the consumed power exceed the computing equipment itself. Recent statistics inquiries show that 2% of the greenhouse released gazes are due to the computing power consumption and that these gazes's emissions are increasing by 6% each year [17]. Therefore a new field of research emerges; it is the High Performance Green Computing. This field is interested in providing new solutions that reduce the computing equipment's power consumption.

In this context we propose some solutions that contribute in the power consumption reduction. For that aim we assume that a set of tasks have to be processed by a set of parallel processors, and we have to assign these tasks to the processors such that the consumed electric energy is reduced. Saving the consumed power is done by finishing the treatment of the tasks as soon as possible (minimizing the total completion time) under the no-idle constraint. This constraint requires that the idle time between the consecutive tasks on each processor during the treatment of the tasks is eliminated, which participates in the electric energy saving. The obtained problem is a scheduling one. Scheduling can be viewed as an assignment of scarce resources (processors) to specific items (tasks), in order to optimize an objective function. During the last two decades, parallel processor scheduling problem has drawn a lot of attention and has become the subject of an extensive study ([16], [7], [4]). For a detailed survey the reader is referred to [5] and the references therein.

It is worth noting that most scheduling research has been focused on deterministic scheduling which generates enough variety. In contrast to stochastic scheduling problems, deterministic problems assume that all parameters are known in advance. Although processing times and task arrivals may be subject to fluctuations, the deterministic assumption may be suitable for several practical situations in particular for our case where we limit our study to the deterministic parameters [20]. Indeed, in many cases, these fluctuations are of no significant impact on the quality of the schedule. Also, the deterministic assumption is often imposed by certain applications such as in computer control systems working in a hard-real-time environment [18]. Moreover, there are many cases where a simple rule which is merely a heuristic for the deterministic model has a stochastic reformulation which solves the stochastic model to optimality [19]. This allows the studying of the stochastic variants of our problem.

The paper is organized as follows: in Section 2, we briefly define the treated scheduling problem. In Section 3, a new lower bound is proposed. In section 4, a family of two-phase heuristics, providing a near optimal solution is presented. In Section 5, we present the results of an extensive computational analysis of the different lower bounds and heuristics. Finally, we conclude by providing a summary of our results and indicating some directions for future research.

2 Problem definition

In order to model the parallel computing with energy saving, we treat in this work the parallel processors scheduling problem with release date, delivery time and with no-idle time, which is stated as follows. We are given a set $J = \{1, 2,\ldots,n\}$ of n tasks that have to be processed on m identical parallel processors, denoted M_i for $(i = 1, 2,\ldots, m)$. In addition, each task $j \in J$ is characterized by:

- r_j: a release date from which j is ready to be processed(arrival time).
- p_j: a processing time on a processor.
- q_j: a delivery time that elapsed between the processing completion in a processor and the exiting of the system(communication for example).

The processing of the tasks is done under the following constraints:

- Idle time between consecutives tasks is not permitted during the processing of tasks for each processor.

- Each task $j \in J$ is totally processed in only one processor.
- For all $j \in J$, r_j, p_j and q_j are assumed to be integral and deterministic.
- All the processors are ready for processing tasks from time zero.
- Preemption is not allowed during the processing of each job $j \in J$.
- Each processor treats at most one job at the same time.

Our objective is to find a feasible schedule, that minimizes the completion time of the last treated job C_{max} (makespan). Following the Graham's notation [9] this problem will be denoted $P_m,NI/r_j,q_j / C_{max}$. This problem is NP-Hard in the strong sense since its relaxation $P_m /r_j,q_j / C_{max}$ is NP-Hard in the strong sense ([7], [8]). At the best of our knowledge this problem is not treated in literature except for the one processor with release date, delivery time and with no-idle time ([13], [3], [12]).

Example 1: Consider the following instance: $m = 2$ and $n = 5$, the processing times, release dates and delivery times are displayed in Table 1.

j	r_j	p_j	q_j
1	2	6	3
2	8	7	2
3	5	3	4
4	3	3	16
5	7	9	6

Table 1: Data of example 1

In this case, we have the following feasible (Figure 1) schedule with makespan 23:

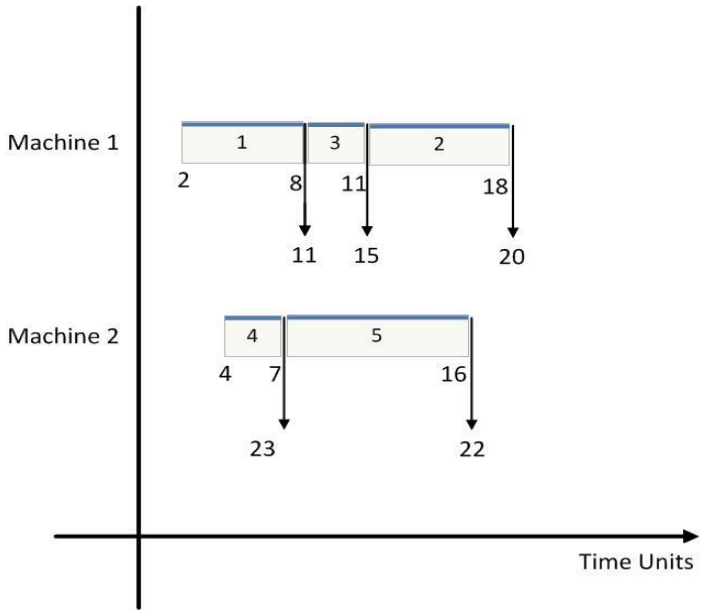


Figure 1: Gantt chart of a feasible schedule having a makespan equal to 23

3 Lower bound and heuristics

3.1 Lower bound

In this subsection, we propose a new lower bound for the treated scheduling problem. This lower bound is based on the optimal solution of the parallel processors scheduling problem with release date and delivery time. In this context we present the following Lemma.

Lemma 1: The value C^*_{max} of an optimal solution for the $P_m /r_j,q_j/C_{max}$ problem is a lower bound for the $P_m,NI/r_j,q_j / C_{max}$ problem.

Proof: An optimal schedule for $P_m,NI/r_j,q_j / C_{max}$ with optimal value C^{NI}_{max} is a feasible schedule for $P_m /r_j,q_j/C_{max}$, thus $C^*_{max} \leq C^{NI}_{max}$ and C^*_{max} is a lower bound for $P_m,NI/r_j,q_j / C_{max}$, which is referred to as ***LB*** *(LB*= C^*_{max}).

It is worth noting that the optimal value C^*_{max} is obtained using the algorithm provided in [7]. Now we return back to the example 1 where the obtained lower bound *LB = 22* which corresponds to the schedule presented in the Figure 2.

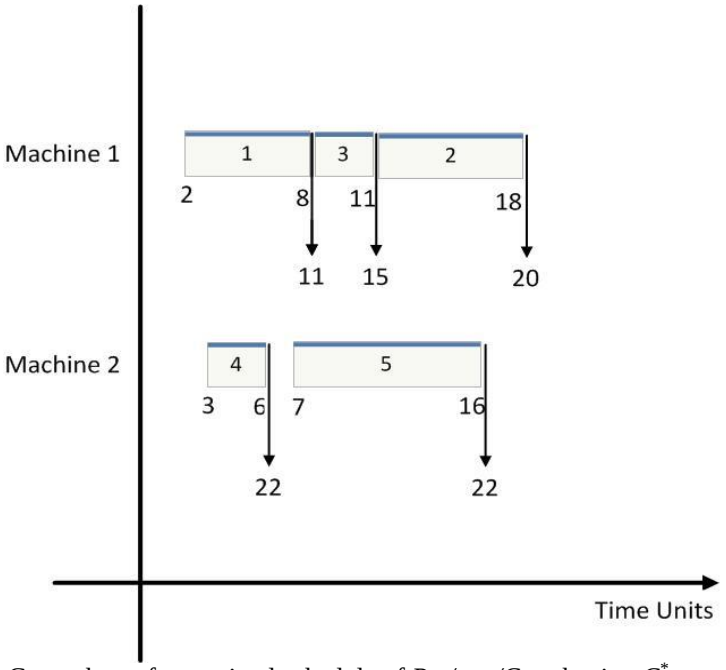


Figure 2: Gantt chart of an optimal schedule of $P_m /r_j,q_j/C_{max}$ having $C^*_{max} = LB = 22$

3.2 Heuristics

In this section we develop a family of heuristics composed of two phases; the first one is constructive and the second phase is an improvement one.

3.2.1 Heuristic $H_{opt\text{-}opt}$

Phase 1: The constructive phase for this heuristic $H_{opt\text{-}opt}$ is based on solving the $P_m / r_j,q_j / C_{max}$ problem. The obtained optimal solution *S* will present one of the following cases:

- There is no idle time in the schedule and then the *S* is an optimal solution for the $P_m,NI / r_j,q_j / C_{max}$ problem.

- There are idle times that will be eliminated by ***right shifting*** the tasks until there is no idle time. In this case the obtained schedule is denoted S^R. If after the right shifting the makespan remains C^*_{max} then S^R is an optimal solution for the $P_m,NI / r_j,q_j / C_{max}$ problem and the procedure is halted.
- There are idle times and the right shifting changes the makespan value, in this case we denote the obtained solution by S^R_1.

Phase 2: Once we get S^R_1 we introduce the following notations that will be required in the improvement phase:

- J_k $(k = 1, \ldots, m)$ the subset of the scheduled jobs on machine M_k
- $C_k = max\{C(j), j \in J_k\}$, with $C(j)$ denotes the completion time of job j.

In addition, we assume that $C_1 \leq C_2 \leq \cdots \leq C_m$. The improvement phase (**Phase 2**) itself consists on selecting iteratively a couple of machines (M_k, M_m) and solving the $P_2 /r_j,q_j/C_{max}$ problem on the $J_k \cup J_m$ job subset $(1 \leq k \leq m\text{-}1)$ with C^k_{max} the optimal corresponding value. It is worth noting that machine M_m contains a job with maximum completion time C_m. If $C^k_{max} \leq C_m$, then the old schedule on machines M_k and M_m is replaced by the new one and the procedure is reiterated once again until there is no improvement: $C^k_{max} = C_m$ $(k = 1, \ldots, m\text{-}1)$ and the obtained feasible schedule is denoted S^R_2 with makespan $UB_{opt\text{-}opt}$

To illustrate the latter procedure we introduce the following example.

Example 2: Consider the following instance: $m = 3$ and $n = 10$, the processing times, release dates and delivery times are displayed in Table 2.

j	1	2	3	4	5	6	7	8	9	10
r_j	1	11	10	3	3	5	9	5	2	7
p_j	8	6	10	1	6	4	10	8	3	10
q_j	8	10	1	7	1	18	3	3	8	10

Table 2: Data of example 2

Solving the $P_m / r_j, q_j / C_{max}$ corresponding problem gives the following feasible schedule (Figure 3):

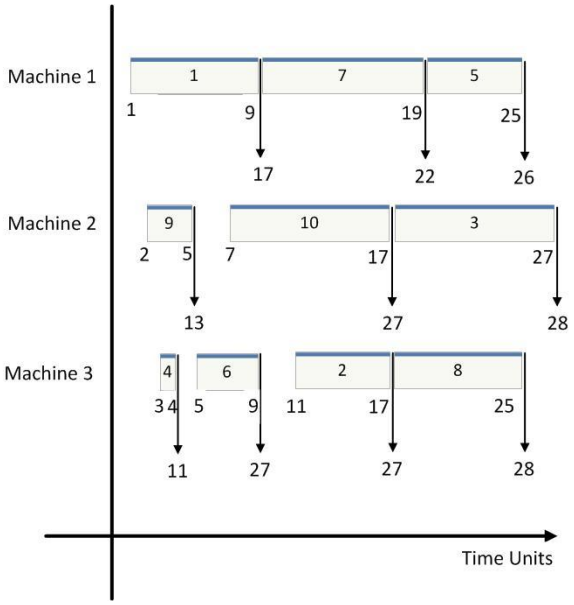


Figure 3: Gantt chart of an optimal schedule of $P_m /r_j,q_j/C_{max}$ *for example 2*

Observing that the obtained schedule has idle times:

- 2 units between jobs 9 and 10,
- *1 unit between jobs 4 and 6,*
- 2 units between jobs 6 and 2.

The ***right shifting*** procedure yields the following schedule (Figure 4), with makespan equal to 29:

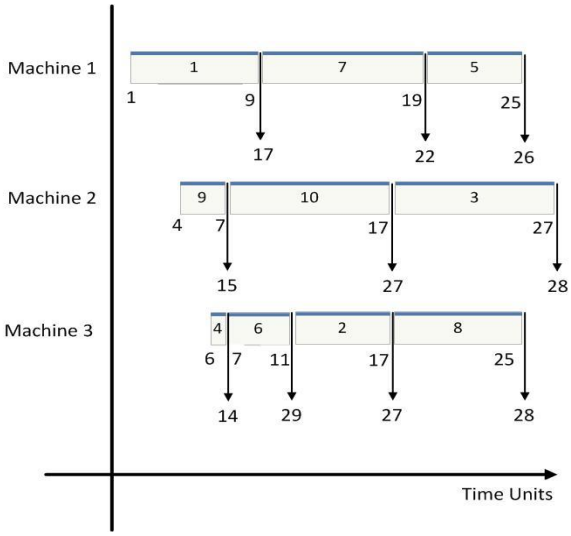


Figure 4: feasible schedule obtained after the ***Right shifting*** procedure

For the improvement phase (**Phase 2**), observing that:

- $J_1 = \{1,2,8\}$ and $C_1=26$,
- $J_2 = \{9,10,3\}$ and $C_2=28$,
- $J_3 = \{4,6,7,5\}$ and $C_3=29$.

Solving the two parallel machine problem $P_2 / r_j, q_j / C_{max}$ on the machines M_3 and M_1, with jobs subset $JN = \{4,6,7,5,1,2,8\}$. The resulting schedule is presented in the Figure 5.
Since the last schedule has a makespan $UB_{opt\text{-}opt} = 28 = LB$, then it is optimal.

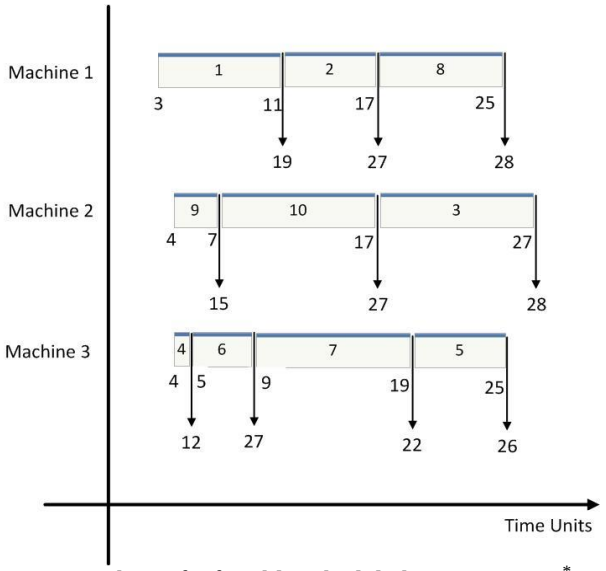


Figure 5: Gantt chart of a feasible schedule having $LB = C^*_{max} = 28$.

3.2.2 Heuristic $H_{Schrage\text{-}Schrage}$

Recall that the Schrage algorithm is developed to provide a feasible solution for the $P_m /r_j,q_j/C_{max}$ scheduling problem. It consists on scheduling among the unscheduled jobs, on the first available machine, the job with the largest delivery time (q_j). In this section we propose to use a modified version of the Schrage algorithm, during the first and the second phase. The modified version consists on scheduling the job with largest delivery time on one of the available machines such that the makespan for the scheduled jobs is minimized. Using the modified version of Schrage algorithm as it was done for the $H_{opt\text{-}opt}$ for Phase 1 and Phase 2 will provide us with an heuristic denoted $H_{Schrage\text{-}Schrage}$ and the obtained makespan is referred to as $UB_{Schrage\text{-}Schrage}$

3.2.3 Heuristics $H_{Schrage\text{-}opt}$ and $H_{opt\text{-}Schrage}$

The two last heuristics are a combination of the Schrage algorithm and the exact resolution algorithm. $H_{Schrage\text{-}opt}$ is set for the one where we use for the **Phase 1** the Schrage algorithm and for the **Phase 2** the exact algorithm**.** If we use the exact algorithm in **Phase 1** and the Schrage algorithm in **Phase 2**, the obtained heuristic is denoted $H_{opt\text{-}Schrage}$. The obtained makespan are $UB_{Schrage\text{-}opt}$ and $UB_{opt\text{-}Schrage}$ for the $H_{Schrage\text{-}opt}$ and $H_{opt\text{-}Schrage}$ heuristics, respectively.

4 Preliminary computational results

4.1 Test problems

The performance of the proposed lower bound and heuristics has been assessed through experimental tests over the test problems that are generated as in [4]. More precisely, the number of jobs $n \in \{10, 20, 40, 50, 200\}$. The number of machines $m \in \{2, 3, 5, 8\}$. The processing times $p_j \in U[1, 10]$, heads and tails r_j , $q_j \in U[1, Kn/m]$, with $K \in \{1, 3, 5, 7\}$. For each combination of n, m and K, 10 instances are generated. All the procedures were coded in C and implemented in Visual C++ 6.0 on a Pentium IV 3.2 GHz Personal Computer with 1.5 GB RAM. We report in Table 3-4 the results with:

- *LB* (Time): average time needed to compute *LB*.
- *RG = 100(UB-LB)/LB*: the relative gap.
- *Gap*: the average relative gap.
- *Time*: average time for the heuristic.
- *Iter*: average number of iterations in **Phase 2**.

4.2 Numerical results

n	*m*	LB (Time)	*UBOpt-Opt*			*UBOpt-Schrage*			*UBSchrage-Opt*			*UBSchrage-Schrage*		
			Gap	Time	Iter	Gap	Time	Iter	Gap	Time	Iter	Gap	Time	Iter
10	2	0	0	0.01	0	0	0	0	0	0.01	1	5.15	0.01	1
20	2	0	1.49	0.01	0.1	0.71	0.01	0.2	1.06	0.02	1	9.94	0.02	1
40	2	0.02	0.15	0.03	0.1	0.02	0.02	1	1.27	0.04	1	9.32	0.02	1
50	2	0.01	0.72	0.03	0.3	0.48	0.03	1	1.56	0.04	1	9.88	0.03	1
200	2	1.1	0.45	1.31	0.3	0.05	1.26	1	0.05	4.77	1	12.44	0.19	1.2
10	3	0	1.36	0.01	0.4	0	0.01	1	0.77	0.01	1.3	2.91	0.01	1.5
20	3	0	1.55	0.03	0.6	0.23	0.02	1	0.11	0.03	1.9	5.85	0.03	1.4
40	3	0.01	0.12	0.03	0.2	0	0.02	1	3.71	0.04	1.5	6.22	0.03	1.4
50	3	0.01	1.27	0.06	0.8	0.46	0.05	1.5	1.85	0.06	1.5	5.66	0.07	3.6
200	3	2.48	0.31	2.72	0.6	0	2.64	2.5	0.12	1.63	2.2	8.78	0.24	2.3
10	5	0	2.3	0.02	0.8	0	0.01	4.4	1.05	0.02	3.3	3.33	0.01	3.1
20	5	0	0.35	0.03	0.4	0	0.01	3	0.72	0.03	1.4	2.52	0.03	1.1
40	5	0	0.74	0.05	0.8	0.51	0.04	2.9	0.85	0.07	3.9	4.15	0.07	4.8
50	5	0	0.7	0.06	0.8	0.3	0.06	5.5	0.61	0.06	3.4	4.2	0.05	2.2
200	5	1.28	1.11	1.51	1.4	0.79	1.55	2.7	1.4	0.37	3.1	5.53	0.32	2.4
10	8	0	0	0.01	0	0	0.01	9	0	0.01	1.9	0.67	0.01	1
20	8	0.01	0.06	0.05	1.4	0	0.04	6.6	0.5	0.06	1.5	2	0.06	1.7
40	8	0	1.43	0.1	2.8	0	0.14	2.4	0.33	0.06	9.2	1.47	0.06	1.7
50	8	0	0	0.05	0.1	0	0.03	1	0.25	0.11	4.9	2.35	0.07	5.2
200	8	1.25	0.66	1.59	2.2	0	1.62	2.8	0.25	0.71	7.6	1.69	0.42	6.4

UBOpt-Opt			*UBOpt-Schrage*			*UBSchrage-Opt*			*UBSchrage-Schrage*		
Gap	Time	Iter	Gap	Time	Iter	Gap	Time	Iter	Gap	Time	Iter
0.7385	0.3855	0.705	0.1775	0.3785	2.525	0.823	0.4075	2.68	5.203	0.0875	2.25

Table 3: Detailed numerical results

According to the obtained results, the $UB_{Opt\text{-}Schrage}$ obtained is the best combination in term of gap and time. In addition, the gap for this heuristic is 0.177 which is a proof of the reasonable performance of the proposed procedures.

4 Conclusion and future directions

In this paper we addressed the parallel processors scheduling problem with release date, delivery time and with no-idle time constraint. A family of a two phases' heuristics is developed and a new lower bound is proposed. An extensive experimental study proofs the efficiency of the proposed procedures. As a future direction, the proposed procedures will be integrated to an exact Branch and Bound algorithm, in order to solve optimally the studied scheduling problem.

Acknowledgements The authors would like to gratefully acknowledge the support for this work provided by the Research Centre in the College of Computer & Information Sciences (CCIS) under project number: RC131038, as well as the Deanship of Scientific Research at King Saud University.

References

1. Amdahl, G, The validity of the single processor approach to achieving large-scale computing capabilities, AFIPS Press, coll, In Proceedings of AFIPS Spring Joint Computer Conference, Atlantic City, N.J, 483–85 (1967)
2. Bernstein, A. J, Program Analysis for Parallel Processing, IEEE Trans. on Electronic Computers .EC-15, 757–62 (1966).
3. Carlier J., Hermès F., Moukrim A., Ghédira K, Exact resolution of the one-machine sequencing problem with no machine idle time, Computers & Industrial Engineering, 59, 193-199 (2010)
4. Carlier J, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European Journal of Operational Research, 29, 298-306 (1987)
5. Culler, D.E., Jaswinder P.S., and Anoop G, Parallel Computer Architecture - A Hardware/Software Approach, Morgan Kaufmann Publishers (1999)
6. Emrah B. E, Oguz C., OzkarahanI, Parallel machine scheduling with additional resources: Notation, classification, models and solution methods, European Journal of Operational Research, 230, 449-463 (2013)
7. Gharbi A., Haouari M., Minimizing Makespan on Parallel Machines Subject to Release Dates and Delivery TimesJournal of Scheduling, 5, 329-355 (2002)
8. Garey M.R., Johnson D.S, Computers and Intractability :A Guide to the Theory of NP-Completeness, Freeman (1979)
9. Graham R.L., Lawler E.L., Lenstra J.K., RinnooyKan A.H.G, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 5, 287-326 (1979)
10. Hao W, Xudong F, Guangqian W, Tiejian L, Jie G, A common parallel computing framework for modeling hydrological processes of river basins, Parallel Computing, 37, 302-315 (2011)
11. Johnson E.A., Proppe C., Spencer Jr. B.F., Bergman L.A., Székely G.S., Schuëller G.I, Parallel computing in experimental mechanics and optical measurement: A review , Probabilistic Engineering Mechanics, 18, 37-60 (2003)
12. Jouglet A, Single-machine scheduling with no idle time and release dates to minimize a regular criterion, J Sched, 15, 217–238 (2012)
13. Kacem I., Kellerer H, Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times", Discrete Applied Mathematics, , (2011)

14. Roosta, Seyed H."Parallel processing and parallel algorithms: theory and computation", Springer, 114 (2000)
15. Wenjing G, Qian K, Parallel computing in experimental mechanics and optical measurement: A review, Optics and Lasers in Engineering, 50, 608-617(2012)
16. Veltman B., Lageweg B.J., Lenstra J.K, Multiprocessor scheduling with communication delays, Parallel Computing, 16, 173-182 (1990)
17. Donald S.,Martin W.,Neal K. ,Min Y.,Grant W.,Gopal N., High Performance Green Computing, ACM, 2007.
18. Bertossi A.A., FusielloA, Rate-monotonic scheduling for hard-real-time systems, European Journal of Operational Research, 96, 429-443 (1997)
19. Lawler E.L., Lenstra J.K., RinnooyKan A.H.G., Shmoys D, Sequencing and scheduling: Algorithms and Complexity, Handbooks in Operations Research and Management Science 4, S.S. Graves, A.H.G. RinnooyKan, P. Zipkin (eds.), 445-522(1993)
20. Kostadis R., Nawaf B., Robert E, Deterministic Batch Scheduling without Static Partitioning, Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'99Workshop, JSSPP'99, San Juan, Puerto Rico, April 16, ,Proceedings, 220-237 (1999)

MISTA 2015

Lower bounds for the parallel processing scheduling problem with multiprocessor tasks, release date and delivery time

Lotfi Hidri • Belgacem Ben Youssef • Achraf Gazdar

Abstract: In this paper, we consider the multiprocessor-task scheduling problem in a parallel processing system. The tasks are characterized by a release date (or arrival time), processing time, and delivery time. In addition, a task requires more than one processor to be processed in parallel. The objective of this research is to provide a feasible schedule that minimizes the completion time, or makespan, of the last treated task. Multiprocessor task scheduling plays a fundamental role in the performance of parallel and distributed computing systems. In this regard, we propose a family of new tight lower bounds. Finally, we present results related to the assessment of the efficiency of the proposed lower bounds after undertaking extensive computational experiments.

1 Introduction

During the last two decades, parallel computing has received a lot of attention from the computational science and engineering community. This has been due to the stunning technological advances that provided ever more precise, fast, and cheap processors. On the other hand, the use of parallel computing systems in the form of multicore and many-core architectures has become widespread. This wide adoption and extensive use in several fields has resulted in the further acceleration of technological and scientific advances in areas such as mechanical engineering, aerospace engineering, mathematical optimization, civil engineering, medicine, biology, chemistry, scientific computing, transportation, management, etc. ([11], [13], [22]).

One of the main concepts of parallel computing is to divide a complex problem into many small parts. These parts can be classified into two sets: One set containing the parts that can be executed in parallel while the other set contains those parts that are related by precedence

Lotfi Hidri
Department of Industrial Engineering
College of Engineering
King Saud University
E-mail: lhidri@ksu.edu.sa

Belgacem Ben Youssef
Department of Computer Engineering
College of Computer & Information Sciences
King Saud University
E-mail: bbenyoussef@ksu.edu.sa

Achraf Gazdar
Department of Software Engineering
College of Computer & Information Systems
King Saud University
E-mail: agazdar@ksu.edu.sa

constraints. The parallel parts are processed using multiple processors concurrently yielding fast execution time and high performance speedup ([2], [5], [7], [19]).

Task scheduling is the assignment of tasks or jobs to the available resources in a system, in order to obtain optimum performance. When multiple processors are used, the objective is to achieve a linear speedup in performance and maximum efficiency. Unfortunately, this is not the case for most problems because of several factors including communication overhead, control overhead, and precedence constraints between tasks [31]. It follows that improving the efficiency and the performance of parallel processing systems requires the development of efficient task scheduling techniques [32]. While several heuristic algorithms with effective solutions have been published, these techniques are limited to small size instances [33].

The concept of a multiprocessor task involves the idea that each task requires more than one processor for execution at the same time ([6], [15], [14], [18], [20]). In addition, a complex problem that will be executed using parallel processors is divided itself into two sets of tasks. The first one contains the parallel processed tasks and the second one contains the tasks subject to precedence constraints. A precedence constraint between two tasks means that some information must be communicated from one task to the other in order for the processing of this task to progress and continue forward. Thus, a communication delay appears in this situation and must be considered when trying to solve the problem at hand ([12], [3], [18], [4]).

In this research work, we are interested in the scheduling of multiprocessor tasks in a parallel computing environment whereby these tasks have a release date, and delivery time. The resolution of the current scheduling problem could provide new computational methods and techniques to solve other related scheduling problems, such as the Hybrid Flow Shop Scheduling problem with Multiprocessor Tasks (HFSMT) [34, 35, 36]. The HFMST is defined as follows: A set of K stages $Z_1,\ldots, Z_K$ containing respectively $m_1,\ldots, m_K$ ($\max(m_1,\ldots, m_K) > 1$) identical parallel processors, has to process a set $J = \{1,..., n\}$ of n tasks in the following way. Each task $j \epsilon J$ is treated over $Z_1,\ldots, Z_K$ in that order, during $p_{1j},\ldots, p_{Kj}$ units of time and using $size_{1j},..., size_{Kj}$ parallel processors, respectively. The processors treat the tasks without preemption with the objective of building a feasible schedule that minimizes the makespan C_{max} or the completion time of the last treated task on Z_K. The restriction to each stage generates K multiprocessor task scheduling problems. The consecutive resolution of these K problems yields a feasible solution for the HFSMT. In general, a resolution of the presented scheduling problem is suitable for any type of parallel architecture containing multiprocessor tasks.

The current state of the art in the literature indicates the presence, however, of some relaxation assumptions that were studied and some heuristic solutions being provided as described in [1, 4, 16, 17, 21, 23, 24, 25, 26, 27, 28, 29, 30]. Therefore, our research work consists of providing efficient computational procedures that solve this scheduling problem of $P_m/ r_j, q_j, size_j /C_{max}$. These procedures will be some lower bounds. The assessment of the efficiency of the proposed procedures will be undertaken over extensive computational experiments.

The paper is organized as follows: in Section 2, we briefly define the treated scheduling problem. In Section 3, a family of new lower bounds is proposed. Section 4 is dedicated to the presentation of the results of an extensive experimental analysis of the different lower bounds. Finally, we present some directions for future research.

2 Problem definition

The addressed scheduling problem in this paper is the parallel processors scheduling problem with release date, delivery time, and multiprocessor tasks (PMT), which can be stated as follows. We are given a set $J = \{1, 2,\ldots,n\}$ of n tasks that have to be processed on m identical parallel processors, denoted by M_i for $(i = 1, 2,\ldots,m)$. In addition, each task $j \epsilon J$ is characterized by these parameters:

- $size_j$: The number of processors required to treat the task j in parallel (ie, at the same time).
- r_j: A release date upon which task j is ready to be processed.
- p_j: A processing time for task j.
- q_j: A delivery time that elapsed between the completion of the processing and the exiting of the system.

Moreover, the processing of the tasks is done under the following constraints:

- For all $j \in J$, r_j , p_j and q_j are assumed to be integral and deterministic.
- All the processors are ready to process from the time zero.
- Preemption is not allowed during the processing of each job $j \in J$.
- Each processor treats at most one task at a given time.

Our objective is to find a feasible task schedule, which minimizes the completion time of the last treated job, C_{max}, also known as the makespan. Following Graham's notation in [10], this problem is denoted as: $P_m/ r_j, q_j, size_j /C_{max}$. It is known to be NP-Hard in the strong sense since its relaxation $P_m/r_j, q_j /C_{max}$ is NP-Hard in the strong sense ([8], [9]). The $P_m/ r_j, q_j, size_j /C_{max}$ scheduling problem is of practical interest since it models realistically encountered scenarios in parallel processing. The precedence constraints induce a release date, which is the earliest date to process the task, and delivery time for each task [37]. This scheduling problem is prevalent in the application of parallel computing in several areas of study such as image and video processing [38] and linear algebra [24], to name just a few. In the following we present an example of a feasible schedule.

Example 1: Consider the following instance of $m = 6$ and $n = 5$, with the processing times, release dates and delivery times being displayed in Table 1.

j	1	2	3	4	5
r_j	0	2	5	7	8
p_j	3	3	3	3	6
q_j	4	8	11	1	3
$size_j$	6	5	4	3	3

Table 1: Data for example 1

And a corresponding feasible schedule with $C_{max}=20$ is presented over Figure 1, where the completion time of each job is presented at the tip of the corresponding arrow.

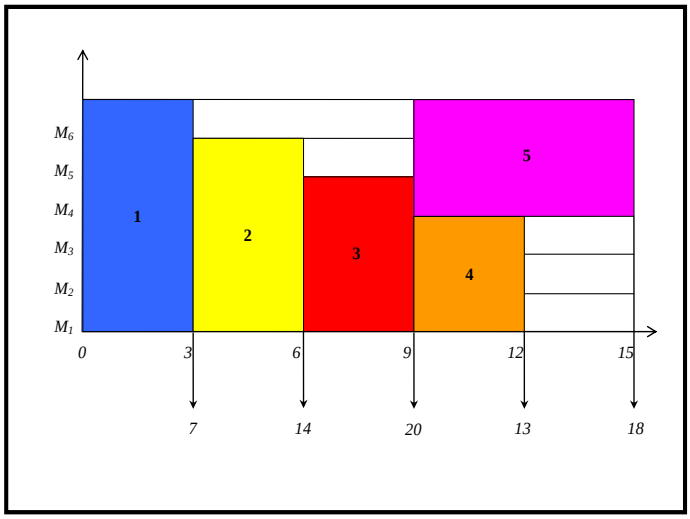


Figure 1: Gantt chart of feasible schedule of $P_m/ r_j, q_j, size_j /C_{max}$ for *example 1*

3 Lower bounds

2.1 Trivial lower bound

A trivial lower bound is presented over the following lemma.

Lemma 1: A valid lower bound is:

$$LB_0 = \max_{j \in J} \left(r_j + p_j + q_j \right) \tag{1}$$

Proof: Any job *j* has to spend at least $r_j+p_j+q_j$ units of time before completion. Thus, we get the result. In addition, LB_0 can be computed in $O(n)$ time.

2.2 Simple lower bound

Lemma 2: A simple and valid lower bound is:

$$LB_1 = \min_{j \in J} r_j + \left| \frac{\sum_{j \in J} size_j p_j}{m} \right| + \min_{j \in J} q_j \tag{2}$$

Proof: Relaxing the starting time of all the jobs to the minimum release date, relaxing the delivery times to the minimum one and allowing the preemption give us the result.

2.3 Subset-based lower bound

Interestingly, if the set of jobs is restricted to a particular subset, we can derive a lower bound. The following proposition presents such a lower bound.

Proposition 1: A valid lower bound for the current scheduling problem is:

$$LB_2 = \min_{j \in A \cup B} r_j + \sum_{j \in A} size_j p_j + \left| \frac{1}{2} \sum_{j \in B} p_j \right| + \min_{j \in A \cup B} q_j \tag{3}$$

Where

$$A = \left\{ j \in J : size_j > \frac{m}{2} \right\}$$

$$B = \left\{ j \in J : size_j = \frac{m}{2} \right\}$$

Proof: Observing that $A \cap B = \varnothing$, then jobs from *A* and *B* can not be treated within the same time interval. Each job *j* in *A* is processed lonely during p_j. In addition, allowing the preemption for the jobs in *B* gives a load equal to:

$$\left| \frac{1}{m}\sum_{j\in B} size_j p_j \right| = \left| \frac{1}{m}\sum_{j\in B} \frac{m}{2} p_j \right| = \left| \frac{1}{2}\sum_{j\in B} p_j \right| ..$$

The remaining part of the proof is obtained by relaxing the starting times and the delivery times to $\min_{j\in A\cup B} r_j$ and $\min_{j\in A\cup B} q_j$, respectively.

2.4 Splitting-based lower bound

This lower bound is based on the concept of splitting relaxation, which consists on subdividing a job *j* into a set $A_j = \left\{ j_1, j_2, ..., j_{size_j} \right\}$ composed of $size_j$ jobs. Each job j_i *(i =1,.., $size_j$)* having the following characteristics: $r_{j_i} = r_j$, $p_{j_i} = p_j$, $q_{j_i} = q_j$ and $size_{j_i} = 1$. Therefore, the obtained problem *(P)* will be a parallel processors scheduling problem with release date and delivery time, where the jobs set is $\bigcup_{j\in J} A_j$. Interestingly, we have the following result.

Lemma 3: Any lower bound for *(P)* is a lower bound for $P_m / r_j, q_j, size_j / C_{max}$.

Proof: An optimal schedule for $P_m / r_j, q_j, size_j / C_{max}$ with optimal value C^A_{max} is a feasible schedule for *(P)*. Let C^*_{max} be the optimal value for *(P)*, then $C^*_{max} \leq C^A_{max}$. For any lower bound *LB* for *(P)*, we have $LB \leq C^*_{max} \leq C^A_{max}$. Thus, *LB* is a valid lower bound for $P_m / r_j, q_j, size_j / C_{max}$.

Remark: The considered splitting based lower bound is:

$$LB_3 = C^*_{max}(P), \qquad (4)$$

where the optimal value $C^*_{max}(P)$ is obtained using the algorithm in [9].

2.5 Revisiting energy-based reasoning lower bound

The concept of energy-based reasoning (ER) relies on the computation of the part of the jobs that must be processed within the time interval $[t_1; t_2]$, in any feasible schedule. This part is called the work or the mandatory part of the job *j* over $[t_1; t_2]$. In order to determine this work, a job *j* starts at it is release date r_j or finishes at it is due date d_j. In other terms the job j is the *left-shifted* (r_j) or *right-shifted* (d_j) on their time window $[r_j; d_j]$. The *left-work* of a job *j* over $[t_1; t_2]$, denoted by $W^l_j(t_1; t_2)$, is defined as the length of $[t_1, t_2] \cap \lfloor r_j, r_j + p_j \rfloor$. Symmetrically, the *right-work* of a job *j* over $[t_1; t_2]$,, denoted by $W^r_j(t_1; t_2)$, is defined as the length of $[t_1, t_2] \cap \lfloor d_j - p_j, r_j \rfloor$. Thus, the work of a job *j* over $[t_1; t_2]$; denoted by $W_j(t_1; t_2)$, is equal to the minimum between $W^l_j(t_1; t_2)$, and $W^r_j(t_1; t_2)$. In other terms we have:

- $W^l_j(t_1, t) = size_j \times \min\left(t_2 - t_1, p_j, \max\left(0, r_j + p_j - t_1\right)\right)$,
- $W^r_j(t_1, t) = size_j \times \min\left(t_2 - t_1, p_j, \max\left(0, t_2 - d_j + p_j\right)\right)$,
- $W_j(t_1, t) = \min\left(W^l_j(t_1, t), W^r_j(t_1, t)\right)$

- $W(t_1,t_2) = \sum_{j \in J} W_j(t_1,t_2)$ this is the total work within *[t_1; t_2]*.

Obviously, if $W(t_1,t_2) > m(t_2 - t_1)$, the instance is infeasible. Moreover, the slack of job *j* is defined by: $s_j(t_1,t_2) = m(t_2 - t_1) - \left(W(t_1,t_2) + W_j(t_1,t_2)\right)$, then r_j and d_j may be adjusted as follows.

- If $s_j(t_1,t_2) < W_j^l(t_1,t_2)$ then $r_j := \max\left(r_j, t_2 - \left\lfloor \frac{s_j(t_1,t_2)}{size_j} \right\rfloor\right)$
- If $s_j(t_1,t_2) < W_j^r(t_1,t_2)$ then $d_j := \max\left(d_j, t_1 + \left\lfloor \frac{s_j(t_1,t_2)}{size_j} \right\rfloor\right)$

The authors of [39] proved that only $O(n^2)$ relevant values of t_1 and t_2 need to be considered. More recently, authors in [41] observed that in any feasible schedule, at times t_1 and t_2 there is no more than m jobs that can be placed. Thus an improvement for the computation of the total work is performed for the parallel processors scheduling problem, this is the Revisited Energetic Reasoning (RER). The ERE is extended to the parallel processors with multiprocessor tasks and the minimum total work is the optimal solution of the following integer linear program.

$$
\begin{aligned}
& W_{RER}(t_1,t_2) = \min \sum_{i=1}^{n} W_i^l(t_1,t_2)\, x_i + \sum_{i=1}^{n} W_i^r(t_1,t_2)\, y_i + \sum_{i=1}^{n} p_i z_i \\
& St \\
& x_i + y_i + z_i = 1 \quad (i = 1,..,n) \\
& \sum_{i \in J;\ r_i \le t_1} size_i \times x_i \le m \\
& \sum_{i \in J;\ t_2 \le d_i} size_i \times y_i \le m \\
& x_i, y_i, z_i \in \{0,1\}
\end{aligned}
\tag{5}
$$

where the decision variables x_i , y_i and z_i are defined as:

$$
\begin{aligned}
x_i &= \begin{cases} 1 \text{ if job } i \text{ is placed left } t_1 \\ 0 \text{ otherwise} \end{cases} \\
y_i &= \begin{cases} 1 \text{ if job } i \text{ is placed right } t_2 \\ 0 \text{ otherwise} \end{cases} \\
z_i &= \begin{cases} 1 \text{ if job } i \text{ is placed inside } [t_1,t_2] \\ 0 \text{ otherwise} \end{cases}
\end{aligned}
\tag{7}
$$

Next, the RER-based lower bound is performed by starting with an initial simple lower bound *LB*. After that, the due dates are setup to: $d_j = LB - q_j$ and the RER is applied, if an infeasibility is detected ($W_{RER}(t_1,t) > m(t_2 - t_1)$) then *LB* is incremented (*LB:= LB+1*). The procedure is halted if there is no infeasibility. The obtained lower bound is denoted LB_5. In addition, we denote:

$$LB_{all} = \max_{i=0,\ldots,4} \left(LB_i \right) \quad (5)$$

3 Preliminary computational results

3.1 Test problems

The performance of the proposed lower bound and heuristics has been assessed through experimental tests over the test problems that are generated as in [40]. More precisely,

- The number of jobs $n \in \{10, 20, 40, 50, 200\}$.
- The number of machines $m \in \{2, 3, 5, 8\}$.
- The processing times $p_j \in U[1, 10]$,
- The heads and tails r_j , $q_j \in U[1, Kn/m]$, with $K \in \{1, 3, 5, 7\}$.
- The required number of processors to treat the jobs $size_j$ is generated uniformly from $[1, m]$.

For each combination of n, m and K, 10 instances are generated. All the procedures were coded in C and implemented in Visual C++ 6.0 on a Pentium IV 3.2 GHz Personal Computer with 1.5 GB RAM.

3.2 Numerical results

We report in Table 2 the obtained results:

- $\%\ LB_{all}$: the percentage the $LB_i = LB_{all}$.
- Time (second): average time for the lower bound.

LB_i	LB_0	LB_1	LB_2	LB_3	LB_4
$\%\ LB_{all}$	*0%*	*0%*	*35%*	*79,5%*	*100%*
Time	$<10^{-9}$	$<10^{-9}$	$<10^{-9}$	*3.89*	*23.56*

Table 2: Summary of numerical results

According to the obtained results, we observe that the RER (LB_4) performs well in terms of reaching the maximum for all 10 instances. However the required computational time is relatively high. In addition, the splitting-based lower bound (LB_3) yields overall acceptable results both in terms of time and lower bound performance, the latter as given by $\%\ LB_{all}$. Thus, the results yielded by LB_3 display a balance between the required computational time and the quality of the lower bound.

4 Conclusion and future directions

In this paper, we investigated some aspects of the multiprocessor task-scheduling problem with release date and delivery time. In particular, a family of new lower bounds has been developed. Using a suite of benchmark tests, our numerical experimentation results show that the proposed lower bounds are efficient and tight. As future directions of this work, we plan to propose several heuristics based on evolutionary computational techniques, such as genetic algorithms, in order to obtain a feasible solution. An exact Branch and Bound technique integrating the proposed lower bounds will be also explored.

Acknowledgements The authors would like to acknowledge the support for this work provided by the Research Centre in the College of Computer & Information Sciences (CCIS) under project number: RC140221, as well as the Deanship of Scientific Research at King Saud University.

References

1. A. V. Fishkin, G. Zhang, On maximizing the throughput of multiprocessor tasks , Theoretical Computer Science, 302, 319-335 (2003)
2. G. Amdahl, The validity of the single processor approach to achieving large-scale computing capabilities, In Proceedings of AFIPS Spring Joint Computer Conference, Atlantic City, NJ, AFIPS Press, 483–485 (1967)
3. A. Moukrim, Scheduling unitary task systems with zero-one communication delays for quasi-interval orders, Discrete Applied Mathematics, 127(3), 461-476 (2003)
4. A. Moukrim, E. Sanlaville, F. Guinand, Parallel machine scheduling with uncertain communication delays, RAIRO - Operations Research, 37(1), 1-16 (2003)
5. A. J. Bernstein, Program analysis for parallel processing, IEEE Transactions on Electronic Computers, EC-15, 757–762 (October 1966)
6. R. C. Correa, A. Ferreira, P. Rebreyend, Scheduling multiprocessor tasks with genetic algorithms, IEEE Transactions on Parallel and Distributed Systems, (1999), 10(8), 825–37 (1999)
7. D. E. Culler, P. S. Jaswinder, G. Anoop, Parallel computer architecture: A hardware/software approach, Morgan Kaufmann Publishers, San Francisco (1999)
8. M. R. Garey, D. S. Johnson, Computers and intractability: A Guide to the theory of NP-Completeness, W. H. Freeman, New York (1979)
9. A. Gharbi, M. Haouari, Minimizing makespan on parallel machines subject to release sates and delivery times, Journal of Scheduling, 5, 329-355 (2002)
10. R. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics, 5, 287-326 (1979)
11. W. Hao, F. Xudong, W. Guangqian, L. Tiejian, G. Jie, A common parallel computing framework for modeling hydrological processes of river basins, Parallel Computing, 37, 302-315 (2011)
12. J. J. Hwang, Y-C. Chow, F. D. Anger, C-Y. Lee, Scheduling precedence graphs in systems with inter-processor communication times, SIAM Journal on Computing, 8, 244–58 (1989)
13. E. A. Johnson, C. Proppe, B. F. Spencer Jr., L. A. Bergman, G. S. Székely, G. I. Schuëller, Parallel computing in experimental mechanics and optical measurement: A review, Probabilistic Engineering Mechanics, 18, 37-60 (2003)
14. K.-H. Yang, P. S. Pulat, Y. Guan, Embedded simulation on a multiprocessor job scheduling system with inspection, Computers & Industrial Engineering, 57, 592-607 (2009)
15. M. Drozdowski, Scheduling multiprocessor tasks — An overview, European Journal of Operational Research, 94, 215-230 (1996)
16. P. Dell'Olmo, A. Iovanella, G. Lulli, B. Scoppola, Exploiting incomplete information to manage multiprocessor tasks with variable arrival rates, Computers & Operations Research, 35, 1589-1600 (2008)
17. P. Baptiste, A note on scheduling multiprocessor tasks with identical processing times, Computers & Operations Research, 30, 2071-2078 (2003)
18. R. Hwanga, M. Genb, H. Katayama, A comparison of multiprocessor task scheduling algorithms with communication costs, Computers & Operations Research, 35, 976–993 (2008)
19. S. H. Roosta, Parallel processing and parallel algorithms: Theory and computation, p. 114, Springer, New York (2000)
20. T. Thanalapati, S. Dandamudi, An efficient adaptive scheduling scheme for distributed memory multicomputers, IEEE Transactions on Parallel and Distributed Systems, 12(7), 758–68 (2001)

21. B. Veltman, B. J. Lageweg, J. K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Computing, 16, 173-182 (1990)
22. G. Wenjing, K. Qian, Parallel computing in experimental mechanics and optical measurement: A review, Optics and Lasers in Engineering, 50, 608-617 (2012)
23. M. Drozdowski**,** Scheduling for parallel processing, Springer-Verlag, London (2009)
24. S. Jin, G. Schiavone, D. Turgut, A performance study of multiprocessor task scheduling algorithms, Journal of Supercomputing, 43, 77-97 (2008)
25. A. Zaki, S. Shahul, O. Sinnen, Scheduling task graphs optimally with A*, Journal of Supercomputing, 51, 310-332 (2010)
26. M. Abdeyazdan, S. Parsa, A. M. Rahmani , Task graph pre-scheduling, using Nash equilibrium in game theory, Journal of Supercomputing, 64, 177-203 (2013)
27. G. Wei, A. V. Vasilakos, Y. Zheng, A game-theoretic method of fair resource allocation for cloud computing services, Journal of Supercomputing, 54, 252-269 (2010)
28. V. Bonifaci, A. Marchetti-Spaccamela, Feasibility analysis of sporadic real-time multiprocessor task systems, Algorithmica, 63, 763-780 (2012)
29. M. E. Moghaddam, An Immune-based genetic algorithm with reduced search space coding for multiprocessor task scheduling problem, International Journal of Parallel Programming, 40, 225-257 (2012)
30. P. Regnier, G. Lima, E. Massa, G. Levin, S. Brandt, Multiprocessor scheduling by reduction to uniprocessor: An original optimal approach, Real-Time Systems, 49, 436-474 (2013)
31. Y. C. Lee, A. Y. Zomaya, Immune system support for scheduling, Advances in Applied Self-Organizing Systems, Self-Organizing Computation, 247-270 (2008)
32. J. Liu, A-X. Zhu, C-Z. Qin, Estimation of theoretical maximum speedup ratio for parallel computing of grid-based distributed hydrological models, Computers & Geosciences, 60, 58-62 (2013)
33. B. Macey, A. Zomaya, A performance evaluation of CP list scheduling heuristics for communication intensive task graphs, In Proceedings of Joint 12th International Parallel Processing Symposium and Ninth Symposium on Parallel and Distributed Programming, 538–541 (1998)
34. A. Lahimer, P. Lopez, M. Haouari, Improved bounds for hybrid flow shop scheduling with multiprocessor tasks, Computers & Industrial Engineering, 66, 1106-1114 (2013)
35. F-D. Chou, Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks, International Journal of Production Economics, 141, 137-145 (2013)
36. C. Kahraman, O. Engin, İ. Kaya, R. E. Öztürk, Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach, Applied Soft Computing, 10, 1293-1300 (2010)
37. R. Kunis, G. Rünger, Optimization of layer-based scheduling algorithms for mixed parallel applications with precedence constraints using move-blocks, In Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 70-77, (2009)
38. A. Merigot, A. Petrosino, Parallel processing for image and video processing: Issues and challenges, Parallel Computing, 34, 694-699 (2008)
39. P. Baptiste, C. Le Pape, W. Nuijten, Satisfiability tests and time bound adjustments for cumulative scheduling problems, Annals of Operations Research, 92, 305-333 (1999)
40. J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European Journal of Operational Research, 29, 298-306 (1987)
41. L. Hidri, A. Gharbi, M. Haouari, Energetic reasoning revisited: Application to parallel machine scheduling, Journal of Scheduling, 11, 239-252 (2008)

MISTA 2015

University Course Timetabling with Conflict Minimization and Elective Courses

A Decomposition-Based Approach to a Real-Life Case

Ernst Althaus · Udo Muttray

Abstract In this paper, we describe an integer programming approach to a real-life university timetabling case from Germany. Due to the special nature of the considered study program, a conflict-free timetable usually does not exist. Therefore, we propose a model to minimize the number of conflicts. In addition, decomposition is used to split the problem into a model for compulsory courses and a model for elective courses. Both models incorporate various additional hard constraints, such as the need for consecutive slots or even for desired conflicts across different semesters.

1 Introduction

1.1 Motivation

Educational timetabling poses a considerable challenge at numerous universities and schools. In addition to the computational complexity of the problem itself, the institutional models vary substantially, as pointed out by McCollum [13].

The problem considered in this paper deals with the scheduling of courses for students who want to become a teacher. At the University of Mainz, these students select 2 out of 21 eligible subjects and take courses in these two subjects as well as in educational sciences. If there is no central planning of the courses, students typically have conflicts, i.e. several of the courses they have to take in a semester are at the same time. Although it will not be possible to obtain a conflict-free timetable for each student, mathematical optimization can drastically reduce the number of conflicts.

Hence, the university opted for a global assignment of the courses from this particular study program a few years ago. The key ideas of the original model developed by Kreuzer [10] are the distinction between compulsory and elective courses and a fixed number of available slots per subject and semester. We improve upon this approach by suggesting a more detailed model, factoring in the actual number of required slots of courses as well as certain hard constraints. Furthermore, integer programming tech-

Ernst Althaus · Udo Muttray
Institute of Computer Science, Johannes Gutenberg University Mainz, Germany
E-mail: {ernst.althaus, udo.muttray}@uni-mainz.de

niques are used to optimize the model, while the previous (non-optimal) solution has been manually obtained.

As a global model with both compulsory and elective courses becomes intractable, we apply a decomposition-based approach. At first, compulsory courses are assigned such that the number of conflicts is minimized and some special constraints are satisfied. After that, the obtained timetables of the compulsory courses form the input data for a subsequent assignment of the elective courses.

1.2 Related Work

A great deal of previous research has been conducted on the various types of educational timetabling. Schaerf describes the common problem variants: university course timetabling, examination timetabling and school timetabling [16].

In most instances of university course timetabling, a conflict-free solution is available and consequently sought for. A recent overview is given by Bettinelli et al. [2].

Kiaer and Yellen develop a weighted graph model to describe problems which do not have conflict-free solutions and use a heuristic algorithm to solve them [9]. Those methods have been further refined by Wehrer and Yellen [19].

Another frequently neglected challenge is the existence of elective courses. Müller and Rudová report on the successful implementation of elective courses and course sections into the course timetabling system UniTime, using local search [15].

Regarding the methodology, according to Burke and Petrovic, at lot of research has focused on heuristic solution strategies [6]. However, with the growing capabilities of IP solvers over the last years, integer programming approaches have gained attention, as shown by Daskalaki et al. [8], MirHassani [14], Al-Yakoob and Sherali [1], Schimmelpfeng and Helber [17], Lach and Lübbecke [11] and Bonutti et al. [7].

Decomposition-based models constitute another common trend. They have effectively been used by Lach and Lübbecke to match courses to slots first, and courses/slots to rooms second. Other successful applications of decomposition-based models include the works form Burke et al. [5] and Sørensen and Dahms [18].

Our models incorporate both conflict-minimization and elective courses. Contrary to most previous approaches to such models, we use integer programming to solve them.

1.3 Organization of the Paper

This paper is organized as follows: In Section 2, we introduce the real-life timetabling problem from Germany. A decomposition-based integer programming model is presented in Section 3. In Section 4, we report computational results from real and artificial instances. In Section 5, we discuss alternative model formulations. Finally, Section 6 contains some conclusions and future research directions.

2 Problem Description

2.1 Conflict Minimization and Other Peculiarities

In comparison to most timetabling problems presented in the literature, our real-life case differs in four fundamental ways. Yet if those special features are considered, it usually does not happen in the context of integer programming techniques.

First, the timetabling instances come from a study program for students who aim at becoming a teacher. As such, they have to choose 2 out of 21 subjects and attend the respective courses. In addition, all students have to take courses from educational sciences. Preassigning courses from educational sciences, this is still roughly equivalent to a timetabling problem with 200 different curricula where each curriculum shares courses with 40 other curricula. In this situation, a conflict-free timetable can rarely be achieved. Instead, the objective must be to minimize the number of conflicts (weighted by the number of affected students).

The second difference is the existence of so called elective courses, which will be described in detail in the following section.

Third, courses may have special constraints, which might even link courses from different semesters. This requires the simultaneous construction of timetables for several semesters and is further explained in Section 2.5.

Finally, if not too many courses are scheduled simultaneously, in our case there is always an adequate number of rooms available. Therefore, the assignment of rooms is not considered. Note that this feature actually makes the problem easier to solve.

2.2 Slots and Course Types

Courses are taught on a weekly basis. To this end, the week is divided into 25 slots or time periods, five slots per working day, each slot two hours long. Courses have a certain number of required slots (usually between one and four).

Courses can be divided into two course types, *compulsory* courses and *elective* courses. Compulsory courses are offered only once a week, while elective courses are offered at least twice. The latter is often the case for identical tutorials offered multiple times in order to keep group sizes small or for advanced lectures (potentially covering different topics, but all equivalent for the purpose of examination regulations).

The important difference between the course types is that students must be able to attend all of the compulsory courses, but only one of at least two offers of each elective course. Notice that according to the policy of our university, all offers of an elective course are equivalent. Consequently, the possibility to select elective courses such that there is no conflict is sufficient. Students' preferences to a specific offer of an elective course are deliberately not taken into account as this would probably induce additional conflicts.

2.3 Assignment of Subjects and Courses

In the interest of students' ability to attend all required courses, the university gives priority to a minimal-conflict timetable over lecturers' preferences. For our objective functions, only an assignment of subjects to slots will be required (though the models

will be able to provide feasible assignments of courses as well). Therefore, the assignment of courses and lecturers to the preassigned subject specific slots is done afterwards by the departments of the subjects. This way, lecturers get some flexibility to customize their personal timetables.

Note that the described procedure is generally applicable, except for courses with special constraints (cf. Section 2.5).

2.4 Semesters

The year is divided into a winter semester and a summer semester. Students can begin their studies in both semesters, implying different curricula depending on the type of start semester. As the majority of students begins in a winter semester, only the curricula for students starting in a winter semester are currently considered. Besides, including curricula for both types of start semester would have the following consequence: All courses that are offered only once a year would now be part of the curricula for both start semester types and would therefore have to be assigned to specific slots, taking further flexibility from the departments. The associated political issue has not been finally decided, though our model can be extended to the general case.

Denoting the set of all winter semesters by $S_{\mathcal{W}}$ and the set of all summer semesters by $S_{\mathcal{S}}$, we can specify these sets. For a study program of 10 semesters and students beginning their studies in a winter semester, we have

$$S_{\mathcal{W}} = \{1, 3, 5, 7, 9\} \text{ and } S_{\mathcal{S}} = \{2, 4, 6, 8, 10\},$$

whereas for students beginning their studies in a summer semester we would have

$$S_{\mathcal{W}} = \{2, 4, 6, 8, 10\} \text{ and } S_{\mathcal{S}} = \{1, 3, 5, 7, 9\}.$$

2.5 Special Constraints

Around 20 % of all courses have special (hard) constraints, falling into one of the following categories:

1. *consecutive slots* (only if the course needs more than one slot),
2. *desired "conflicts" with other courses*, involving courses with the same course type from different subjects or semesters (w.l.o.g. all with the same number of required slots),
3. *no conflicts with other courses*, involving courses with the same course type from the same subject from different semesters,
4. *restrictions on the set of feasible slots*.

We give some examples to illustrate why the above constraints are important: Consecutive slots are usually required for practicals with long experiments that can not be done in one slot. Desired conflicts can occur when two or more subjects have a common course (e.g. "Introduction to Linguistics" in French, Italian and Spanish) or when a course is offered only once, but must be taken in multiple semesters (e.g. a choir in music with all students from semesters 1 – 4). No conflicts between two courses from one subject and different semesters can be required when the same courses are also to be attended by students from another study program. When those courses occur

during the same semester according to the curriculum of the other study program, they can not be placed in the same slot in our study program. Finally, some courses are instructed by external lecturers who can only adapt to specific slots.

2.6 Period of Validity of the Constructed Timetables

Since a conflict-free timetable does not exist for the presented problem, it is important for students to be able to plan ahead in case they encounter conflicting lectures. The same is true for students who exchange one of their subjects for another one during their course of studies, which is a very common phenomenon amongst students from the study program for future teachers. In other words, students' ability to plan ahead is preferred to an optimization with up to date input data. Therefore, a constructed timetable should be in use for several semesters, possibly subject to some local modifications due to subsequent changes to one of the curricula. Once a critical mass of changes to the input parameters has been reached, the timetable will be globally rebuilt.

3 Decomposition-based Integer Programming Model

3.1 General Considerations

In the beginning of this section, we discuss the decomposition approach and introduce notation. Sections 3.2 and 3.3 contain the models for compulsory and elective courses.

The main idea is a decomposition, assigning compulsory courses first and elective courses second. Both parts will be formulated and solved as integer programs. From a theoretical point of view, there is a chance that the first step produces a solution which leads to a (globally) non-optimal solution during the second step, as it is the case for all heuristic decomposition-based models. From a practical point of view, the suggested decomposition enables us to tackle an otherwise intractable problem. In our case, the disadvantages seem to be acceptable since there are usually a lot more compulsory courses than elective courses. Furthermore, elective course are sometimes offered more often than it is required by the model (cf. Section 3.3) which provides additional flexibility.

As the optimization problems for winter and summer semesters are independent, we solve the models for both compulsory and elective courses once for the winter semesters $S_{\mathcal{W}}$ and once for the summer semesters $S_{\mathcal{S}}$. On the other hand, all courses in a winter semester need to be dealt with simultaneously due to the second and third special constraint. The same statement applies for the courses in a summer semester. As a result, for each timetabling instance we need to perform four computations as shown in Figure 1.

All our timetabling instances are defined over the same sets, namely the set of subjects I, the set of slots K, the previously introduced sets of winter and summer semesters $S_{\mathcal{W}}$ and $S_{\mathcal{S}}$, the set of elective blocks T and various sets for courses X/E, $(X/E)^s$, $(X/E)^s_i$, X_* and X^s_*. To simplify the notation, we uniquely assign indices to these sets. Later on, we will only use the indices and omit the sets if the latter ones are obvious from the context.

Fig. 1: Decomposition based on semester type

Table 1: Sets and indices

Set	Index	Description
I	i, j	Set of subjects (excluding educational sciences),
	$*$	Subject "educational sciences",
$P \subset I$	(i, j)	Set of eligible subject combinations,
2^I	L	Set of all subsets of subjects,
K	k	Set of slots,
$S_{\mathcal{W}}/S_{\mathcal{S}}$	s	Set of winter/summer semesters,
T	t, u	Set of elective blocks,
X/E	c	Set of compulsory/elective courses from all semesters $s \in S_{\mathcal{W}}/S_{\mathcal{S}}$ and all subjects $i \in I$,
$(X/E)^s$	c	Set of compulsory/elective courses from semester s from all subjects $i \in I$ (i.e. $\forall\, c \in (X/E)^s : \sigma_c = s$),
$(X/E)_i^s$	c	Set of compulsory/elective courses from semester s from subject i (i.e. $\forall\, c \in (X/E)_i^s : \sigma_c = s$ and $\iota_s = i$),
X_*	c	Set of compulsory courses from all semesters $s \in S_{\mathcal{W}}/S_{\mathcal{S}}$ from educational sciences,
X_*^s	c	Set of compulsory courses from semester s from ed. sciences.

Table 2: Numerical and course specific data

Data	Description
θ_{ij}	Number of students with subject combination (i, j),
θ_L	Weighted number of conflicts for a set of subjects $L \subset I$,
ι_c	Subject of a course c,
σ_c	Semester of a course c according to the curriculum,
ρ_c	Number of required slots for a course c,
$\Phi_c \subset K$	Set of feasible slots for a course c,
C^α	Set of courses that must have consecutive slots,
$\{C_g^\beta\}_{g \in G^\beta}$	Sets of courses that must be assigned to the same slot, using index set G^β,
$\{C_g^\gamma\}_{g \in G^\gamma}$	Sets of courses that must be assigned to different slots, using index set G^γ.

In Table 1 we present the sets and indices. In addition to those sets we have some numerical and some course specific data as given in Table 2. Notice that for the sets introduced at first we always use capital letters, and for numerical and course specific data we use Greek letters. For indices, we use small letters (except for index L which represents a subset of I). Variables of our integer programming models will also have small letters and can be distinguished from indices as the variables have indices themselves.

Table 3: Variables for the compulsory courses

$h^s_{Lk} \in \{0,1\}$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ L \subset I : \lvert L\rvert \leq n,\ k \in K,$	
$h^s_{Lk} = 1$	$\Leftrightarrow$	in semester s, exactly the subjects in L are assigned to slot k,
$x_{ck} \in \{0,1\}$	$\forall\, c \in X \cup X_*,\ k \in K,$	
$x_{ck} = 1$	$\Leftrightarrow$	course c is assigned to slot k (the course determines the semester),
$a_{ck} \in \{0,1\}$	$\forall\, c \in (X \cup X_*) \cap C^\beta,\ k \in K,$	
$a_{ck} = 1$	$\Leftrightarrow$	consecutive slots of course c start in slot k.

3.2 Model for Compulsory Courses

The model for the compulsory courses can be seen as a problem of quadratic semi-assignment type, incorporating additional constraints as the special constraints from Section 2. Since the number of conflicts only depends on the subjects of the conflicting courses, one could formulate an objective function using variables $y^s_{ik} \in \{0,1\}$ to describe whether subject i is assigned to slot k in semester s or not. This was done in a similar fashion for examination timetabling by Laporte and Desroches [12] and Bullnheimer [4]. In our case, the objective function would be given by

$$\min \sum_{s,k} \sum_{i<j} \theta_{ij} \cdot y^s_{ik} \cdot y^s_{jk}.$$

Instead of linearizing the quadratic objective function, we introduce (exponentially many) variables $h^s_{Lk} \in \{0,1\}$ for each subset of subjects L, slot k and semester s, and link them to the respective course variables x_{ck}. Analogously, Cacchiani et al. propose models with exponentially many variables for the Udine Course Timetabling instances [3], which result in equally elegant expressions for the objective functions [7].

In addition to the variables for subject sets and courses, we need variables a_{ck} for the special consecutiveness constraint, indicating where a consecutive series of slots starts. All the variables are also listed in Table 3.

For the set variables, we made the heuristic assumption that no slot will have more than a given number n of different subjects. This assumption was supported by preliminary analysis without taking the special constraints into account, comparing weighted conflicts between both models. It will additionally contribute to the room distribution.

The refined objective function is minimizing the conflicts weighted by the number of affected students, using variables h^s_{Lk}:

$$\min \sum_{s,k} \sum_{L} \theta_L \cdot h^s_{Lk}.$$

For a subset of subjects $L \subset I$, the weighted number of conflicts θ_L is calculated as:

$$\theta_L := \sum_{\substack{i,j \in L \\ i<j}} \theta_{ij}.$$

To count the undesired conflicts, one would have to subtract the desired conflicts within a semester according to the second special constraint (this number is a constant):

$$\frac{1}{2} \sum_{g \in G_\beta} \sum_{\substack{c_1, c_2 \in C_g^\beta \cap X: \\ \sigma_{c_1} = \sigma_{c_2}}} \rho_{c_1} \cdot \theta_{\iota_{c_1} \iota_{c_2}}.$$

Finally, we present the various types of constraints for the compulsory courses. We start with the constraints involving the subset variables (and courses from educational sciences), move on to those involving the other course variables and conclude with the special constraints.

For each semester and slot, there can be either a subset of subjects or a course from educational sciences assigned to that slot:

$$\sum_L h_{Lk}^s + \sum_{c \in X_*^s} x_{ck} \leq 1 \quad \forall\, s,\ k. \tag{1}$$

There are $|K|$ slots for subsets of subjects and compulsory courses from educational sciences. Technically, this constraint is redundant in terms of describing a correct model as it is implied by the previous constraint and Constraint (6) for courses from educational sciences. Nonetheless, the following constraint did improve upon the runtime, which is why we include it (notice that further alternative formulations are discussed in Section 5):

$$\sum_{L,k} h_{Lk}^s + \sum_{c \in X_*^s} \rho_c = |K| \quad \forall\, s. \tag{2}$$

There are as many subsets containing a specific subject as are needed for the courses from that subject. This constraint is also technically redundant, because it can be obtained from the constraint for the slot requirements of the courses (6) and the linking constraint (7). We include it for the same reason as the above constraint:

$$\sum_{L: i \in L} \sum_k h_{Lk}^s = \sum_{c \in X_i^s} \rho_c \quad \forall\, s,\ i. \tag{3}$$

In order to guarantee that every combination is viable to study, no eligible combination should have more than one conflict, regardless of the number of affected students. This is considered to be the maximal acceptable number of conflicts for any student. Note that this constraint could potentially lead to infeasible models (this usually only happens for instances with a very low ratio of slots to subjects). In that case, it should be relaxed by incrementing the right-hand side by one. In addition, those constraints will be relaxed for subjects having courses with desired conflicts:

$$\sum_{L: i,j \in L} \sum_k h_{Lk}^s \leq 1 \quad \forall\, s,\ (i,j) \in P. \tag{4}$$

The following constraint is a model simplification, setting those subset variables to zero where courses from educational sciences are preassigned:

$$h_{Lk}^s = 0 \quad \forall\, s,\ L,\ k : [\exists\, c \in X_*^s : \Phi_c = \{k\}]. \tag{5}$$

Now we move on to the constraints involving course variables. Each course has the required amount of slots:

$$\sum_k x_{ck} = \rho_c \quad \forall\, s,\ c \in X^s \cup X^s_*. \tag{6}$$

Courses are assigned to slots where the chosen subset contains the subject of the course. Combined with Constraint (1) the following constraint also implies that for each semester no two courses from the same subject can be assigned to the same slot:

$$\sum_{c \in X^s_i} x_{ck} = \sum_{L: i \in L} h^s_{Lk} \quad \forall\, s,\ i,\ k. \tag{7}$$

We conclude by describing the special constraints. Some courses c, which require more than one slot, may need consecutive slots. If a slot k is characterized as the first slot of such a course (i.e. $a_{ck} = 1$), this one and the following $\rho_c - 1$ slots are taken by the course:

$$a_{ck} \leq x_{c(k+f)} \quad \forall\, c \in X \cap C^\alpha,\ k \in K,\ f \in \{0, \ldots, \rho_c - 1\} : k + f \leq |K|. \tag{8}$$

There is exactly one starting slot:

$$\sum_k a_{ck} = 1 \quad \forall\, c \in X \cap C^\alpha. \tag{9}$$

The first slot k of a consecutive sequence of slots for course c must be chosen in such a way that

- all $\rho_c - 1$ following slots are during the same day and
- all ρ_c slots including the first one are feasible slots for course c (i.e. in Φ_c).

If this is not the case, we fix the corresponding variables a_{ck} at zero.

As for the sets of courses which must be assigned to the same slot, we simply equalize the corresponding course variables:

$$x_{c_1 k} = x_{c_2 k} \quad \forall\, g \in G^\beta,\ c_1, c_2 \in C^\beta_g \cap X,\ k. \tag{10}$$

Furthermore, for each common slot from two courses within the same semester, we will allow one additional conflict between the subjects in that particular semester by incrementing the right-hand side of Constraint (4).

Some courses from a set of courses shall not be assigned to the same slot (i.e. no two courses of such a set shall be assigned to the same slot):

$$\sum_{c \in C^\gamma_g \cap X} x_{ck} \leq 1 \quad \forall\, g \in G^\gamma,\ k. \tag{11}$$

Some courses c can not be assigned to certain slots:

$$x_{ck} = 0 \quad \forall\, c \in X,\ k \in K \backslash \Phi_c. \tag{12}$$

This constraint completes the model for the compulsory courses.

Table 4: Data gathered from the assignment of compulsory courses

$y^s_{ik} \in \{0,1\}$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i \in I,\ k \in K,$
$y^s_{ik} = 1$	$\Leftrightarrow$ in semester s, a compulsory course from subject i is in slot k,
$y^s_{*k} \in \{0,1\}$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ k \in K,$
$y^s_{*k} = 1$	$\Leftrightarrow$ in semester s, a compulsory course from ed. sciences is in slot k,
$Q \subset I^2$	combinations with undesired conflicts from compulsory courses.

3.3 Model for Elective Courses

Elective courses are characterized by the fact that they are offered at least twice. Therefore, we introduce two *elective blocks* and require that each elective course is offered at least once in each elective block.

As pointed out in Section 2, it is important to give departments as much influence as possible on the assignment of specific courses to slots. Departments insist that the model for the elective courses respects the following rule, which was originally developed by Kreuzer [10]: Each student has to choose for each subject either the complete elective block 1 or the complete elective block 2 (in contrast to choose for each course either elective block 1 or elective block 2). Naturally, the chosen block can vary among students with different subject combinations. This way, staff from the department has the freedom to assign the specific courses to slots for each elective block.

Courses offered more often than twice can be assigned to arbitrary slots as long as they are offered at least once in each elective block. When, for example, tutorials associated to an elective course are offered more often than twice, it is usually a good idea to offer them multiple times in each elective block to accommodate group sizes. On the other hand, it can also help to offer some tutorials in other slots to provide additional options to the students.

Technically speaking, for each semester s and combination of subjects (i, j), there must exist a valid choice of elective blocks $(t, u) \in T^2 = \{1, 2\}^2$ such that students can attend the elective courses in block t for subject i and in block u for subject j without further conflicts between either the two chosen elective blocks or a chosen elective block with previously assigned compulsory courses. This can be seen as a disjunctive constraint, stating that choice $(1, 1)$, $(1, 2)$, $(2, 1)$ or $(2, 2)$ must be valid. Since such a choice may not exist, we allow additional conflicts from the elective courses and minimize their weighted number.

Elective courses from educational sciences are assumed to be offered in an ample amount such that every student always has a suitable choice. Consequently, elective courses from educational sciences are not part of the elective model (otherwise one would have to guarantee a valid choice $(t, u, v) \in T^3$).

Caused by the decomposition, this model uses the results from the first step as input data. We refer to the data specified in Table 4, which can be easily computed from the values of variables h^s_{Lk} or x_{ck}.

Instead of explicitly describing the four choices, we introduce additional variables w^s_{ijtu} to indicate which pair of elective blocks (t, u) is a valid choice for a combination (i, j), allowing at most one conflict. We further use the variables presented in Table 5. Variables m^s_{itk} indicate whether a subject i is assigned to elective block t in slot k

Table 5: Variables for elective courses

$m^s_{itk} \in \{0,1\}$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i \in I,\ t \in T,\ k \in K,$	
$m^s_{itk} = 1$	$\Leftrightarrow$	in semester s, subject i, elective block t, is assigned to slot k,
$w^s_{ijtu} \in \{0,1\}$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i,j \in I : i < j,\ t,u \in T,$	
$w^s_{ijtu} = 1$	$\Leftrightarrow$	in semester s, (t,u) is a valid choice for combination (i,j), allowing at most one conflict,
$r^s_{ijtuk} \in [0,1]$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i,j \in I : i < j,\ t,u \in T,\ k \in K,$	
$r^s_{ijtuk} = 1$	$\Leftrightarrow$	in semester s, courses from subject i, elective block t, and courses from subject j, elective block u, have a conflict in slot k,
$\dot{r}^s_{ijtk} \in [0,1]$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i,j \in I : i < j,\ t \in T,\ k \in K,$	
$\dot{r}^s_{ijtk} = 1$	$\Leftrightarrow$	in semester s, courses from subject i, elective block t, and compulsory courses from subject j have a conflict in slot k,
$\ddot{r}^s_{iju k} \in [0,1]$	$\forall\, s \in S_{\mathcal{W}}/S_{\mathcal{S}},\ i,j \in I : i < j,\ u \in T,\ k \in K,$	
$\ddot{r}^s_{iju k} = 1$	$\Leftrightarrow$	in semester s, compulsory courses from subject i and courses from subject j, elective block u, have a conflict in slot k,
$e_{ctk} \in \{0,1\}$	$\forall\, c \in E,\ t \in T,\ k \in K,$	
$e_{ctk} = 1$	$\Leftrightarrow$	course c, elective block t, is assigned to slot k (the course determines the semester),
$a_{ctk} \in \{0,1\}$	$\forall\, c \in E \cap C^b m,\ t \in T,\ k \in K,$	
$a_{ctk} = 1$	$\Leftrightarrow$	consecutive slots of course c, elective block t, start in slot k.

during semester s. Variables r^s_{ijtuk} describe if there is a conflict between an elective course of i in block t and an elective course of j in block u in slot k. Variables $\dot{r}^s_{ijtk}$ and $\ddot{r}^s_{iju k}$ describe if there is a conflict between an elective course of subject i in block t and compulsory courses of j respectively a compulsory course of subject i and an elective course of j in block u in slot k. Notice that we always assume $i < j$. Similar to the previous model, we have variables e_{ctk} for course assignments and variables a_{ctk} indicating where a consecutive series of slots starts.

The objective function minimizes the additional conflicts induced by elective courses:

$$\min \sum_{s,k} \sum_{i<j} \theta_{ij} \left(\sum_{t,u} r^s_{ijtuk} + \sum_{t} \dot{r}^s_{ijtk} + \sum_{u} \ddot{r}^s_{iju k} \right).$$

Once again, we start by describing the constraints involving the variables for the subjects, move on to those involving courses and conclude with the special constraints.

Elective courses do not have conflicts with compulsory courses from educational sciences or from the same subject:

$$m^s_{itk} = 0 \quad \forall\, s,\ i,\ t,\ k : y^s_{*k} + y^s_{ik} \geq 1. \tag{13}$$

For a valid choice of elective blocks, there are no conflicts between the chosen elective blocks, unless indicated otherwise by variables r^s_{ijtuk}:

$$m^s_{itk} + m^s_{juk} + w^s_{ijtu} \leq 2 + r^s_{ijtuk} \quad \forall\, s,\ i < j,\ t,\ u,\ k. \tag{14}$$

For a valid choice of elective blocks, there are no conflicts between the chosen elective blocks and the previously assigned compulsory courses, unless indicated otherwise by variables $\dot{r}^s_{ijtk}$ or $\ddot{r}^s_{ijtk}$:

$$m^s_{itk} + y^s_{jk} + \sum_u w^s_{ijtu} \leq 2 + \dot{r}^s_{ijtk} \quad \forall\, s,\ i < j,\ t,\ k, \tag{15}$$

$$y^s_{ik} + m^s_{juk} + \sum_t w^s_{ijtu} \leq 2 + \ddot{r}^s_{ijuk} \quad \forall\, s,\ i < j,\ u,\ k. \tag{16}$$

There can be no further conflict for combinations which already have undesired conflicts from the compulsory courses:

$$r^s_{ijtuk} = \dot{r}^s_{ijtk} = \ddot{r}^s_{ijuk} = 0 \quad \forall\, s,\ (i,j) \in Q : i < j,\ t,\ u,\ k. \tag{17}$$

For each (other) combination, there can be at most one conflict from elective courses. In case of desired conflicts, the right-hand side will be relaxed as in Constraint (4).

$$\sum_k (r^s_{ijtuk} + \dot{r}^s_{ijtk} + \ddot{r}^s_{ijuk}) \leq 1 \quad \forall\, s,\ i < j,\ t,\ u. \tag{18}$$

For each eligible combination of subjects, there is at least one valid choice of elective blocks (with at most one conflict):

$$\sum_{t,u} w^s_{ijtu} \geq 1 \quad \forall\, s,\ (i,j) \in P. \tag{19}$$

For each elective block, there are as many slots containing a specific subject as are needed for the courses from that subject (for all courses, not just for those with special constraints):

$$\sum_k m^s_{itk} = \sum_{c \in E^s_i} \rho_c \quad \forall\, s,\ i,\ t. \tag{20}$$

Now we move on to the constraints involving course variables. Each course has in each elective block the required amount of slots:

$$\sum_k e_{ctk} = \rho_c \quad \forall\, s,\ c \in E^s,\ t. \tag{21}$$

Courses are assigned to slots where the subject of the course has a slot. This constraint also implies that for each semester and each elective block no two courses from a subject can be assigned to the same slot:

$$\sum_{c \in E^s_i} e_{ctk} = m^s_{itk} \quad \forall\, s,\ i,\ t,\ k. \tag{22}$$

The special constraints for elective courses are very similar to those for compulsory courses. We simply describe the semantics of these constraints:

- Courses with consecutive slots need to have consecutive slots in each elective block.
- Two courses with a desired conflict must have a conflict in elective block 1 and a conflict in elective block 2.
- Courses that can not have a conflict with certain other courses can not have a conflict in either elective blocks with any of the other courses.
- Courses with restrictions on the set of feasible slots must respect those in both elective blocks.

These constraints complete the model for the elective courses.

Table 6: Comparison of test instances

				Undesired Conflicts		Runtime	
Instance	Students	$\theta = 0$	Sem.	Compulsory	Elective	Co. [min]	El. [sec]
InstReal	8836	30	$S_{\mathcal{W}}$	620	2	805	5
			$S_{\mathcal{S}}$	167	0	322	3
Inst01	9381	31	$S_{\mathcal{W}}$	474	0	200	5
			$S_{\mathcal{S}}$	141	0	31	4
Inst02	9892	37	$S_{\mathcal{W}}$	-	-	mem	-
			$S_{\mathcal{S}}$	106	0	238	4
Inst03	10054	31	$S_{\mathcal{W}}$	371	0	103	2
			$S_{\mathcal{S}}$	46	0	68	2
Inst04	10973	25	$S_{\mathcal{W}}$	626	0	136	3
			$S_{\mathcal{S}}$	171	0	47	3
Inst05	10566	37	$S_{\mathcal{W}}$	277	0	29	4
			$S_{\mathcal{S}}$	45	0	3	4
Inst06	8719	32	$S_{\mathcal{W}}$	354	0	84	4
			$S_{\mathcal{S}}$	86	0	55	2
Inst07	7972	30	$S_{\mathcal{W}}$	539	0	322	8
			$S_{\mathcal{S}}$	173	0	35	4
Inst08	6756	26	$S_{\mathcal{W}}$	525	5	477	9
			$S_{\mathcal{S}}$	164	0	118	2
Inst09	11326	28	$S_{\mathcal{W}}$	619	0	255	4
			$S_{\mathcal{S}}$	195	0	162	4
Inst10	9737	30	$S_{\mathcal{W}}$	637	0	37	7
			$S_{\mathcal{S}}$	198	0	13	4

4 Computational Results

In this section, we start by providing some basic information from our test cases. Then, we present the computational results obtained from the real and the artificial instances.

The problem includes 21 different subjects with 658 courses in total which need to be assigned to 25 slots in 10 semesters. Hence, each model is run twice with 5 simultaneously considered semesters. 350 courses are offered during a winter semester and 308 courses during a summer semester. Out of the 658 courses, 415 courses are compulsory courses and 243 are elective ones. The average amount of required slots per semester is 74 slots, ranging from 43 to 98 slots. 12 courses need consecutive slots, 85 courses need desired conflicts, 5 courses have other courses they are not allowed to share slots with and 35 courses are restricted regarding the slots.

In the real-life case, 8836 students were enrolled in total and 30 of the 210 combinations were not chosen by any student (i.e. $\theta = 0$). Two combinations were ineligible, a phenomenon that was exploited by constraints (4) and (19).

The decomposition and the models were implemented in Java 8, using Gurobi 6.0.0 as the solver. Notice that the programming language has almost no influence on the runtime, as the solver is responsible for by far the greatest part of the runtime. Experiments were performed on a 64-bit Linux system, running on an Intel Core i7-5820K CPU ($6 \times 3.3\,\text{GHz}$) with 32 GB of RAM. For the compulsory courses, a size limit of $n = 4$ for the subject sets L and a time limit of 24 hours were laid down.

Furthermore, 10 random, artificial instances were generated to test our model. To this purpose, the number of students for each subject combination was varied while the data for the courses was kept the same. Numbers were generated using a model with two components: First, unchosen combinations were determined using a Bernoulli distribution. Then, random numbers for the chosen combinations were generated, assuming a log-normal distribution and ceiling values to obtain integers. Parameters for both components were estimated by maximum-likelihood estimators. The results from the artificial instances can also be found in Table 6.

To reduce variability from the solver, optimizations were executed twice for both $S_{\mathcal{W}}$ and $S_{\mathcal{S}}$ in each instance, using solver seeds 0 and 1. In each entry of Table 6, the better one of the two results is presented, considering a solution with less conflicts to be better than a solution obtained in less runtime.

As expected, the number of conflicts and the runtime of the model for the compulsory courses depend heavily on the specific instance. In one case, the model for the compulsory models could not be solved due to memory restrictions. Taking the better one of the two results, in 19 out of 21 cases the elective courses did not produce further conflicts, proving optimality of the solution with respect to the decomposition. Although we only allow conflicts from the elective courses for those combinations that do not already have conflicts from the compulsory courses, the number of additional conflicts in the second step is very low in comparison to the existing conflicts from the first step. Obviously, the number of additional conflicts from the elective courses depends on the previous assignment of the compulsory courses.

For all instances, subsequent experiments without the size limitation and without taking some of the special constraints into account were performed. Those models were usually solved within 10 to 20 minutes. In all cases, the experiments yielded the same number of conflicts as the original models, proving optimality with respect to the size assumption $n = 4$.

In contrast, size limitations of $n < 4$ for the original models led to additional conflicts, while models with size limitations of $n > 4$ were not solvable within 24 hours.

5 Alternative Model Formulations

In this section, we discuss some conceivable modifications to the provided models.

There are several modifications to the described model that technically do not change the set of feasible solutions, but may have influence on the runtime or memory consumption. In our case, one could remove Constraint 2 or use for several other constraints an equality relation instead of an inequality relation, or vice versa. The variants presented in Sections 3.2 and 3.3 are the ones which performed best in our case. However, when working with black box solvers, such tuning should always be interpreted with caution.

For the compulsory courses, experiments were performed with constraint types given by Hall's theorem to describe the special constraints without explicit course variables. If there is only one special constraint, e.g. if courses from subjects i and j_1 should have one desired conflict, one can describe this in the following way (without using Hall's theorem yet):

$$\sum_{L: i, j_1 \in L} \sum_{k} h^s_{Lk} \geq 1,$$

If there are several such constraints, it does not suffice to add their isolated formulations, but one has to consider them together. For example, when in semesters s courses from subjects i, j_1 should have one desired conflict and courses from subjects i, j_2 should have one, we would express this, using Hall's theorem, in the following way:

$$\sum_{L: i, j_1 \in L} \sum_{k} h^s_{Lk} \geq 1, \tag{23}$$

$$\sum_{L: i, j_2 \in L} \sum_{k} h^s_{Lk} \geq 1, \tag{24}$$

$$\sum_{\substack{L: i, j_1 \in L \\ \vee i, j_2 \in L}} \sum_{k} h^s_{Lk} \geq 2. \tag{25}$$

The point is that simply one slot with subjects i, j_1 and j_2 is not enough because both desired conflicts come from different courses. In this case, we call constraints (23) and (24) *interacting*. If there are more than two constraints to be considered, we have to factor in additional restrictions for every subset of interacting constraints. As long as there are only constraints from one type involved, it is quite easily possible to formulate necessary and sufficient Hall-type constraints as demonstrated above and solve the corresponding integer programming models. This agrees with the successful application of Hall's theorem to timetabling construction used by Lach and Lübbecke [11]. Formulating necessary and sufficient Hall-type constraints for constraints of different types (one needs to consider all possible combinations of them) requires an exponential number of *types of constraints* and is very complicated and error prone. Hence, we decided to use the course variables.

Course variables were generally introduced for all courses and not only for courses with special constraints. This led the solver to earlier heuristic solutions which overall accelerated the solving process in comparison to models incorporating course variables only for courses with special constraints.

As was mentioned in Section 3.3, elective courses offered more often than twice can be assigned to arbitrary slots as long as they are offered at least once in each elective block. A formal exploitation of these courses does currently not seem to be advantageous as the current model performs sufficiently well and provides flexibility for the departments.

Although the model was not presented explicitly, we did a first test for the real-life case using a combined model for both students starting in a winter semester and students starting in a summer semester. To this purpose, one replaces all semester indices s by pairs (b, s), where b indicates the type of start semester, winter or summer. To give an estimate of the quality of the solution for the combined model, we separately determined the optima for the compulsory courses for the cases with students starting in a winter semester and with students starting in a summer semester. The sum of these optima is a valid lower bound for the model with all students. The solution for $S_{\mathcal{W}}$ was within 4 % of the lower bound and was obtained after 2 days, the solution for $S_{\mathcal{S}}$ was within 2.5 % of the lower bound and was obtained after 0.5 hours. Even if good quality solutions were found quite quickly without solver tuning, proving optimality within the model turned out to be intractable. In smaller test cases, where we could prove optimality for the solutions of the combined model, those lower bounds were usually no sharp bounds, but also rarely much more than 5 % off. Therefore, it seems promising to use the following procedure for the combined model:

1. Solve the models for $S_{\mathcal{W}}$ and $S_{\mathcal{S}}$ separately to optimality.
2. Find good solutions for the combined model.
3. Prove optimality/estimate the quality of the solutions for the combined model using the lower bound given by the sum of the separate solutions for $S_{\mathcal{W}}$ and $S_{\mathcal{S}}$.

6 Conclusions

In this paper, we presented an approach to a real-life timetabling case. The problem incorporates certain peculiarities, uncommon to most timetabling problems from the literature. We proposed a model that minimizes the number of conflicts and is in particular able to cope with the request for flexibility by the involved departments at the university. Decomposition was used to split the problem into a model for compulsory courses and a model for elective courses.

For the compulsory courses, additional variables for subsets of subjects were introduced. These simplified the computation of the objective function and made a linearization expendable. Elective courses were modelled using elective blocks and without the explicit need for disjunctive constraints. Both models were solved by integer programming and for the great majority of cases optimal solutions were obtained.

As there is a real-life application associated to the research project, the obtained timetables are expected to be implemented soon. Although the description of the problem seems to be specific to our university, several other universities in Germany face the same type of challenges. Therefore, we consider our work a significant contribution towards the practical applicability of timetabling.

Future research should address the problem of students beginning their studies during a summer semester. This phenomenon remained mostly unconsidered, as it drastically increases the computational complexity and is mostly unwanted by the departments. Nonetheless, having the students' best interests in mind, it appears to be a reasonable requirement, even if the current solution already greatly improves upon the previously used timetables. Besides from the suggested procedure in Section 5, it would be interesting to implement a column generation approach, since the computational complexity arises mostly from the vast amount of variables. Furthermore, column generation approaches have recently been successfully applied to timetabling problems by Cacchiani et al. [7] as well as by Sørensen and Dahms [18] and might enable us to relinquish the heuristic size limitation for the subset variables.

On the practical side, data for the numbers of students on a semester specific level of detail should and will be taken into account, allowing us to consider drop-out rates. In addition, a time series forecasting model for the numbers of students is planned and will hopefully also compensate for the rather long period of validity of the timetables.

References

1. Al-Yakoob SM, Sherali HD, A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations, European Journal of Operational Research, 180, 1028–1044 (2007)
2. Bettinelli A, Cacchiani V, Roberti R, Toth P, An overview of curriculum-based course timetabling, TOP, 1–37 (2015)
3. Bonutti A, De Cesco F, Di Gaspero L, Schaerf A, Benchmarking curriculum-based course timetabling: formulations, data format, instances, validation, visualization, and results, Annals of Operations Research, 194(1), 59–70 (2012)

4. Bullnheimer B, An Examination Scheduling Model to Maximize Student's Study Time, In: Burke EK, Carter MW (eds.), PATAT 1997, Lecture Notes in Computer Science, 1408, 78–91. Springer, Berlin Heidelberg (1998)
5. Burke EK, Mareček J, Parkes AJ, Rudová H, Decomposition, reformulation, and diving in university course timetabling, Computers & Operations Research, 37, 582–597 (2010)
6. Burke EK, Petrovic S, Recent research directions in automated timetabling, European Journal of Operational Research, 140, 266–280 (2002)
7. Cacchiani V, Caprara A, Roberti R, Toth P, A new lower bound for curriculum-based course timetabling, Computers & Operations Research, 40, 2466–2477 (2013)
8. Daskalaki S, Birbas T, Housos E, An integer programming formulation for a case study in university timetabling, European Journal of Operational Research, 153, 117–135 (2004)
9. Kiaer L, Yellen J, Weighted graphs and university course timetabling, Computers & Operations Research, 19(1), 59–67 (1992)
10. Kreuzer A, Starcevic-Srkalovic L, Das Zeitfenstermodell für Lehramtsstudiengänge, In: Zervakis PA (ed.), Lehrerbildung heute. Impulse für Studium und Lehre, 22–23. HRK Hochschulrektorenkonferenz self-published, Bonn (2014, in German)
11. Lach G, Lübbecke ME, Optimal University Course Timetables and the Partial Transversal Polytope, In: McGeoch CC (ed.), WEA 2008, Lecture Notes in Computer Science, 5038, 235–248. Springer, Berlin Heidelberg (2008)
12. Laporte G, Desroches S, Examination timetabling by computer, Computers & Operations Research, 11(4), 351–360 (1984)
13. McCollum B, A Perspective an Bridging the GAP Between Theory and Practice in University Timetabling, In: Burke EK, Rudová H (eds.), PATAT 2006, Lecture Notes in Computer Science, 3867, 3–23. Springer, Berlin Heidelberg (2007)
14. MirHassani SA, A computational approach to enhancing course timetabling with integer programming, Applied Mathematics and Computation, 175, 814–822 (2006)
15. Müller T, Rudová H, Real-life curriculum-based timetabling with elective courses and course sections, Annals of Operations Research, 1–18 (2014)
16. Schaerf A, A Survey of Automated Timetabling, Artificial Intelligence Review, 13, 87–127 (1999)
17. Schimmelpfeng K, Helber S, Application of a real-world university-course timetabling model solved by integer programming, OR Spectrum, 29, 783–803 (2007)
18. Sørensen M, Dahms F, A Two-Stage Decomposition of High School Timetabling applied to cases in Denmark, Computers & Operations Research, 43, 36–49 (2014)
19. Wehrer A, Yellen J, The design and implementation of an interactive course-timetabling system, In: McCollum B, Burke EK, White G (eds.), PATAT 2010, Conference Proceedings, 556–558. Queen's University, Belfast (2010); Annals of Operations Research, 218(1), 327–345 (2013)

MISTA 2015

A Hybrid Constraint Integer Programming Approach to Solve Nurse Scheduling Problems

Erfan Rahimian • Kerem Akartunali • John Levine

Abstract The Nurse Scheduling Problem can be simply defined as assigning a series of shift sequences (schedules) to several nurses over a planning horizon according to some constraints and preferences. The inherent benefits of having higher-quality and more flexible schedules are a reduction in outsourcing costs and an increase of job satisfaction in health organizations. In this paper, we present a novel systematic hybrid algorithm, which combines Integer Programming (IP) and Constraint Programming (CP) to efficiently solve highly-constrained Nurse Scheduling Problems. Our focus is to exploit the problem-specific information to improve the performance of the algorithm, and therefore obtain high-quality solutions as well as strong lower bounds. We test our algorithm based on some real-world benchmark instances. Very competitive results are reported compared to the state-of-the-art algorithms from the recent literature, showing that the proposed algorithm is able to solve a wide variety of real-world instances with different complex structures.

1 Introduction

In order to ensure the right staff on the right duty at the right time, Nurse Scheduling (NS) has drawn significant attention during the last few decades, helping many health organizations to increase their efficiency and productivity. Creating a high-quality nurse schedule raises the recruitment and retention levels of nursing personnel, and maintains a reasonable overtime budget for nursing staff. In terms of financial issues, it can reduce outsourcing and planning costs due to hiring fewer bank nurses to compensate gaps in rosters, and having flexible schedules [1, 2]. In terms of human resource issues, it can increase the job satisfaction and diminish the fatigue and stress, and hence result in improving caring services provided to patients [3, 4].

Nurse Scheduling Problem (NSP) aims to generate schedules for several nurses over a planning horizon. A schedule consists of a sequence of different types of shifts (e.g. early, late,

Erfan Rahimian, Kerem Akartunali
Dept. of Management Science, University of Strathclyde, Glasgow, G1 1QE, UK
E-mail: {erfan.rahimian, kerem.akartunali}@strath.ac.uk

John Levine
Computer And Information Sciences, University of Strathclyde, Glasgow, G1 1XH, UK
E-mail: john.levine@strath.ac.uk

vacations) spanning over the whole planning period. The pattern of shifts is generated according to a set of requirements such as hospital regulations, and a number of preferences such as fair distribution of shifts between nurses. Due to their complex and highly-constrained structure, most NSPs in real-world situations are computationally challenging and they can be also classified as NP-hard [5, 6]. The inherent nature of the problem usually leads to divide all constraints to two categories in practice: hard and soft constraints. Hard constraints must be satisfied to have a feasible roster, whereas soft constraints may be violated. To evaluate the quality of a roster, one can minimize the sum of all penalties incurred due to soft constraint violations. For more information regarding NSPs and generally staff scheduling problems, we refer interested readers to [3, 28].

The focus of this paper is on integrating Integer Programming (IP) and Constraint Programming (CP) to solve NSPs, where we exploit the problem-specific information in order to improve both IP and CP performance. In the literature, there are two areas of general methods used to solve these problems: exact and heuristic methods. Exact methods involve IP [1, 7, 8] and CP [9, 10], which are capable of finding the optimal solution, albeit often resulting in unacceptable computational times. However, recent research in Operations Research and Artificial Intelligence communities, combined with powerful solvers such as IBM Ilog Cplex and Gurobi, focused on using these methods in hybrid settings [14-16]. On the other hand, in order to address the computational limitations of exact methods, many heuristic methods have been proposed in the literature. However, these methods sacrifice the guarantee of an optimal solution (or even any information about the solution quality) in order to generate good solutions in acceptable computational times. We note [11-13] as some recent examples of using heuristic methods in the NSP literature.

In recent years, some researchers experimented with hybridizations of different methods, e.g. CP and heuristics [14], IP and heuristics [15], and less well-investigated combination of IP and CP [16], in order to utilize the strengths of all methods together. In this paper, we propose a new systematic hybrid algorithm using IP and CP approaches, which is capable of finding the optimal solution. Due to the exact nature of the proposed algorithm, it can generate a good solution as well as a good lower bound in contrast to heuristic methods. The hybrid algorithm exploits the problem-specific information to reduce the search space, to fine tune the search parameters, and to improve the efficiency of the search process in a novel way. In other words, using an IP approach as the main solution method, we employ a CP approach and some other algorithmic aids to improve the efficiency of the algorithm. Moreover, the proposed algorithm is designed to obtain the best result in a pre-defined limited computational time. We model the problem according to a general comprehensive model reported in the literature [17] and evaluate it using some test instances published therein.

The rest of this paper is organized as follows: problem definition and assumptions are explained in Section 2. The mathematical and CP formulations is presented in Section 3 and 4. In Section 5, we describe the proposed hybrid algorithm and its components. Computational results are reported in Section 6, and some conclusions are drawn in Section 7.

2 Problem Definition

NSP is the process of assigning a number of nurses to a number of work shifts during a planning horizon according to a set of requirements and constraints. These constraints are usually categorized to hard and soft constraints. In the following, we define decision variables and constraints according to the conceptual model described in [17], which will be used to construct an IP model.

We define our decision variables for each nurse, for each day, and for each shift type. This way of modeling allows us to better utilize the problem-specific structure in order to reduce the search space, although it is less flexible and contains more symmetry compared to the pattern-based modelling (e.g. [12]), which generates all possible weekly shift sequences (patterns) and hence considers all constraints except coverage constraints. We assume the current roster is modelled over a planning horizon in an isolated way, i.e. no information (history) from the

previous roster is used to construct the current one. We also consider a day-off as a shift type for modelling purposes. For the sake of simplicity, we assume all nurses belong to the same skill category. In addition, we assume all rosters start from Monday and are made from a complete week (includes seven days with a two-day weekend). The constraints of the model are:

1. Maximum one assignment per shift type per day,
2. Coverage constraints: the number of shift types for each day must be fulfilled,
3. The minimum and maximum number of:
   (a) shift assignments within the scheduling period,
   (b) consecutive working days over the planning horizon,
   (c) working hours within the scheduling period (and/or during a week),
   (d) shift assignments within a week,
   (e) shift assignments at the weekend,
   (f) consecutive shift types over the planning period,
4. Minimum number of days-off after a night shift or a series of night shifts,
5. Complete weekends: over the weekends, there should be either an assignment to all days of weekends or no assignments at all,
6. No night shift before free weekends, where there is no assignment at all,
7. Maximum number of consecutive worked weekends, where there is at least one assignment,
8. Requested shifts (days) on or off,
9. Forbidden shift type patterns (e.g. the "ND" pattern, where the shift type "D" is not allowed to be assigned right after the shift type "N").

In the next two sections, we formulate this model using Integer Programming (IP) and Constraint Programming (CP). We also note that the above constraints can be considered hard or soft according to different settings. For the sake of simplicity, we only provide here a formulation assuming that all constraints are hard. In case any soft constraints exist in the model, our objective function can be defined as the weighted sum of all slack variables in IP or reified variables in CP for each soft constraint.

3 Mathematical Formulation

Here we present our mathematical formulation using Integer Programming based on the definitions and assumptions from the previous section. The variables, parameters, and constraints of the model are defined as follows:

Decision Variables:

x_{ead}	Binary variable indicating whether shift type *a* on day *d* is assigned to nurse *e* or not.
p_{ed}	Binary variable indicating whether nurse *e* works on day *d* or not.
k_{ew}	Binary variable indicating whether nurse *e* is assigned to weekend *w* or not.
y_{ea}	Total number of times that shift type *a* assigned to nurse *e* over the planning period.
z_{ewa}	Total number of shift type *a* assigned to nurse *e* during week *w*.

Parameters:

N	Set of nurses.
D	Set of days.
A	Set of shift types.
W	Set of weeks.

H_a	Set of shift types that cannot be assigned immediately after shift type *a*.
PR_{ad}	Set of pre-assigned nurses to shift type *a* on day *d*.
ML_e, MU_e	Minimum and maximum number of shifts that can be assigned to nurse *e* within the planning period.
WL_w, WU_w	Minimum and maximum number of shifts that can be assigned to a nurse within week *w*.
VL_d, VU_d	Minimum and maximum number of shifts that can be assigned to nurses on day *d*.
AL, AU	Minimum and maximum number of hours that can be assigned to each nurse during the planning period.
EL_w, EU_w	Minimum and maximum number of hours that can be assigned to each nurse during week *w*.
NL, NU	Minimum and maximum number of consecutive working days over the planning period.
HL_a, HU_a	Minimum and maximum number of consecutive shift type *a* over the planning period.
KL, KU	Minimum and maximum number of worked weekends over the planning horizon.
CU	Maximum number of consecutive worked weekends over the planning period.
UT_a	Total workloads (hours) of shift type *a* within the planning period.
UT_{aw}	Total workloads (hours) of shift type *a* during week *w*.

Constraints:

Next, we present our IP formulation, where the order of the constraints is preserved the same as the order of the constraints presented in Section 2:

$$\sum_{a \in A} x_{ead} = 1, \forall e \in N, d \in D \quad (1)$$

$$p_{ed} = \sum_{a \in A - \{r\}} x_{ead}, \forall e \in N, d \in D \quad (2)$$

$$VL_d \leq \sum_{e \in N} p_{ed} \leq VU_d, \forall d \in D$$

$$y_{ea} = \sum_{d \in D} x_{ead}, \forall e \in N, a \in A \quad (3.a)$$

$$ML_e \leq \sum_{a \in A} y_{ea} \leq MU_e, \forall e \in N$$

$$\sum_{g=d}^{NU+d} p_{eg} \leq NU, \forall e \in N, d \in \{1 \dots |D| - NU\} \quad (3.b)$$

$$\sum_{i=1}^{NL-1} p_{ed+i} \leq p_{ed} + p_{ed+NL} + NL - 2, \forall e \in N, d \in D$$

$$AL \leq \sum_{a \in A} y_{ea} UT_a \leq AU, \forall e \in N \quad (3.c)$$

$$z_{ewa} = \sum_{d=7(w-1)+1}^{7w} x_{ead}, \forall e \in N, a \in A, w \in W$$

$$EL_w \le \sum_{a \in A} z_{ewa} UT_{aw} \le EU_w, \forall e \in N, w \in W$$
$$WL_w \le \sum_{a \in A} z_{ewa} \le WU_w, \forall e \in N, w \in W \quad (3.d)$$

$$k_{ew} \le p_{ed} + p_{ed+1} \le 2k_{ew}, \forall e \in N, w \in W, d = 7w - 1 \quad (3.e)$$

$$KL \le \sum_{w \in W} k_{ew} \le KU, \forall e \in N$$
$$\sum_{g=d}^{HU_a+d} x_{eag} \le HU_a, \forall e \in N, a \in A, d \in \{1 \dots |D| - HU_a\} \quad (3.f)$$

$$\sum_{i=1}^{HL_a-1} x_{ead+i} \le x_{ead} + x_{ead+HL_a} + HL_a - 2, \forall e \in N, a \in A, d \in D$$
$$x_{end} \le x_{end+1} + 1 - p_{ed+1}, \forall e \in N, d \in \{1 \dots |D| - 1\} \quad (4)$$

$$x_{end} - p_{ed+1} \le 1 - p_{ed+2}, \forall e \in N, d \in \{1 \dots |D| - 2\}$$
$$x_{erd} = x_{erd+1}, \forall e \in N, d \in \{6,13, \dots, |D| - 1\} \quad (5)$$
$$x_{end} \le p_{ed+1} + p_{ed+2}, \forall e \in N, d \in \{5,12, \dots, |D| - 2\} \quad (6)$$

$$\sum_{i=0}^{CU} k_{ew+i} \le CU, \forall e \in N, w \in \{1 \dots |W| - CU\} \quad (7)$$
$$x_{ead} = 1, \forall e \in PR_{ad}, a \in A, d \in D \quad (8)$$
$$x_{ead} + x_{ehd+1} \le 1, \forall e \in N, a \in A, h \in H_a, d \in \{1 \dots |D| - 1\} \quad (9)$$

In constraint (4), we assume that there should be two days-off after a night shift or a series of night shift types. Furthermore, in constraints (2), (4), (5), and (6), "*n*" and "*r*" indicate a night shift type and a day-off, respectively.

4 Constraint Programming Formulation

Here we present our CP formulation based on Constraint Satisfaction Problem (CSP) model according to the definitions and assumptions provided in Section 2. The presented model is detailed enough for the needs of this paper, however, we would add other redundant constraints or variables to increase the efficiency of the CP solver. In this section, first, we concisely explain the two types of global constraints which we use in the CP model: *Cardinality* and *Stretch*. For more information about global constraints in CP, we refer to [24-25]. Next, we define the variables, parameters, and constraints of the model. We use the same parameters as defined in IP formulation (Section 3), therefore we define only new ones here.

Cardinality constraints (aka. *GCC* or *generalized cardinality*) bounds the number of times that variables take a certain set of domain values. It is written as $cardinality(x, v, l, u)$ where x is a set of variables $(x_1, \dots, x_n)$; v is a m-tuple of domain values of the variables x; l and u are m-tuples of non-negative integers defining the lower and upper bounds of the times value v being taken by variable x, respectively. The constraint defines that, for $j = 1, \dots, m$, at least l_j and at most u_j of the variables x take value v_j.

Stretch constraints bounds the sequence of consecutive variables that take the same value (stretch), i.e. $x_{j-1} \neq 1, x_j, \dots, x_k = v, x_{k+1} \neq v$. It is written as $stretch(x, v, l, u, P)$ where x is a set of variables $(x_1, \dots, x_n)$; v is a m-tuple of possible domain values of x; l and u are m-tuples of lower and upper bounds for x, respectively. P is a set of patterns, i.e. pairs of values (v_j, v_k),

requiring that when a stretch of value v_j immediately precedes a stretch of value v_k, the pair (v_j, v_k) must be in P.

Decision Variable:

s_{ed}	Integer variable indicating the shift type assigned to nurse *e* on day *d*.

Parameters:

$\widetilde{H}_a$	Set of shift types that can be assigned immediately after shift type a.
UT	The vector of total workloads (hours) of the shift types within the planning period.
UT_w	The vector of total workloads (hours) of the shift types during week *w*.

Constraints:

Next, we present our CP formulation based on the defined global constraints, where the order of the constraints is preserved the same as the order of the constraints presented in Section 2:

$$cardinality\left(\bigcup_{e\in N} s_{ed}, A, VL_d, VU_d\right), \forall d \in D \quad (2)$$

$$cardinality\left(\bigcup_{d\in D} s_{ed}, A, ML_e, MU_e\right), \forall e \in N \quad (3.a)$$

$$stretch(s_{ed}, A, NL, NU, P), \forall e \in N, d \in D, P = \{(a, r) | a \in A\} \quad (3.b)$$

$$AL \leq prod(s_{ed}, UT) \leq AU, \forall e \in N, d \in D \quad (3.c)$$

$$EL_w \leq prod(s_{ed}, UT_w) \leq EU_w, \forall e \in N, w \in W, d = 7(w-1)+1$$

$$cardinality\left(\bigcup_{d=7(w-1)+1}^{7w} s_{ed}, A, WL_w, WU_w\right), \forall e \in N, w \in W \quad (3.d)$$

$$cardinality(s_{ed}, r, KL, KU), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1\}\} \quad (3.e)$$

$$stretch(s_{ed}, a, HL_a, HU_a, P), \forall e \in N, d \in D, a \in A, P = \{\} \quad (3.f)$$

$$stretch(s_{ed}, n, 2,3, P), \forall e \in N, d \in D, P = \{(n, r)\} \quad (4)$$

$$s_{ed} = s_{ed+1}, \forall e \in N, d \in \{6,13, \dots, |D| - 1\} \quad (5)$$

$$stretch(s_{ed}, n, 2,3, P), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1,2\}\}, P = \{(n, r)\} \quad (6)$$

$$stretch(s_{ed}, r, 2(|W| - CU), 2|W|, P), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1\}\}, P = \{(r, r)\} \quad (7)$$

$$s_{ed} = a, \forall e \in PR_{ad}, a \in A, d \in D \quad (8)$$

$$stretch(s_{ed}, a, 0,2, P), \forall e \in N, d \in D, P = \{(a, \widetilde{H}_a) | a \in A\} \quad (9)$$

In constraint (4), we assume that there should be two days-off after a night shift or a series of night shift types. Furthermore, in the mentioned constraints, "*n*" and "*r*" indicate a night shift type and a day-off, respectively. It should be noted that constraint (1) is already satisfied due to the inherent structure of the CP model.

5 Integration of CP and IP

For small- to medium-sized problems, IP solvers are often efficient to find the optimal solution and to generate strong lower bounds. Similarly, CP solvers are capable of finding feasible solutions. However, using these approaches on their own for solving large-scale problems, or even small-scale problems with a highly-constrained structure often leads to a very poor performance. For example, solving NSPs using the model presented in Section 3 with a pure IP approach, we were not able to obtain an optimal solution (and in some cases even a good-quality solution) in a reasonable amount of time, where some instances took more than 24 hours to solve. Similarly, a pure CP approach results in poor performance as well, since it often takes a long time to achieve a feasible or optimal solution. Therefore, it is intuitive to hybridize them in order to utilize their strengths for efficiently solving NSPs. In this paper, we integrate IP and CP approaches in a pipeline fashion to solve the problem. To improve the efficiency of the hybrid algorithm, we exploit the problem structure to provide valuable information about search space, hence improve the performance of the proposed algorithm. Indeed, we use a CP approach and some other algorithmic procedures to help the IP approach as our main solution method.

The algorithm presented in this paper is tested on nine different instances published in [17]. The diversity in the structure and complexity of these instances allows us to test our algorithm thoroughly. Table 1 provides more information about these instances, where the reported number of variables and constraints are based on the described model presented in Section 3. It is noteworthy to mention that although we tried to solve a few instances based on real-world cases, we developed our solution method without any fine tuning. Therefore, we believe that our approach can be easily generalized to solve different instances based on the presented IP model.

Table 1. Benchmark instances

Instance	Nurses	Shift types	Days	Shift permutations	Variables	Constraints
GPOST	8	3	28	3136	5680	5504
GPOSTB	8	3	28	3136	5680	5496
ORTEC01	16	5	33	7821	19096	19170
ORTEC02	16	5	33	7821	19101	19175
Valouxis-1	16	4	28	5824	9776	9968
SINTEF	24	6	21	6867	8118	6927
WHPP	30	4	14	5880	6000	5842
MILLAR-1	8	3	14	784	1956	1820
LLR	27	4	7	1323	1139	979

In the following, we provide a brief description of the performance of the hybrid algorithm, and later we will elaborate each associated component. After a quick pre-processing in order to create appropriate data structures for the algorithm, at first step, we employ an *IP pre-solver* in order to identify any valuable information. If any valuable information is identified, we continue to use the *IP solver* (rather than a *CP solver*) for the next steps, since it has more potential to be successful in solving the problem, as we experienced in our experiments. In the next step, we employ a *CP solver* to solve the problem considering only those constraints which will not make the problem difficult to solve. Identifying difficult constraints is achieved with solving a hierarchy of different CSPs iteratively. Next, using the information provided by the CP solver operated on a modified problem and generated CSPs, we solve the problem by an *IP solver* (or the *CP solver* based on the obtained information from the IP pre-solver) during the remaining time. We also add three other components to reinforce the search process using the exploited problem-specific information: i) *Symmetry breaker*, which tries to remove (or mitigate) the symmetric structures; ii) *Weight balancer*, which tries to modify each constraint's weight based on a pre-defined threshold in order to tighten the problem formulation; and iii) *Decomposer*, which provides a lower bound for the IP solver.

It should be noted that the proposed hybrid algorithm runs in a pre-defined time to solve the problem. In fact, the user determines the running time of each component by setting the relevant computational time parameter.

The schematic diagram of the proposed algorithm is depicted in Figure 1:

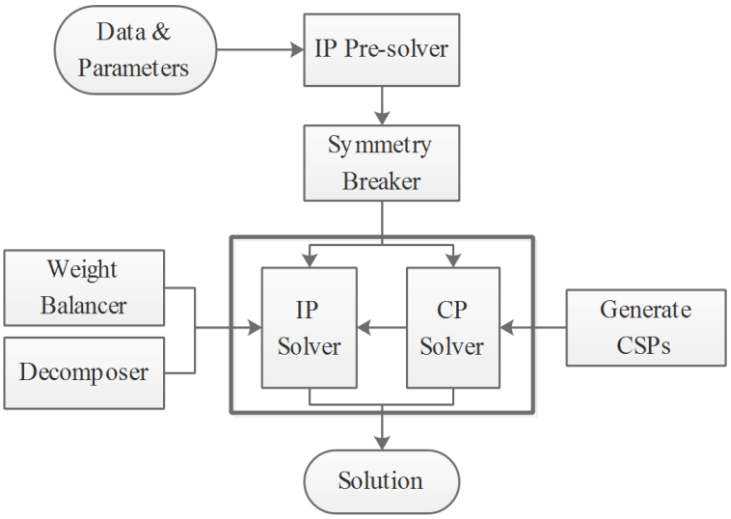


Fig. 1. Schematic diagram of the proposed hybrid algorithm

Next, we explain each component individually in more details:

IP Pre-solver: In fact, this component is the first step in most of the commercial solvers to analyze and simplify the problem structure, and also identify any specific structures such as network flow or assignment problems. If the IP solver can identify any particular structures, it often leads to a better performance during the search process. Here, we only call the pre-solve step of an IP solver from the hybrid algorithm as a black-box. We use the information obtained from this step to predict if there are any specific structures, and therefore improving the performance of the IP solver. Particularly, we use the obtained lower bound and relaxed objective function value to understand the existence of any specific structures in this black-box indirectly. According to our experiments, if the IP pre-solver component provides a stronger (greater) lower bound compared to the relaxed objective function value (the initial identified lower bound for an IP problem), the employed IP solver is a better choice to solve the problem, otherwise we will use the *CP solver* instead. We also switch on the relevant parameter for the pre-solve step of the IP solver to the highest degree (aggressive mode) in this component (e.g. setting the *Presolve* parameter in Gurobi). Moreover, using the reported number of constraints and variables in this step, if they are more than a user-defined threshold (*psThr*), we will change the search strategy of the *IP Solver* accordingly. We will explain this setting in the IP Solver component in more details.

CP Solver: During the search process, the hybrid algorithm may call the CP solver in two cases: first, as the main solver if the IP pre-solver does not provide valuable information about the problem due to its complex and highly-constrained structure; second, as an aid for the IP solver to provide a good-quality initial solution. This solver solves the problem based on the Constraint Satisfaction Problem (CSP) model presented in Section 4. In our experiments on the benchmark instances, CP approach did not provide very good-quality solutions in a limited time. To address this issue, we implement the following procedure: First, we generate a CSP model considering all constraints that have a weight higher than a user-defined threshold (*cspThr*). If the problem

is infeasible, we will increase the threshold by one unit. Otherwise, we will generate a number of solutions based on the modified model according to a user-defined parameter (*numSols*). Therefore, on each threshold level, there might be several feasible solutions. This process continues until the number of constraints in the new generated model is equal to the number of constraints in the original model. Finally, we report the best-quality solution in terms of objective function value. Next if the IP solver is a candidate for solving the problem, the reported solution will be imported to the IP solver. Otherwise, we continue solving the problem using the CP solver in the remaining time. We will explain this setting in the IP solver component in more details. The pseudo code of this procedure is presented below, where *p*, *p'*, *cspThr*, and *numSols* indicate the original problem, the new generated problem in each threshold level, the user-defined threshold level, and the user-defined number of solutions needs to be generated in each threshold level, respectively.

```
Solutions = empty
p' = p
While (true)
   p' = generateCSP(p, cspThr)
   If p' is feasible then
      For i = 1 to numSols
            Solutions.add(solve(p'))
      Next
   Else
      cspThr++
   End If
   If numConstraint(p') >= numConstraint(p) then break
End While
Return bestObj(Solutions)
```

Using the information provided in this procedure by solving a variety of CSP problems, we can also find out an estimate for the difficulty of each constraint. If the solution time for adding a constraint to a problem in order to generate a new modified problem is significant, we will count it as a "difficult constraint". We only record the solution time for the last occurrence when a specific constraint is added to a problem during the process of generating CSPs. To our experiments, 15 seconds is sufficient for most of the benchmark instances. This simple inference helps us later in the *Weight Balancer* component to make the formulation of the problem tighter.

IP Solver: In this component, we use a state-of-the-art IP solver to solve the problem during the remaining time. The only difference between this component and running a pure IP solver is the initial solution and parameter settings provided to the solver from other relevant components. We use the solution obtained from the CP solver as a warm start for the IP solver. Moreover, we change some parameters of the IP solver based on the information provided by the IP Pre-solver. Indeed, if the IP Pre-solver provides a good lower bound (elaborated in the IP Pre-solver component), we switch off the pre-solve step in this component (e.g. setting the *Presolve* parameter in Gurobi). We also change the search strategy based on the number of constraints and variables provided by the IP Pre-solver, and a user-specified threshold, i.e. *psThr*. If the number of constraints and variables of the problem are more than *psThr*, we set the search strategy to spend more efforts on obtaining a feasible solution rather than proving optimality. We do not change the default search strategy in case a problem is not difficult to solve. In most of the modern solvers, the user can change the search strategy by a specific parameter defined therein. For example, in Gurobi solver, the user can tailor the search strategy by setting the *MIPFocus* parameter. Furthermore, using the lower bound provided by the *Decomposer* component, we enforce it on the IP solver by setting the relevant parameter accordingly (e.g. setting the *Start* parameter in Gurobi).

Symmetry Breaker: As we mentioned in Section 2, modeling the problem using indexed variables can create symmetry issues. To resolve these issues, we add lexicographic ordering constraints [25] to both CSP and IP models applied to the main variables (i.e. x_{ead} and s_{ed}, respectively). We then use the new model for both the IP and CP Solver components. In Section 6, we will mention that breaking a symmetric structure in the model is often beneficial for the solver.

Weight Balancer: In order to improve the efficiency of the IP solver during the search process, we modify the weights in the objective function due to the difficulty degree of constraints, which we elaborated in the CP Solver component. Based on this degree, if a constraint is not difficult, we impose it to the IP solver as a hard constraint. Theoretically, this process may lead to an infeasible problem. In this case, we undo the relevant change and continue the process for the rest of the constraints. Finally, we solve the new modified problem using the IP solver. This technique helps to reduce the search space, which results in a better efficiency during the search process.

Decomposer: One of the design aspect of the proposed hybrid algorithm is to generate a good lower bound for most of the benchmark instances. In this component, we decompose the problem to weekly rosters, and then we evaluate all possible shift patterns according to "forbidden shift pattern" and "request on or off" constraints (constraints 8 and 9 in Section 2). In this process, we try to find out whether there is an inevitable conflict in the model, which can be discovered before solving the problem. When there is an inherent conflict in the model according to the current data, we can calculate the associated penalty based on the objective function and consider it as a new lower bound. We do this particular evaluations for all decomposed weekly rosters in a problem. This process is elaborated in [7], where the authors try to infer a lower bound for two specific instances. However, here we use the same technique but for all decomposed weekly rosters, and not only for particular instances. Apart from this process, we also solve all decomposed weekly rosters by an IP solver to discover any further potential lower bounds. Finally, the best lower bound calculated in this component will be imported to the IP solver by setting the relevant parameter (e.g. setting the *Start* parameter in Gurobi).

6 Computational Results

To evaluate the proposed hybrid algorithm, we implemented our algorithm in Java 1.7, and used the IBM ILOG CP solver 1.7 for solving all CSPs and Gurobi 5.6 to solve all IPs. The reason to use the aforementioned solvers is that we found them easier to implement in terms of modeling, and also they suit our hybrid framework better than other software packages. In addition, we note that the benchmarks reported in [27] show that Gurobi and IBM Ilog Cplex produce very similar results for most of the instances. We run our experiments on a PC with Intel 3.4 GHz processor and 4 GB of RAM, and we used the benchmark instances introduced in Section 5. The variety in benchmark instances helps us to test and analyze our algorithm in different circumstances. To the best of our knowledge, we are the first researchers experimenting with all these instances.

For evaluation purposes, we run the hybrid algorithm for 10 minutes, and distribute 10%, 30%, and 50% of the time to IP Pre-solver, CP Solver, and IP Solver components, respectively. The rest of the time is distributed equally to other components as they require very short times in comparison. The reasons for benchmarking the proposed algorithm in 10 minutes are two-fold: i) we primarily designed the hybrid algorithm to run in a short time; ii) the selected time is in line with the testing times used by most of the algorithms reported in the literature, including the time used in the first International Nurse Rostering Competition (INRC-I) [26], and hence provides a platform for a fair comparison. Furthermore, we set the threshold parameters for the IP Pre-solver and CP Solver components, i.e. *psThr* and *cspThr*, to 10000 and 10 respectively. We also set the *numSols* parameter for the CP Solver component to 500. The design of the

algorithm is primarily deterministic, however to address the minor random behavior due to the intrinsic nature of the employed solvers, we run it three times per instance for each experiment and report average values.

We conducted two experiments to test the proposed algorithm: first, we investigate the benefit and efficiency of the Symmetry Breaker and Weight Balancer components, and how they affect the performance of the algorithm. Then, we compare the hybrid algorithm against five most recent best algorithms in the literature.

The first experiment is designed to investigate the effects of breaking symmetry and modifying weights on overall performance of the hybrid algorithm. For each test, the best objective function value, lower bound, and duality gap were recorded. The results are shown in Table 2. It should be noted that the algorithm solved instances SINTEF, MILLAR-1, and LLR in less than 3 seconds, therefore, we only report the results for the rest of the instances (six instances) in this experiment.

The results of running the hybrid algorithm using all the components are indicated as "default setting" in the first part of Table 2. For the next two parts, we remove the Symmetry Breaker and Weight Balancer components, respectively. As it can be seen, having symmetry structures in the problem worsens the duality gap for three of instances, i.e. GPOST, ORTEC01, ORTEC02, whereas it does not change the duality gaps for instances GPOSTB, Valouxis-1, and WHPP. The reason to obtain the same results is because of the limited complexity in the structure of these instances. As a result, the hybrid algorithm solved them easily compared with the other instances, although they have symmetry issues. Therefore, Symmetry Breaker component seems to improve the efficiency of the hybrid algorithm, in particular for problems with a very complex structure.

In the third part, we removed only the Weight Balancer component. The results show similar duality gaps to the second part for all the instances except a reduction for instance ORTEC02, and an increase for instance ORTEC01, which are not significant. Indeed, these instances have a particular structure that only modifying weights could not improve the performance of the algorithm. However, one can see the effect of this component when it is accompanied by the Symmetry Breaker component (Table 3). Consequently, we decided to include this component in the default setting for two reasons: first, our aim is to develop a hybrid algorithm, which is able to solve a variety of instances (in particular hard ones) successfully. Since the third instance is one of the difficult instances in our benchmark, and adding the Weight Balancer component results in a better solution, it is reasonable to keep this component in the hybrid algorithm. Second, according to our other experiments on some modified version of the current instances, and also some new generated instances, we found that generally including the Weight Balancer component leads to better-quality solutions.

Table 2. The hybrid algorithm results in different settings

Instance	Default setting			No Symmetry Breaker			No Weight Balancer		
	UB	LB	G(%)	UB	LB	G(%)	UB	LB	G(%)
GPOST	5	5	0	8	5	37.5	5	5	0
GPOSTB	5	0	100	3	0	100	5	0	100
ORTEC01	380	150	60.52	530	140	73.58	680	140	79.41
ORTEC02	370	150	59.45	570	140	75.44	340	140	58.82
Valouxis-1	20	0	100	20	0	100	20	0	100
WHPP	5	0	100	5	0	100	5	0	100

To compare the performance of the current algorithm against the stat-of-the-art algorithms reported in the literature, Table 3 shows the best-published results from: a hybrid Variable Neighborhood Search [18], a Memetic Algorithm [19], a Variable Depth Search [20], a Harmony Search Algorithm [21], a Scatter Search [22], and another hybrid Variable Neighborhood Search [23]. Unfortunately, to the best of our knowledge, we are not aware of any exact approaches and

hence we did not include any in our benchmarking. Moreover, as we mentioned in Section 5, we do not report the results of pure IP and CP solvers due to their poor performance on most of the benchmark instances. In Table 3, the column "Opt." shows the known optimal solution for the benchmark instances according to [17], where often obtained using column generation and relaxation techniques with an IP solver for a long runtime. We report the best results and their computational times (in seconds) in columns "Best" and "T", respectively. Although we run all experiments only for 10 minutes, we report the computational times for any instances the hybrid algorithm could find an optimal solution sooner. Columns "LB" and "G(%)" indicate the obtained lower bounds and duality gaps for the benchmark instances, respectively.

As it can be seen, our proposed hybrid algorithm is able to outperform other algorithms for six instances, and obtained promising results for instances ORTEC01 and ORTEC02. For instance WHPP, we could not find out any reported result in the literature other than the optimal solution mentioned in [17]. Furthermore, for instances GPOST, SINTEF, MILLAR-1, and LLR, the hybrid algorithm obtained the optimal solution in a very short time compared to other algorithms. Comparing the results of our algorithm with the Scatter Search, for instances GPOSTB, MILAAR-1, and LLR, we obtained the same results, but in a shorter time. For instances GPOST, Valouxis-1, and SINTEF, the hybrid algorithm found the best solutions, which are significantly better than the others.

It is worth noting that the proposed algorithm found the solutions reported in Table 3, while our aim of designing the hybrid algorithm was not only to find a good feasible solution, but also to achieve a better duality gap for ensuring solution quality.

Table 3. Benchmark results for our algorithm versus other algorithms reported in the literature

Instance	Opt.	The hybrid algorithm				[18]		[19]		[20]		[21]		[22]		[23]	
		Best	LB	G(%)	T(s)	Best	T(h)	Best	T(s)	Best	T(s)	Best	T(s)	Best	T(s)	Best	T(s)
GPOST	5	5	5	0	323			915	121					9	861	8	234
GPOSTB	3	5	0	100				789	95					5	791		
ORTEC01	270	380	150	60.52		541	12	535	1516	360	300	310	412	365	680		
ORTEC02	270	370	150	59.45								330	446				
Valouxis-1	20	20	0	100				560	593					100	800	160	3780
SINTEF	0	0	0	0	6			8	175					4	821		
MILLAR-1	0	0	0	0	1			100	8					0	182	0	1
WHPP	5	5	0	100													
LLR	301	301	301	0	8			305	38					301	423	314	79

7 Summary and Conclusion

This paper proposed a new systematic hybrid algorithm combining IP and CP to solve real-world Nurse Scheduling Problems. The algorithm utilized the strengths of CP to aid the IP solver to achieve better solutions. Concentrated on the problem structure, we developed some components to provide valuable problem-specific information for both IP and CP solvers so that better performance can be achieved in solving highly-constrained instances. In contrast to heuristic methods reported in the literature, we attempted to design a hybrid method to generate a good optimality gap. Moreover, we provided both CP and IP models of the problem.

We tested our algorithm on a diverse test bed of nine real-world instances from the literature. We conducted two experiments to evaluate the effectiveness of different components of the proposed algorithm, and its performance compared to some state-of-the-art algorithms. The results show that proposed algorithm is capable of obtaining competitive results.

Our future work will investigate different models for the Nurse Scheduling Problems compared with the classical model consists of indexed variables. We will also try to add a heuristic component to the proposed hybrid algorithm to improve its performance. Exploiting the problem-specific information, we will attempt to design a more sophisticated framework

doing lots of heuristics, automation, etc. to accommodate different characteristic of the problem. Finally, we are going to extend our algorithm to other scheduling problems.

References

1. M'Hallah, R. and A. Alkhabbaz, Scheduling of nurses: A case study of a Kuwaiti health care unit. Operations Research for Health Care, 2013. 2(1–2): p. 1-19.
2. Kazahaya, G., Harnessing technology to redesign labor cost management reports. Healthcare financial management: journal of the Healthcare Financial Management Association, 2005. 59(4): p. 94-100.
3. Burke, E., et al., The State of the Art of Nurse Rostering. Journal of Scheduling, 2004. 7(6): p. 441-499.
4. Ozcan, Y.A., Quantitative methods in health care management: techniques and applications. Vol. 4. 2005: John Wiley & Sons.
5. Brucker, P., R. Qu, and E. Burke, Personnel scheduling: Models and complexity. European Journal of Operational Research, 2011. 210(3): p. 467-473.
6. Karp, R.M., Reducibility among combinatorial problems. 1972: Springer.
7. Glass, C.A. and R.A. Knight, The nurse rostering problem: A critical appraisal of the problem structure. European Journal of Operational Research, 2010. 202(2): p. 379-389.
8. Maenhout, B. and M. Vanhoucke, Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. Journal of Scheduling, 2009. 13(1): p. 77-93.
9. Soto, R., et al., Modeling NRPs with Soft and Reified Constraints. AASRI Procedia, 2013. 4(0): p. 202-205.
10. Gîrbea, A., C. Suciu, and F. Şişak, Design and implementation of a fully automated planner-scheduler constraint satisfaction problem. 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), 2011: p. 477-482.
11. Lü, Z. and J.-K. Hao, Adaptive neighborhood search for nurse rostering. European Journal of Operational Research, 2012. 218(3): p. 865-876.
12. Burke, E.K., J. Li, and R. Qu, A Pareto-based search methodology for multi-objective nurse scheduling. Annals of Operations Research, 2012. 196(1): p. 91-109.
13. Brucker, P., et al., A shift sequence based approach for nurse scheduling and a new benchmark dataset. Journal of Heuristics, 2010. 16(4): p. 559-573.
14. Stølevik, M., et al., A Hybrid Approach for Solving Real-World Nurse Rostering Problems, in Principles and Practice of Constraint Programming – CP 2011, J. Lee, Editor. 2011, Springer Berlin Heidelberg. p. 85-99.
15. Valouxis, C., et al., A systematic two phase approach for the nurse rostering problem. European Journal of Operational Research, 2012. 219(2): p. 425-433.
16. Spencer, K.L., L. Ho-fung, and J.H.M. Lee. Guided complete search for nurse rostering problems in Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on. 2005.
17. Burke, E.K., et al., Problem model for nurse rostering benchmark instances. 2009: http://www.cs.nott.ac.uk/~tec/NRP/papers/ANROM.pdf [last accessed on: 2nd July 2014].
18. Burke, E.K., Curtois, T., Post, G., Qu, R., Veltman, B.: A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operation-al Research 188, 330-341 (2008)
19. Burke, E., Cowling, P., De Causmaecker, P., Berghe, G.V.: A memetic approach to the nurse rostering problem. Applied intelligence 15, 199-214 (2001)

20. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A Time Pre-defined Variable Depth Search for Nurse Rostering. (2007)
21. Hadwan, M., Ayob, M., Sabar, N.R., Qu, R.: A harmony search algorithm for nurse rostering problems. Information Sciences 233, 126-140 (2013)
22. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A scatter search methodology for the nurse rostering problem. Journal of the Operational Research Society 61, 1667-1679 (2009)
23. Métivier, J.-P., Boizumault, P., Loudni, S.: Solving Nurse Rostering Problems Using Soft Global Constraints. In: Gent, I. (ed.) Principles and Practice of Constraint Programming - CP 2009, vol. 5732, pp. 73-87. Springer Berlin Heidelberg (2009)
24. Laburthe, F., Jussien., N.: CHOCO solver documentation. (2012)
25. Beldiceanu, N., Carlsson, M., Rampon, J.-X.: Global Constraint Catalog. (2014)
26. Haspeslagh, S., Causmaecker, P.D., Stolevik, M., Schaerf, A.: INRC-First International Nurse Rostering Competition 2010. (2010)
27. Mittelmann, H.D.: Decision Tree for Optimization Software. (2014)
28. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153, 3-27 (2004)

MISTA 2015

Minimizing regular criteria in the flexible job-shop scheduling problem

A. García-León · S. Dauzère-Pérès · Y. Mati

Abstract Algorithms for minimizing other regular criteria than the makespan in the Flexible Job-shop Scheduling Problem (FJSP) are rather scarce. In this paper, we propose a local search algorithm to optimize any regular criterion in the FJSP, which makes use of the disjunctive graph model to represent schedules and search for an optimal solution. Two neighborhood structures are proposed based on moving critical operations. Efficient conditions for testing the feasibility of moves are presented and new move evaluation functions are proposed. The efficiency of the algorithm is shown on instances of the classical job-shop scheduling problem with total weighted tardiness, as well as on instances of the FJSP with other regular criteria.

1 Introduction

Scheduling is a fundamental decision within organizations regardless of their types of activities. It aims at providing the optimal schedules of the resources such as workforce and machines in order to reduce costs, improve the productivity and hence increase customer satisfaction. In many manufacturing systems, increasing the customer service is essential for companies to compete. Thus, optimizing customer-oriented criteria such as maximum tardiness, total weighted tardiness and total weighted number of tardy jobs is very important. An illustrative example can be found in the Tolima region of Colombia where, in the small and medium companies of the lithographic industry, most of the customer orders are not delivered on time.

Andrés Alberto García-León
École des Mines de Saint-Étienne, Department of Manufacturing Sciences and Logistics, CNRS UMR 6158 LIMOS, France
Universidad de Ibagué, Industrial Engineering Program, Ginnova research team, Colombia
E-mail: garcia-leon@emse.fr

Stéphane Dauzère-Pérès
École des Mines de Saint-Étienne, Department of Manufacturing Sciences and Logistics, CNRS UMR 6158 LIMOS, France
E-mail: Dauzere-peres@emse.fr

Yazid Mati
Qassim University, College of Business and Economics, Saudi Arabia
E-mail: matie@qu.edu.sa

The Job-shop Scheduling Problem (JSP) is widely studied in scheduling literature since it allows a great number of real-life applications to be modeled. The JSP assumes that the machine on which an operation must be processed is fixed. However, this assumption may not be verified in some situations where the machine assigned to an operation must be selected in a predefined candidate set. For instance, in the lithographic industry, the operation needed in the printing area can be processed by several alternative machines which have different characteristics that vary according to the technological level.

The Flexible Job-shop Scheduling Problem (FJSP) is an extension of the JSP where the operation flexibility is allowed. Different from the JSP for which a huge number of papers have been published, the FJSP needs more attention from researchers, in particular to deal with customer-based criteria. Indeed, the majority of the previous studies to solve the FJSP focus on minimizing the makespan criterion and, even for the JSP, very few papers address other criteria than the makespan. This paper is the first study, to our knowledge, that addresses the minimization of any regular criterion in the flexible job-shop scheduling problem.

The paper is organized as follows. Section 2 presents the description of the FJSP with regular criteria and introduces the disjunctive graph used to model and solve the FJSP. Section 3 highlights the main previous studies for solving the FJSP with the makespan criterion and also reviews some papers on the JSP with other regular criteria than the makespan. Section 4 presents the proposed local search algorithm and its parameters. Computational results are shown and discussed in Section 5. Section 6 provides some conclusions and future research directions.

2 Problem description and modeling

The flexible job-shop scheduling problem can be stated as follows. A set of n jobs $J = \{J_1, \ldots, J_n\}$ must be performed on a set $M = \{M_1, \ldots, M_m\}$ of m machines that are always available for processing the jobs. Each machine can only process one job at a time. A job J_i consists of n_i operations that must be performed according to a predefined sequence, called routing, which can be different from one job to another. For sake of clarity, we associate a unique integer number y ($y = 1, \ldots, \sum_i n_i$) to each operation of the jobs and denote the operation by j. The preemption of operations is not allowed, which means that an operation cannot be interrupted once started. Each job J_i has a release date r_i, a weight w_i and a due date d_i. In the FJSP, contrary to most classical shop scheduling problems, there is a flexibility when performing operations on machines, i.e. the machine needed to perform an operation j is not fixed in advance but must be selected from a subset $R_j \subseteq M$ of eligible machines. The processing time of an operation j depends on the selected machine in R_j. These processing times are non-negative integer, known and deterministic. For sake of clarity of notations, the processing time of operation j will be denoted by p_j regardless of the assigned machine to operation j.

To obtain a feasible schedule of the FJSP, two main decisions have to be made, namely assignment and sequencing. The assignment decision consists of selecting, for each operation j, the machine that will perform the operation from the subset R_j while the sequencing decision deals with obtaining a sequence of operations on each of the selected machines. The FJSP aims at obtaining a feasible schedule that minimizes a given objective function. In this paper, contrary to most of the literature on solving

the FJSP, we consider the minimization of any regular criterion. A criterion is said to be regular if it is an increasing function of the completion times C_i of the jobs J_i. Among the most widely used regular criteria for scheduling problems are the makespan $C_{\max}$, the maximum tardiness $T_{\max} = \max_i T_i$ where $T_i = d_i - C_i$ if $d_i > C_i$ and 0 otherwise, the total weighted completion time $\sum w_i C_i$, the total weighted tardiness $\sum w_i T_i$, and the total weighted number of late jobs $\sum w_i U_i$ where $U_i = 1$ if $d_i > C_i$ and 0 otherwise.

When the assignment of machines to operations is fixed, the FJSP becomes the job-shop problem with possible repetition of machines for different operations of the same job. In this case, an extension of the disjunctive graph originally developed in Roy and Sussmann (1964) can be nicely used to represent the problem. This extension, that has been used in Mati et al. (2011) allows representing the job-shop problem with any regular criteria. The graph is noted $G = (V, A, E)$, where V is the set of nodes, A is the set of conjunctive arcs and E is the set of disjunctive arcs. The nodes in the set V represent operations of jobs, plus a dummy node 0 that represents the start of each job, and n dummy nodes ϕ_i, where ϕ_i represents the completion time of job J_i. The set A contains conjunctive arcs that connect two consecutive operations on the routing of jobs, the node 0 and every first operation of each job, and the last operation of each job J_i to a dummy node ϕ_i. The set $E = \cup_k E_k$ $(k = 1, \ldots, m)$ contains disjunctive arcs between every pair of operations assigned to the same machine M_k. The arc from 0 to the first operation of a job J_i has a length equal to the release date r_i of J_i. Any remaining conjunctive or disjunctive arc has a length equal to the processing time of the operation from which it starts.

A selection, which corresponds to a schedule of the FJSP for a given assignment, is obtained by fixing a direction to each disjunctive arc in E. The selection is feasible if the induced graph is acyclic. The graph contains many redundant arcs that must be removed to ensure that every node x has at most one predecessor and one successor on the machine that performs x. We denote by pr_x (resp. fr_x) the node preceding (resp. following) x on the routing, and by ps_x (resp. fs_x) the node preceding (resp. following) x on the sequence of the machine assigned to x.

The starting time $h_x = L(0, x)$ of a node x, called head, corresponds to the length of a longest path from 0 to x. Thus, the completion time C_i of a job J_i is equal to h_{ϕ_i}. The tail q_x^i from a node x to a dummy nodes ϕ_i is equal to $L(x, \phi_i) - p_x$ if a path exists from x to ϕ_i and $-\infty$ otherwise. A path from 0 to ϕ_i is called critical if its length is equal to C_i, and every node x belonging to this critical path is critical according to job J_i. A critical node x for job J_i satisfies $h_x + p_x + q_x^i = C_i$. An arc belonging to the critical path from 0 to ϕ_i is called critical if it connects two operations x and $y \neq fr_x$ assigned to the same machine. Note that an arc may be critical for several jobs. The level of a node x in G, which is the maximum number of arcs in a path from 0 to x, is denoted by l_x. Having obtained the heads of the nodes of the graph, the objective function of the feasible schedule represented by the selection can be determined in $O(n)$ from the starting times of the dummy nodes ϕ_i. For instance, the makespan is obtained using the formula $C_{\max} = \max C_i$ $(i = 1, \ldots, n)$.

3 Literature review

The aim of this section is not to provide a complete list of the contributions on the FJSP, but rather to highlight the papers closely related to our work, in particular

those dealing with regular criteria other than the makespan. For the JSP, a state-of-the-art is provided in Jain and Meeran (1999) and there has been an increased interest during the recent years but still most of the papers consider the minimization of the makespan. Among the approaches that form the state-of-the-art for the JSP with $\sum w_i T_i$ are genetic algorithms Essafi et al. (2008), simulated annealing Zhang and Wu (2011), shifting bottleneck Pinedo and Singer (1999), hybrid methods Bülbül (2011), and large step random walk Kreipl (2000). A general approach that deals with any regular criterion is proposed in Mati et al. (2011), which is based on a new and efficient evaluation of swap moves. This evaluation has been extended in Braune et al. (2013) to deal with insertion moves.

Methods for solving the FJSP are not as abundant as for solving the JSP, but there has been a significant increase of papers in the last ten years. Some papers analyze the complexity of special cases of the FJSP Brucker and Schlie (1990) and Mati and Xie (2004). The majority of the methods for solving the FJSP are heuristic methods that can be hierarchical Brandimarte (1993) or integrated approaches Dauzère-Pérès and Paulli (1997). In the former type of approaches, assignment and sequencing decisions are separated whereas, in the second type, assignment and sequencing decisions are considered simultaneously. Most of the state-of-the-art approaches for the FJSP are metaheuristics that make use of the disjunctive graph model. In Hurink et al. (1994), a tabu search algorithm that uses the block concept is developed for the case where the processing times of operations are independent of the assigned machines. Another tabu search algorithm is developed in Dauzère-Pérès and Paulli (1997) based on moving operations on the critical path. New results on testing the feasibility of insertion moves and evaluating the quality of moves are developed. The tabu search in Mastrolilli and Gambardella (2000) uses the insertion moves but new conditions for ensuring the feasibility of moves are developed as well as new move estimations. An integrated greedy heuristic that schedules jobs iteratively is developed in Mati et al. (2001). Two algorithms, namely hybrid harmony search and large neighborhood search, are developed in Yuan and Xu (2013). An analysis of four mathematical formulations of the FJSP is presented in Demir and İşleyen (2013). An artificial bee colony algorithm is proposed in Wang et al. (2012), and a variant of the climbing discrepancy search approach is described in Hmida et al. (2010). The FJSP with additional constraints such as limited resource constraints is also investigated Rajkumar et al. (2011). It is worth noting that many authors have also concentrated on solving the FJSP with multi-objective functions (see for instance Jia and Hu (2014)).

Note again that the majority of the papers on FJSP are dedicated to the makespan. An important contribution of this paper is the development of an approach for optimizing any regular criterion in the FJSP. This approach is a local search method and its parameters are described in the next section.

4 A Tabu thresholding based local search algorithm

This section starts with an overview of the proposed tabu thresholding algorithm for the FJSP. We then focus on describing the most important characteristics of the algorithm that are the main contributions of the paper.

4.1 Overview of the algorithm

The disjunctive graph model introduced in Section 2 is used to represent feasible solutions of the FJSP and to explore the solution space to search for an optimal solution. Using this graph, a tabu thresholding algorithm is proposed that consists of two iterative steps: Improvement and diversification. The former is a steepest descent that performs iterative improvements until a local optimum is reached. At each iteration of this step, a set of neighbor solutions is generated using a neighborhood structure that consists of moving a critical operation i between two operations j and k. After testing the feasibility of moves and estimating the value of the criterion, the best move in the neighborhood is determined. When the improvement step reaches a local optimum, the diversification step starts by selecting a random number $b \in [t_{min}, t_{max}]$ that represents the maximum number of moves that will be performed. During this step, a critical arc (x, y) is randomly selected and a move is randomly chosen among the resequencing of x, the resequencing of y, the reassignment of x or the reassignment of y. If the selected move is feasible, the algorithm advances to the next iteration, otherwise the above process is repeated until a feasible move is obtained. If a new best value of the criterion is obtained in the diversification step, the search returns to the improvement step, otherwise it continues until b iterations are performed. Experimental analysis showed that the value of $b \in [4, 10]$ allows the search to escape from local optima. The remaining important characteristics of our tabu thresholding algorithm, i.e. the initial solution, the neighborhood structure, the feasibility test and the evaluation of moves, are described in the following subsections.

4.2 Initial solution

The initial solution is iteratively constructed by examining an operation at each step. The jobs are sorted in non-decreasing order of their weights, and the ties are broken using due dates d_i of jobs and then the average processing times $\sum_{j=1}^{n_i} \frac{1}{|R_j|} \sum_{a \in R_j} p_j$. Note that the weights and due dates are not considered if the considered criterion does not consider them. At each step, a set of eligible operations is defined, which initially contains the first operation of each job. Operations of this set are examined according to the established order of jobs to which they belong. For a given operation x and for each machine $M_k \in R_x$, the time t_k at which the machine finishes on its previous operation v (if it exists) is calculated. Thus, operation x is assigned to the machine M_a with the minimum value $t_k + p_x$. An arc (v, x) is then added to the graph and the procedure continues until all operations of jobs are considered.

4.3 Neighborhood structures

The neighborhood structure is one of the most important parameters of local search methods. It allows generating a solution by performing small perturbations of the current one. In our paper, the perturbation consists in choosing an operation x currently processed on a machine M_x and moving it between two operations j and k assigned to a machine belonging to the subset R_x of candidate machines of operation x. Note that if the current machine of x is the same for j and k, the perturbation is a resequencing move, otherwise it is a reassignment move. The move is performed in the following

way: The arcs (ps_x, x) with $ps_x \neq 0$ and (x, fs_x) with $fs_x \neq 0$ are first deleted, and the arc (ps_x, fs_x), with $ps_x \neq fs_x \neq 0$ is added; then the arc (j, k) is deleted and the arcs (j, x) with $j \neq 0$ and (x, k) with $k \neq 0$ are added. Accordingly, two neighborhood structures, denoted N_1 and N_2, are proposed. In N_1, the operation x to be moved must be critical for at least one job. The neighborhood $N_2 \subset N_1$ is a slight modification of N_1 by focusing on the first and last operations of consecutive operations on the same machine on a critical path (notion of "block" in classical job-shop scheduling). For both neighborhoods N_1 and N_2, only feasible moves are considered as described in the next subsection. It is worth mentioning, based on the results in Dauzère-Pérès and Paulli (1997), that the two neighborhoods N_1 and N_2 are connected.

4.4 Feasibility of moves

Note that, if a path from x to y exists in the graph, then $h_y \geq h_x + p_x$ and $q_x \geq q_y + p_y$. This property is the basic idea for testing the feasibility of moves. Indeed, after moving an operation x between operations j and k, the resulting graph will not contain circuits whenever the two following statements are true: (i) No path from fr_x to j exists and (ii) No path from k to pr_x exists. In Dauzère-Pérès and Paulli (1997), the two statements are verified using sufficient conditions that only use the heads while, in Mastrolilli and Gambardella (2000), the first (resp. second) statement is only verified using heads (resp. tails). In this paper, we use a combination of conditions based on heads and tails leading to the property below.

Moving x between j and k is feasible if the two following conditions are satisfied: (i) $(h_j < h_{fr_x} + p_{fr_x}) \vee (q_{fr_x} < q_j + p_j)$ and (ii) $(h_k + p_k > h_{pr_x}) \vee (q_{pr_x} + p_{pr_x} > q_k)$.

We will show through computational tests in Section 5.1 that the new combined conditions are more efficient to prove the feasibility of moves than using heads and tails alone.

4.5 Move evaluation

We propose a new estimation function for evaluating the criterion of the neighbor solutions without performing the moves. This estimation has several properties: It does not distinguish between reassignment and resequencing moves, deals with any regular criterion, is efficient and fast, and provides a lower bound of the criterion if the move is performed. Let x be an operation sequenced between operations s and t which is moved between operations j and k. Since regular criteria are considered in this paper, we need to evaluate the new completion times $\widetilde{C}_i$ of the dummy nodes ϕ_i if the move is performed.

The proposed estimation function extends the one proposed in Dauzère-Pérès and Paulli (1997) by using the idea of Mati et al. (2011) for the classical job-shop scheduling problem. Indeed, rather than only estimating the length of the paths that go through the node x, we estimate the length of two suitably selected groups of paths. The first group contains the new paths created after the move is performed while the second group consists of a subset of paths that are available in the current and the new graphs. To do so, we distinguish the two cases below.

- **Case 1:** $l_x \leq l_j$

  We focus in this case on estimating the length of the new created paths that use the

arcs (s,t) or (j,fr_j) or that go through nodes x, which are evaluated respectively using L_1, L_2 and L_3; and the length L_4 of a subsets of paths that already exist in the graph, which are paths that use a node w in the same partition of i (Mati et al. (2011)). Thus, the estimated value of $\widetilde{C}_i$ is calculated as follows:

$$\widetilde{C}_i = \begin{cases} C_i & \text{if } q_x^i = q_k^i = -\infty \\ \\ \max\{L_1, L_2, L_3, L_4\} & \text{otherwise} \end{cases}$$

with

$$\begin{gathered} L_1 = h_s + p_s + p_t + q_t^i \\ L_2 = \tilde{h}_j + p_j + p_{fr_j} + q_{fr_j}^i \\ L_3 = \tilde{h}_x + p_x + \max\{p_{fr_x} + q_{fr_x}^i, p_k + q_k^i\} \\ L_4 = \max_{\omega \in S_1}\{h_\omega + p_\omega + q_\omega^i\} \end{gathered}$$

where p_x is the processing time of x after the move and

$$\begin{gathered} \tilde{h}_j = h_j - h_t + \max\{h_s + p_s, h_{pr_t} + p_{pr_t}\} \\ \tilde{h}_x = \max\{\tilde{h}_j + p_j, h_{pr_x} + p_{pr_x}\} \\ S_1 = \{\omega \neq i / l_\omega = l_i\} \end{gathered}$$

It can be proved that the quantities h_s, q_k^i, $q_{fr_x}^i$ and $q_{fr_j}^i$ remain the same after performing the move. Hence, $\widetilde{C}_i$ is a lower bound of the completion time of ϕ_i in the new graph.

– **Case 2:** $l_x > l_j$

Again we investigate the new created paths that use the arcs (s,t) or (x,fr_i) or that go through node k, using respectively the expressions L_1, L_2 and L_3; and a subset of paths that use a node w in the same partition of i, which are evaluated by L_4. Thus, the estimated value of $\widetilde{C}_i$ is calculated as follows:

$$\widetilde{C}_i = \begin{cases} C_i & \text{if } q_x^i = q_k^i = -\infty \\ \\ \max\{L_1, L_2, L_3, L_4\} & \text{otherwise} \end{cases}$$

with

$$\begin{gathered} L_1 = h_s + p_s + p_t + q_t^i \\ L_2 = \tilde{h}_x + p_x + p_{fr_x} + q_{fr_x}^i \\ L_3 = \tilde{h}_k + p_k + \tilde{q}_k^i \\ L_4 = \max_{\omega \in S_1}\{h_\omega + p_\omega + q_\omega^i\} \end{gathered}$$

where p_x is the processing time of x after the move and

$$\begin{gathered} \tilde{q}_k^i = q_k^i - q_s^i + \max\{p_{fr_s} + q_{fr_s}^i, p_t + q_t^i\} \\ \tilde{h}_x = \max\{h_j + p_j, h_{pr_x} + p_{pr_x}\} \\ \tilde{h}_k = \max\{\tilde{h}_x + p_x, h_{pr_k} + p_{pr_k}\} \\ S_1 = \{\omega \neq i / l_\omega = l_i\} \end{gathered}$$

It can be shown that the quantities h_j, the q_t^i and $q_{fr_i}^i$ do not change after performing the move, which means that $\widetilde{C}_i$ is a lower bound of the completion time of ϕ_i in the new graph.

5 Computational experiments

To validate and evaluate our approach, computational experiments are conducted on the feasibility tests and for various criteria: The makespan for the FJSP, the Total Weighted Tardiness (TWT) for the JSP and the Total Flow Time (TFT) and the maximum tardiness (T_{max}) for the FJSP. The latter instances can be used as benchmarks for future research. Our algorithm was developed in Java and the experiments were conducted on PC with 3.40 GHz and 8GB RAM. Each best result in Tables 2 4 and 5 is provided with the computing time required to obtain the result. Although many best solutions are obtained rather fast, our future work aims at reducing the computing times.

5.1 Feasibility tests

To study the efficiency of the feasibility tests presented in Section 4.4, we want to analyze how the test HT (based on heads and tails) increases the number of possible moves compared to the tests H (heads alone) and T (tails alone). By considering the total number of feasible moves provided by HT, it is possible to determine the percentage of moves allowed by the two other tests (H and T). Table 1 shows the percentage of efficiency of each test for the two types of moves (resequencing and reassignment) for the test H, T and HT using the v-data instances of Hurink et al. (1994) and when the makespan is minimized. Additionally, improving and non-improving moves are separated. Only improving moves lead to a better value of the objective function. The values in Table 1 correspond to the percentage of success of each test. For instance, on average for the instances from la01 to la05, there are 27.73% of improving moves when resequencing with HT, 21.92% with H and 26.41% with T.

It is important to highlight that the percentage of improving and non-improving moves depends on how close the current solution is to a (globally or locally) optimal solution. Note that the feasibility test T allows for more improving moves than the feasibility test H. The difference between the improving moves obtained with HT and T is lower than 3%, whereas the difference for non-improving moves varies between 8 and 10%.

5.2 Minimizing the makespan for the FJSP

To minimize the makespan, the two neighborhood structures N_1 and N_2 were tested on the data sets e-data, r-data and v-data of Hurink et al. (1994). Table 2 is structured as follows: Column "Best known value" provides the best known value for the instance, columns "N_1" and "N_2" provide the makespan obtained with neighborhoods N_1 and N_2 respectively, where * indicates that the best known value is found and the number in parentheses is the computational time in seconds to find the best solution.

The results in Table 2 show that our approach obtains the best known value in many instances, and the gap is small when the best known solution is not found. We believe it is still possible to reduce computational times by decreasing the number of evaluations and improving the structure of the algorithm. In general, using N_2 leads to

Table 1 Average efficiency of the different feasibility tests

Type of moves	Instances	m x n	Improving moves			Non-improving moves		
			H (%)	T (%)	HT (%)	H (%)	T (%)	HT (%)
Resequencing	la01-la05	5x10	21.92	26.41	27.73	53.16	61.08	72.27
	la06-la10	5x15	29.87	37.05	38.35	46.04	54.25	61.65
	la11-la15	5x20	37.19	45.78	47.03	39.43	46.94	52.97
	la16-la20	10x10	17.39	21.48	24.65	54.28	56.10	75.35
	la21-la25	10x15	14.53	17.02	18.71	63.97	67.86	81.29
	la26-la30	10x20	17.85	20.36	21.74	63.70	68.08	78.26
	la31-la35	10x30	27.15	29.71	30.75	57.45	62.21	69.25
	la36-la40	15x15	13.93	16.97	19.70	61.03	61.20	80.30
Reassignment	la01-la05	5x10	18.07	20.84	21.72	62.52	72.54	78.28
	la06-la10	5x15	19.04	23.70	24.66	59.51	70.84	75.34
	la11-la15	5x20	21.15	25.84	26.79	59.62	69.30	73.21
	la16-la20	10x10	23.20	26.19	28.14	59.20	62.77	71.86
	la21-la25	10x15	16.92	18.61	19.81	69.88	73.58	80.19
	la26-la30	10x20	16.02	17.71	18.70	76.07	81.20	81.30
	la31-la35	10x30	15.91	17.27	17.98	71.41	78.34	82.02
	la36-la40	15x15	21.75	23.04	25.05	63.86	66.01	74.95

better solutions than N_1, although the dominance is not very strong. This is because less and more promising moves are explored in N_2, which is particularly relevant for the C_{max} criterion. This illustrates that the notion of "block" used in the job-shop scheduling literature is relevant.

Table 2 Results for the makespan on the instances of Hurink et al. (1994) (computational times to obtain the best solution are in parentheses)

Inst	Best	e-data		Best	r-data		Best	v-data	
	known value	N_1	N_2	known value	N_1	N_2	known value	N_1	N_2
la01	609	*(7)	*(18)	570	573(134)	572(124)	570	*(73)	*(107)
la02	655	*(87)	*(12)	529	535(85)	531(87)	529	*(258)	*(310)
la03	550	563(85)	554(94)	477	478(168)	478(175)	477	479(245)	479(237)
la04	568	576(42)	*(57)	502	507(145)	504(187)	502	*(77)	*(96)
la05	503	*(97)	*(25)	457	458(189)	458(227)	457	458(143)	458(154)
la06	833	*(180)	*(190)	799	*(357)	*(402)	799	*(427)	*(457)
la07	762	766(114)	765(125)	749	751(615)	751(578)	749	750(124)	750(141)
la08	845	*(154)	*(127)	765	768(446)	766(473)	765	766(380)	766(351)
la09	878	*(361)	*(327)	853	854(687)	854(657)	853	*(319)	*(289)
la10	866	*(347)	*(296)	804	805(580)	805(561)	804	805(147)	805(173)
la11	1103	*(458)	1104(370)	1071	*(304)	*(412)	1071	*(529)	*(614)
la12	960	*(387)	*(438)	936	*(614)	*(573)	936	*(452)	*(497)
la13	1053	*(814)	*(741)	1038	*(661)	*(578)	1038	*(289)	*(257)
la14	1123	*(547)	*(589)	1070	*(875)	*(784)	1070	*(324)	*(315)
la15	1111	1118(875)	*(967)	1089	1090(1320)	1090(1278)	1089	1090(275)	1090(287)
la16	892	*(168)	*(177)	717	*(327)	*(364)	717	*(328)	*(345)
la17	707	*(147)	*(176)	646	*(412)	*(387)	646	*(264)	*(217)
la18	842	843(54)	*(89)	666	669(514)	669(496)	663	*(224)	*(195)
la19	796	*(394)	*(415)	700	703(591)	703(578)	617	*(512)	*(394)
la20	857	*(187)	*(227)	756	*(293)	*(320)	756	*(187)	*(149)
la21	1009	1037(712)	1018(621)	835	856(635)	860(514)	804	815(514)	815(498)
la22	880	883(224)	888(145)	760	782(745)	782(714)	736	743(275)	743(264)
la23	950	*(680)	954(430)	842	860(479)	857(578)	815	819(425)	819(408)
la24	908	914(378)	909(419)	808	823(714)	823(679)	775	788(519)	788(473)
la25	936	945(521)	947(254)	791	806(724)	806(701)	756	764(647)	764(625)
la26	1107	1137(432)	1138(314)	1061	1077(541)	1075(841)	1052	1057(415)	1057(278)
la27	1181	1205(513)	1208(378)	1091	1108(812)	1108(758)	1084	1090(315)	1090(248)
la28	1142	1161(325)	1160(469)	1080	1097(857)	1100(785)	1069	1075(524)	1075(457)
la29	1111	1151(478)	1145(680)	998	1010(814)	1010(726)	993	999(790)	999(783)
la30	1195	1238(452)	1239(217)	1078	1091(614)	1091(527)	1069	1074(487)	1074(452)
la31	1538	1571(478)	1566(785)	1520	1534(671)	1529(874)	1520	1522(394)	1522(370)
la32	1698	1704(752)	*(955)	1659	1670(158)	1670(108)	1658	1660(609)	1660(521)
la33	1547	1565(658)	1580(430)	1499	1518(189)	1511(278)	1497	1500(287)	1500(312)
la34	1599	1652(549)	1630(746)	1535	1554(612)	1546(578)	1535	1537(745)	1537(814)
la35	1736	*(479)	*(501)	1550	1570(279)	1558(327)	1549	1551(357)	1551(381)
la36	1160	1169(189)	1162(289)	1030	1041(852)	1034(925)	948	*(499)	*(479)
la37	1397	*(458)	1398(267)	1077	1088(529)	1088(444)	986	*(571)	*(524)
la38	1141	1151(688)	1158(715)	962	969(785)	969(743)	943	*(591)	*(587)
la39	1184	1186(679)	1187(389)	1018	1027(557)	1034(549)	922	*(287)	*(221)
la40	1144	1155(218)	1146(267)	970	981(157)	974(478)	955	*(292)	*(267)
mt06	55	*(27)	*(5)	47	*(35)	*(37)	47	*(48)	*(12)
mt10	871	*(104)	*(87)	686	*(24)	*(37)	655	*(107)	*(91)
mt20	1088	1091(799)	1105(347)	1022	*(879)	*(891)	1022	*(468)	*(457)

5.3 Minimizing the Total Weighted Tardiness (TWT) for the JSP

For the JSP, the instances in Singer and Pinedo (1998) are used to minimize the TWT. The numerical experiments consisted in running our algorithm 10 times for each instance and with each neighborhood structure. For each instance, 600 000 evaluations were performed, and the time of each run is 380 seconds. Table 3 shows the results of the experiments and is structured as Table 2. Note that many best known solutions are reached with our approach. Note that neither N_1 nor N_2 is dominating.

Table 3 Results for the TWT on the instances of Singer and Pinedo (1998) (380 seconds for each instance)

Inst	Best	f=1.30		Best	f=1.50		Best	f=1.60	
	known value	N_1	N_2	known value	N_1	N_2	known value	N_1	N_2
abz5	1403	*	*	69	*	*	0	*	*
abz6	436	*	*	0	*	*	0	*	*
la16	1169	*	*	166	*	*	0	*	*
la17	899	*	*	260	*	*	65	*	*
la18	929	*	*	34	*	*	0	*	*
la19	948	*	*	21	*	*	0	*	*
la20	805	*	*	0	*	*	0	*	*
la21	463	*	*	0	*	*	0	*	*
la22	1064	1072	1072	196	*	*	0	*	*
la23	835	*	*	2	*	*	0	*	*
la24	835	*	*	82	*	94	0	*	*
MT10	1363	*	*	394	*	*	141	*	*
ORB1	2568	2618	**2605**	1098	1141	1141	566	604	**569**
ORB2	1408	*	1434	292	*	*	44	52	52
ORB3	2111	2199	2199	918	*	*	422	**426**	526
ORB4	1623	1694	*	358	505	546	66	74	*
ORB5	1593	1736	1736	405	443	433	163	181	181
ORB6	1790	*	*	426	*	*	28	*	*
ORB7	590	*	*	50	*	*	0	*	*
ORB8	2429	2494	**2439**	1023	*	1079	621	646	646
ORB9	1316	*	*	297	*	*	66	*	*
ORB10	1679	*	*	346	372	372	76	**95**	128

5.4 Minimizing T_{max} and the Total Flow Time (TFT) for the FJSP

To create instances for minimizing T_{max} and the TFT for the FJSP, we used the instances of Hurink et al. Hurink et al. (1994). To generate due dates for T_{max}, we introduced a parameter f, fixed at 1.3. The due date d_i of job J_i is determined by multiplying f by the average processing time of jobs.

The results obtained for T_{max} with our two neighborhood structures can be found in Table 4, where a bold number indicates the best value and the number in parentheses specifies the computing time to obtain the best solution. It can be noted that most of the best results are obtained with N_2. Hence, N_2 is not penalized by the fact that the search space from a given solution is smaller than the search space using N_1. Again, this illustrates that the notion of "block" is often relevant. For some instances in the r-data and v-data sets (la16-la20, la36-la40, mt06 and mt10), our approach obtains the optimal solutions.

The results in Table 5 show that the neighborhood N_2 strongly dominates the neighborhood N_1. In all instances, using N_2 is at least as effective, and the gap with

Table 4 Results for T_{max} on the instances of Hurink et al. (1994) (computational times to obtain the best solution are in parentheses)

Inst	e-data		r-data		v-data	
	N_1	N_2	N_1	N_2	N_1	N_2
la01	**267**(5)	**267**(75)	**129**(21)	**129**(32)	**126**(110)	**126**(124)
la02	**177**(27)	**177**(8)	**113**(28)	**113**(32)	**108**(53)	**108**(65)
la03	**187**(7)	**187**(29)	**123**(74)	**123**(10)	**109**(85)	**109**(54)
la04	249(104)	**246**(121)	121(87)	**120**(51)	120(125)	**119**(167)
la05	**210**(5)	**210**(18)	**131**(73)	**131**(95)	127(89)	**125**(92)
la06	**439**(57)	**439**(11)	353(154)	**352**(168)	358(547)	**356**(598)
la07	**392**(214)	**392**(199)	**355**(67)	**355**(44)	352(24)	**350**(26)
la08	**425**(49)	**425**(36)	**351**(149)	**351**(154)	337(48)	**336**(56)
la09	**441**(32)	**441**(18)	**400**(314)	**400**(208)	**394**(87)	**394**(46)
la10	**475**(8)	**475**(69)	348(208)	**346**(489)	340(239)	**339**(520)
la11	697(589)	**696**(944)	**606**(81)	**606**(95)	**598**(26)	**598**(89)
la12	**585**(15)	**585**(42)	519(647)	**517**(664)	514(134)	**512**(179)
la13	**624**(50)	**624**(102)	597(846)	**595**(981)	570(710)	**569**(706)
la14	**696**(70)	**696**(78)	598(628)	**597**(604)	**595**(215)	**595**(190)
la15	674(327)	**666**(125)	640(429)	**639**(650)	630(654)	**629**(1024)
la16	**95**(40)	**95**(54)	**0**(28)	**0**(49)	**0**(41)	**0**(87)
la17	**70**(245)	**70**(127)	**0**(20)	**0**(68)	**0**(28)	**0**(37)
la18	124(89)	**120**(130)	**0**(45)	**0**(16)	**0**(47)	**0**(30)
la19	**65**(53)	**65**(98)	**0**(120)	**0**(133)	**0**(48)	**0**(27)
la20	**93**(31)	**93**(59)	**0**(25)	**0**(37)	**0**(79)	**0**(60)
la21	244(198)	**235**(294)	115(615)	**110**(723)	62(492)	**57**(653)
la22	287(324)	**280**(425)	121(689)	**116**(883)	65(875)	**62**(994)
la23	**225**(70)	**225**(89)	132(548)	**127**(571)	88(317)	**86**(478)
la24	218(294)	**209**(374)	120(498)	**117**(666)	67(547)	**62**(684)
la25	244(89)	**238**(67)	107(418)	**102**(464)	**43**(745)	**43**(815)
la26	396(498)	**394**(522)	324(548)	**321**(643)	284(669)	**278**(762)
la27	451(247)	**450**(285)	359(477)	**351**(515)	318(501)	**314**(574)
la28	**457**(734)	**457**(857)	344(378)	**338**(367)	295(715)	**291**(750)
la29	403(486)	**396**(693)	307(587)	**297**(642)	263(587)	**259**(601)
la30	433(805)	**416**(752)	306(548)	**303**(687)	256(874)	**251**(918)
la31	851(678)	**833**(713)	791(654)	**763**(728)	734(614)	**733**(789)
la32	936(842)	**916**(861)	874(894)	**865**(987)	**843**(1098)	**843**(1258)
la33	891(785)	**883**(726)	**778**(962)	**778**(1154)	**740**(905)	**738**(806)
la34	917(810)	**899**(805)	825(354)	**796**(338)	**776**(1025)	**776**(727)
la35	**1019**(280)	**1019**(486)	841(1240)	**819**(1041)	787(985)	**778**(1184)
la36	130(145)	**123**(136)	**0**(425)	**0**(143)	**0**(289)	**0**(239)
la37	**263**(189)	**263**(120)	**0**(124)	**0**(99)	**0**(67)	**0**(40)
la38	186(214)	**182**(295)	**0**(108)	**0**(129)	**0**(248)	**0**(95)
la39	**118**(489)	**118**(370)	**0**(155)	**0**(297)	**0**(215)	**0**(73)
la40	102(638)	**99**(821)	**0**(148)	**0**(291)	**0**(40)	**0**(99)
mt06	**9**(1)	**9**(62)	**0**(3)	**0**(25)	**0**(1)	**0**(12)
mt10	190(62)	**187**(32)	**0**(37)	**0**(85)	**0**(16)	**0**(58)
mt20	**692**(263)	**692**(325)	589(295)	**588**(283)	581(524)	**580**(1073)

the solution obtained using N_1 is sometimes significant. This again could be explained by the fact that many non-relevant moves in N_1 are not considered in N_2. Moreover, the evaluation of moves in N_2 is probably more effective and closer to the actual value after the move is performed, thus leading to a steepest descent in the improvement phase.

6 Conclusions

This paper presented a local search algorithm for solving the flexible job-shop scheduling problem with any regular criterion. Using the properties of the disjunctive graph model, we developed two neighborhood structures N_1 and N_2 that consist of moving a critical operation between two positions on the sequence of machines belonging to its candidate set. Feasibility test conditions as well as new move evaluation functions are proposed. The computational results show the efficiency of the feasibility test con-

Table 5 Results for TFT on the instances of Hurink et al. (1994) (computational times to obtain the best solution are in parentheses)

Inst	e-data		r-data		v-data	
	N_1	N_2	N_1	N_2	N_1	N_2
la01	**4567**(58)	**4567**(40)	4341(785)	**4339**(764)	4172(742)	**4139**(854)
la02	4259(254)	**4235**(320)	**3962**(542)	**3962**(458)	**3850**(985)	**3840**(1427)
la03	**3931**(62)	**3931**(128)	**3564**(578)	**3564**(758)	3510(378)	**3500**(475)
la04	**4156**(192)	**4156**(180)	3795(687)	**3791**(821)	3796(524)	**3768**(478)
la05	**3796**(157)	**3796**(52)	3512(389)	**3504**(548)	3466(584)	**3451**(794)
la06	**8425**(441)	8425(458)	7965(358)	**7912**(248)	7781(857)	**7750**(945)
la07	**7812**(198)	**7812**(180)	7498(657)	**7441**(594)	7378(299)	**7298**(117)
la08	**8071**(245)	**8017**(34)	7628(456)	**7583**(847)	7418(627)	**7388**(901)
la09	8930(269)	**8848**(305)	8583(99)	**8499**(127)	8324(378)	**8262**(473)
la10	8496(359)	**8461**(418)	7908(358)	**7830**(319)	7690(459)	**7639**(662)
la11	14136(710)	**13902**(702)	13119(259)	**13094**(119)	12886(504)	**12740**(682)
la12	**11779**(917)	**11779**(1012)	11205(367)	**11082**(689)	10898(995)	**10781**(1748)
la13	13184(269)	**13069**(158)	12701(994)	**12673**(1884)	**12343**(876)	**12343**(991)
la14	**14032**(789)	**14032**(658)	13093(1589)	**12987**(2272)	12794(1289)	**12711**(1680)
la15	**14275**(458)	**14275**(317)	13955(587)	13607(470)	13309(2547)	**13190**(4401)
la16	**7202**(128)	7202(657)	6216(513)	**6141**(438)	5595(89)	**5549**(70)
la17	**6148**(80)	**6148**(139)	5419(611)	**5402**(794)	4917(885)	**4879**(976)
la18	6935(669)	**6895**(727)	6001(587)	**5971**(459)	5462(1052)	**5384**(1162)
la19	6963(785)	**6939**(265)	6408(398)	**6306**(282)	5664(2385)	**5581**(2482)
la20	7132(297)	**7115**(363)	6418(319)	**6377**(267)	5742(920)	**5677**(1370)
la21	12183(1015)	**12098**(996)	11135(2614)	**11036**(2457)	10595(2230)	**10504**(2160)
la22	11447(899)	**11387**(787)	10402(985)	**10279**(775)	9937(1258)	**9735**(3151)
la23	12345(568)	**12343**(684)	11498(1879)	**11359**(1910)	10975(2547)	**10823**(3576)
la24	11645(652)	**11623**(587)	**10830**(2600)	**10830**(2584)	10283(2147)	**10204**(3060)
la25	11521(687)	**11456**(267)	10450(2101)	**10367**(2372)	9883(4257)	**9753**(4593)
la26	18802(1970)	**18459**(2476)	18224(1489)	**17798**(1784)	17054(2247)	**16694**(10974)
la27	20045(1957)	**19850**(1528)	18517(997)	**18415**(653)	17780(5471)	**17419**(9861)
la28	19717(957)	**19414**(864)	18296(2004)	**18264**(2506)	17527(1985)	**17141**(5911)
la29	17597(1261)	**17498**(915)	16922(993)	**16465**(1634)	15832(2489)	**15527**(3726)
la30	19181(1478)	**18882**(1677)	17847(4589)	**17778**(7956)	16672(2578)	**16573**(5108)
la31	38145(1278)	**36912**(1341)	36819(2499)	**36259**(9162)	35422(2762)	**34479**(4575)
la32	41544(3527)	**41189**(3679)	40227(5427)	**39298**(9034)	38468(1989)	**38312**(8153)
la33	37416(3247)	**36697**(4125)	35754(1489)	**35104**(6021)	34493(3312)	**33884**(3484)
la34	39062(2180)	**38338**(2748)	37820(8402)	**37297**(11537)	36639(2697)	**36032**(3896)
la35	39089(5107)	**38762**(5746)	37145(6278)	36853(9755)	35923(1845)	**35211**(2880)
la36	15976(1025)	**15714**(759)	14378(776)	**14216**(1212)	12608(1854)	**12462**(3141)
la37	17453(665)	**17305**(897)	15074(1856)	**14960**(2174)	13487(1007)	**13428**(1255)
la38	15335(697)	**14994**(1027)	13507(1025)	**13273**(1122)	12409(1897)	**11996**(4078)
la39	15372(859)	**15258**(924)	13879(2148)	**13689**(3987)	12664(3059)	**12326**(4099)
la40	15219(883)	**15162**(1007)	13997(1885)	**13645**(2921)	12638(2024)	**12302**(2142)
mt06	**255**(28)	**255**(59)	**219**(10)	**219**(28)	**209**(32)	**209**(10)
mt10	**7138**(379)	**7138**(687)	6066(429)	**6042**(554)	5324(29)	**5297**(67)
mt20	13645(888)	**13572**(1479)	**12644**(752)	**12644**(622)	**12282**(3425)	**12048**(3565)

ditions and that both neighborhoods N_1 and N_2 are able to find satisfactory solutions with an advantage to N_2.

We are currently investigating various improvements to speed-up the algorithm, to improve the move evaluation functions and to propose new neighborhood structures.

Acknowledgements This work is supported by École des Mines de Saint-Étienne (France), Universidad de Ibagué (Colombia), the government of Colombia in the program of scholarships Colfuturo-Ascun and the embassy of France in Colombia.

References

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research 41*(3), 157–183.

Braune, R., G. Zäpfel, and M. Affenzeller (2013). Enhancing local search algorithms for job shops with min-sum objectives by approximate move evaluation. *Journal of Scheduling 16*(5), 495–518.

Brucker, P. and R. Schlie (1990). Job-shop scheduling with multi-purpose machines. *Computing 45*(4), 369–375.

Bülbül, K. (2011). A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research 38*(6), 967–983.

Dauzère-Pérès, S. and J. Paulli (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research 70*, 281–306.

Demir, Y. and S. K. İşleyen (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling 37*(3), 977–988.

Essafi, I., Y. Mati, and S. Dauzère-Pérès (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research 35*(8), 2599–2616.

Hmida, A. B., M. Haouari, M.-J. Huguet, and P. Lopez (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research 37*(12), 2192–2201.

Hurink, J., B. Jurisch, and M. Thole (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectrum 15*(4), 205–215.

Jain, A. S. and S. Meeran (1999). Deterministic job-shop scheduling: Past, present and future. *European journal of operational research 113*(2), 390–434.

Jia, S. and Z.-H. Hu (2014). Path-relinking tabu search for the multi-objective flexible job shop scheduling problem. *Computers & Operations Research 47*, 11–26.

Kreipl, S. (2000). A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling 3*(3), 125–138.

Mastrolilli, M. and L. M. Gambardella (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling 3*(1), 3–20.

Mati, Y., S. Dauzre-Prs, and C. Lahlou (2011). A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research 212*(1), 33 – 42.

Mati, Y., N. Rezg, and X. Xie (2001). An integrated greedy heuristic for a flexible job shop scheduling problem. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, Volume 4, pp. 2534–2539. IEEE.

Mati, Y. and X. Xie (2004). The complexity of two-job shop problems with multi-purpose unrelated machines. *European Journal of Operational Research 152*(1), 159–169.

Pinedo, M. and M. Singer (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics 46*(1), 1–17.

Rajkumar, M., P. Asokan, N. Anilkumar, and T. Page (2011). A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research 49*(8), 2409–2423.

Roy, B. and B. Sussmann (1964). Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds 9*.

Singer, M. and M. Pinedo (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE transactions 30*(2), 109–118.

Wang, L., G. Zhou, Y. Xu, S. Wang, and M. Liu (2012). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International*

Journal of Advanced Manufacturing Technology 60(1-4), 303–315.

Yuan, Y. and H. Xu (2013). An integrated search heuristic for large-scale flexible job shop scheduling problems. *Computers & Operations Research 40*(12), 2864–2877.

Zhang, R. and C. Wu (2011). A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardinessobjective. *Computers and Operations Research 38*(5), 854 – 867.

MISTA 2015

Robustness of Partial Order Schedules: Understanding the Chaining algorithm

Daan Wilmer · Tomas Klos

Abstract If a schedule is executed under conditions that are uncertain at the time the schedule is constructed, then a flexible schedule is more useful than a fixed-time schedule. A flexible schedule can absorb unforeseen events that arise during execution, by incorporating a variety of fixed-time schedules. We say a schedule is robust if it admits the realized execution as one of these schedules.

Various methods have been proposed that give flexibility to schedules for the resource-constrained project scheduling problem. We focus on schedules that have the form of a partially ordered set of the project activities (a 'partial order schedule'). A partial order schedule is flexible because it incorporates all the fixed-time schedules that are consistent with the partial order.

In this paper we study the 'chaining' algorithm for constructing partial order schedules. In particular, we use empirical methods to try to understand how chaining heuristics affect the robustness of the resulting schedules, which we estimate using simulations. We propose an explanatory model, and test its implications in controlled experiments. Our experimental results confirm some but not all of our predictions, and so motivate us to elaborate our model.

1 Introduction

We study a problem model called the resource-constrained project scheduling problem (RCPSP). Scheduling a project means assigning start times to a number of precedence- and resource-constrained activities. If we want to minimize an objective function such as the last activity's start time, this is an NP-hard problem [2].

There is often considerable uncertainty about the environment in which a schedule is executed: resources may become unavailable, activities may take longer or shorter than expected, etc. A lot of research work has focused on incorporating flexibility in schedules, to make them more robust in the face of uncertainties arising during

Daan Wilmer
E-mail: daan@daanwilmer.nl

Tomas Klos
TU Delft
E-mail: t.b.klos@tudelft.nl

schedule execution. One approach is to design a *set of schedules* as the solution to a scheduling problem, as opposed to only a single assignment of fixed starting times. During execution, a suitable fixed-time schedule can be chosen from the set. This set of solutions can be represented as a Partial Order Schedule (POS), which is in fact a scheduling problem of its own, albeit one that's efficiently solvable. A POS consists of the activities and precedence constraints of the original problem instance, plus a set of extra precedence constraints such that any assignment of starting times that satisfies all precedence constraints is guaranteed to satisfy all resource constraints as well. Such an assignment of starting times can be generated during execution in little time.

One such method is Precedence Constraint Posting (PCP). This starts from a schedule that satisfies only the precedence constraints, and iteratively identifies potential conflicts that can arise when this schedule allows the concurrent execution of several activities whose combined use of at least one resource exceeds its capacity. Every such conflict is eliminated by adding ('posting') precedence constraints between some activities involved, such that the resource conflicts can not arise in the execution of any schedule that also satisfies these additional constraints. The result is a POS that can be used during execution to efficiently dispatch activities. *Chaining* is a method for selecting the precedence constraints to be posted between activities, by considering each unit of each resource, and posting precedence constraints that smartly chain activities that use each unit. Several heuristics have been proposed for selecting chains. The method was evaluated by computing the flexibility and the slack of the resulting POS, which are metrics that characterize its structure.

In fact what we are interested in, is the robustness of the schedule when it is executed. A schedule's robustness indicates how accurately it predicts its own execution: a robust schedule is executed as it was designed, while the execution of a schedule that's not robust will deviate from the schedule. Policella et al. measure a POS's flexibility because they "expect a flexible schedule to be easy to change, and the intuition is that the degree of flexibility in this schedule is indicative of its robustness."

In this paper, we first investigate the validity of this intuition. Then we undertake an empirical study, aimed at understanding how chaining decisions affect the robustness of the resulting POS. Our study is not aimed at comparing methods by setting up a 'horse race' among different heuristics [7] but rather at improving our understanding of individual algorithms.

After a formal treatment of relevant concepts in section 2, we outline our approach in section 3. This involves an exploratory study in section 4, the description of our explanatory model and the derivation of experimental predictions in section 5, and the empirical test of these predictions in section 6. We conclude in section 7.

2 Background and Related work

RCPSP The problem model we study is the resource-constrained project scheduling problem (RCPSP). An instance of the RCPCP is a tuple $I = \langle A, P, R, \mathbf{c}, \mathbf{d}, \mathbf{q} \rangle$, where A is a set of n activities, $P \subseteq A \times A$ is a binary precedence relation on A, and R is the set of resources. Each activity $a \in A$ has a duration $d_a \in \mathbb{N}_{>0}$ and uses $q_{a,r}$ units of resource r during the duration of its execution. A resource r has capacity c_r.

If $(a_i, a_j) \in P$, activity a_i has to finish before a_j can start. The set of *immediate predecessors* of a is $pred(a) = \{x \in A \mid (x, a) \in P\}$ and the set of (all) *predecessors* of a is $pred^*(a) = pred(a) \cup \bigcup_{x \in pred(a)} pred^*(x)$. Similarly, the set of *immediate successors*

of an activity a is denoted $succ(a) = \{x \in A \mid (a, x) \in P\}$, while the set of *successors* of a is $succ^*(a) = succ(a) \cup \bigcup_{x \in succ(a)} succ^*(x)$.

Schedule A (fixed time) *schedule* is an assignment $s : A \to \mathbb{R}^{\geq 0}$ of start times to activities, that satisfies all precedence and resource constraints. Precedence constraints are satisfied if $s(a_j) \geq s(a_i) + d_i$ for all $(a_i, a_j) \in P$, and resource constraints if

$$\sum_{a \in \delta(t)} q_{a,r} \leq c_r \quad \forall t \in [0, mks(s)], \forall r \in R,$$

where $\delta(t) = \{a \in A \mid s(a) \leq t < s(a) + d_a\}$ is the set of activities that are active at time t according to assignment s, and $mks(s) = \max_{a \in A}(s(a) + d_a)$ is the makespan of schedule s.

Partial Order Schedule In order to allow for adaptation to unexpected circumstances that arise during execution, it is desirable to have a *set* of schedules available, so that one can switch to an alternative when the fixed-time schedule currently in use is rendered invalid by unforeseen events such as failures or delays. Several possibilities for obtaining such a set of schedules have been explored in the literature. One option is to assign *intervals* of starting time points to activities, rather than single time points [17, 21]. The option we consider here is to construct a Partial Order Schedule (POS) that encodes a set of schedules for the scheduling problem [16, 14], by only specifying the precedence constraints that have to be satisfied by fixed-time schedules.

A POS S for an RCPSP instance $I = \langle A, P, R, \mathbf{c}, \mathbf{d}, \mathbf{q} \rangle$, is a pair $S = \langle A, P \cup U \rangle$, where U is a set of additional precedence constraints chosen such that every assignment of start times to activities s that satisfies all precedence constraints in $(P \cup U)$ is a fixed-time schedule for I. One of the fixed-time schedules that is consistent with a POS, and can be derived from it in linear time, is the earliest time schedule $\hat{s}$, in which every activity starts at its earliest start time $\hat{s}(a_i) = est(a_i)$ without violating any precedence constraints [14].

Constructing a POS Following the definition above, a Partial Order Schedule is constructed by generating a set U of precedence constraints to supplement the precedence constraints P of the instance. The set U should be chosen such that in every assignment of starting times that satisfies $P \cup U$, there can be no 'resource conflict,' a timepoint where the number of units required by the activities active at that timepoint, exceed the available capacity for at least one resource. An easy solution is to enforce the activities to execute in a topological ordering, allowing no activities to run concurrently. Constructing POSes with more desirable qualities, e.g. a shorter makespan, is non-trivial.

According to Policella et al. [13], methods for computing a POS come in two flavors: In the "outside-in" approach called precedence constraint posting (PCP), precedence constraints are iteratively added ("posted") to the set of constraints, to reduce the set of fixed-time schedules until they are all valid for the RCPSP instance; In the "inside-out" approach called Solve-and-Robustify, a fixed-time schedule for the RCPSP instance is first created by some heuristic, and subsequently extended ("robustified") to a POS. We study an algorithm that implements Solve-and-Robustify, but first discuss both methods for added understanding.

Precedence Constraint Posting This method starts with a schedule for just the temporal constraints, and iteratively finds a resource conflict and solves it by posting precedence constraints between activities involved in the conflict. One method for identifying resource conflicts [4] uses 'resource profiles,' which show the resource usage over time, given a schedule. Since the start times for activities in a POS are not fixed, Cesta et al. use an upper bound resource profile and a lower bound resource profile. The lower bound resource profile counts activities only at time points when they *must* be scheduled because of resource constraints, while the upper bound resource profile counts activities at all time points where they *could* be scheduled. The upper bound resource profile is used to detect possible resource conflicts, while the lower bound resource profile is used to prioritize conflicts. When it is decided which conflict to solve, it is reduced or resolved by adding a precedence constraint chosen to maximize slack, or the temporal distance between activities.

Another method focuses on 'critical sets': sets of activities that, when looking at the precedence constraints only, *could* be scheduled concurrently, but that would violate a resource constraints if they *were*. These sets can be detected by solving a minimum-flow problem, as proposed in [10]. To solve a resource conflict, a critical set is reduced to a 'minimal critical set': a critical set that is no longer critical when one of the activities is removed from it. Just one precedence constraint then suffices between two activities in this MCS to eliminate the conflict.

Solve-and-Robustify, Chaining With Solve-and-Robustify, one first "solves" the RCPSP instance by constructing a fixed-time schedule, for example using the ESTA algorithm [4] or a Schedule Generation Scheme. Then, this fixed-time schedule is "robustified" by transforming it into a POS. There are several methods for doing this; we focus on a greedy algorithm called 'chaining' [4, 14]. This method is based on the idea that, during execution of a schedule, each unit of each resource may be used by a series, or chain, of activities. When it finishes executing, an activity releases all the resource units is used to other activities that may subsequently use them. A chain is created for each unit of resource. Then, in the order of their starting time in the fixed-time schedule, activities are assigned to one chain for each unit of each resource they use, such that two activities that are scheduled to execute concurrently cannot be assigned to the same chain. The chains are then fixed by adding precedence constraints between each pair of consecutive activities in the chain.

A partial order schedule constructed in this way always contains the fixed-time schedule that was used as input for chaining. However, which other schedules will be admitted by the POS depends on its structure, which is determined to a large extent by the way in which chains are selected to assign activities to. Whenever a chain is selected for activity a_i, a precedence constraint (a_k, a_i) is added to U, where a_k was the last activity on the chain—before this becomes a_i. The "basic" chain selection heuristic [13] says that "the next activity a_i is always dispatched to the first available resource unit (chain) associated with its required resource r_j" [14]. A chain with last activity a_k is available for a_i if $s(a_i) \geq (s(a_k) + d_k)$. In [14], Policella et al. introduce two further heuristics, "Maximize Common Chains" (maxCC) and "Minimize InterDependencies" (minID), both aimed at reducing the number of added precedence constraints, so as to produce a POS that maximizes the number of incorporated solutions.

The maxCC heuristic selects a chain for activity a_i at random from those available, and adds the precedence constraint (a_k, a_i) to U, where a_k is the last activity in that chain. If a_i requires more units of this resource, maxCC allocates a_i to other chains that

also end in a_k, and otherwise to random other available chains. The minID heuristic also tries to maximize common chains, but starts not with a randomly selected chain for a_i, but if possible, with a chain that ends in an activity a_j for which $(a_j, a_i) \in P$, because then this constraint is already part of the POS and doesn't have to be added. These two heuristics are combined to form a policy for assigning new activities to chains, called maxCCminID. It first tries to use the maxCC heuristic, if that does not work it tries the minID heuristic, and otherwise a random activity is chosen.

Robustness A schedule's robustness indicates how accurately it predicts its own execution: a robust schedule is executed as it was designed. This entails that we can establish a POS's robustness *ex post*, by measuring the deviations between the schedule (the planned execution) and it's realized execution in an environment with disturbances and unforeseen events. When designing robust schedules, however, one will try to optimize some *ex ante* measure that is expected to yield a robust schedule. In our study, we make use of three ex ante measures and two ex post measures, which we describe now.

Ex ante robustness measures capture structural aspects of schedules that can be (i) influenced, and (ii) expected to lead to robust schedules. Chtourou and Haouari [5] present a set of measures for the robustness of a fixed-time schedule. We adapt them to measure the robustness of a POS by considering its earliest time schedule $\hat{s}$. The measures are all based on *slack*, which is defined for an activity as "the time that an activity i (...) can slip without delaying the start of any of its immediate successors, while upholding resource feasibility." For a POS, this means an activity's slack is

$$slack(a) = \min_{x \in succ(a)} (\hat{s}(x) - (\hat{s}(a) + d_a))$$

The slack of the complete schedule is calculated by taking the weighted sum of the slack values of the activities, using one of four weightings (see [5] for details).

Policella et al. use a metric called *flexibility* that applies directly to POSs [15, 13, 14]. It was introduced by Aloulou and Portmann [1] as 'sequential flexibility,' and defined as "the number of non-oriented edges in the transitive graph representing the partial order". Policella et al. normalized Aloulou and Portmann's measure:

$$flex = \frac{|\{(a_i, a_j) \mid (a_i, a_j) \notin P^+ \wedge (a_j, a_i) \notin P^+\}|}{n(n-1)}, \tag{1}$$

where P^+ is the transitive closure of P.

The *pairwise float* (PF) metric was introduced in [3] as the objective function of a (mixed) integer programming formulation of the problem of creating a robust POS, based on a fixed-time schedule. Therefore, PF is based on both the fixed-time schedule and the partial order schedule. For two activities a_i and a_j, $PF_{a_i,a_j} = s(a_j) - (s(a_i) + d_{a_i})$ if $(a_i, a_j) \in P$. If $(a_i, a_j) \in (P^+ - P)$, then $PF_{a_i,a_j} = \min_{\pi \in \Pi} PF_\pi$, where π is a path from a_i to a_j in P^+, Π is the set of all such paths, and PF_π is the sum of the pairwise floats of all pairs of activities on π. If $(a_i, a_j) \notin P^+$, then $PF_{a_i,a_j} = C$, a constant. The pairwise float of a schedule is

$$PF = \sum_{a_i, a_j \in A} PF_{a_i,a_j}.$$

Ex post robustness measures measure the difference between the schedule, and the realized state of affairs after the schedule is executed under disturbances. An

important characteristic of a schedule is its makespan. The metric $\Delta makespan$ measures the deviation of the realized end time of the last activity from this activity's end time in a POS S's earliest time schedule. We normalize this difference by the summed duration of all activities, to account for the fact that delays are often dependent on these durations. The *number of delays* metric [22] measures the number of activities of which the start time is delayed, compared to their start times in the POS S's earliest time schedule.

3 Empirical Algorithm Analysis

Various chain selection heuristics have been proposed for use in the chaining algorithm. They are designed with the aim of increasing the resulting POS's flexibility or fluidity (see [13]). The assumption is that this will also lead to a robust schedule [13]:

> *"We expect a flexible schedule to be easy to change, and the intuition is that the degree of flexibility in this schedule is indicative of its robustness."*

Previous experimental work has shown that different chain selection heuristics indeed influence the flexibility of the POS. However, these studies have the 'algorithmic race' character lamented by Hooker [7], and have merely focused on singling out the best—usually the authors'—algorithm. There has not been any investigation aim at understanding *how* chain selection heuristics influence flexibility, and more importantly, robustness—the property of a POS we're ultimately interested in optimizing.

In this paper, we adopt an empirical approach as advocated by Hooker [6] and McGeoch [11], to the study of chain selection heuristics in the chaining algorithm. We are interested in improving our *understanding* of the relation between the way chain selection decisions are made, and the robustness of the resulting POS. Ultimately, we will want to exploit this understanding, but the current paper addresses the first step of deepening our understanding. In this context of algorithmics as an empirical science, as emphasized by Hooker [7], "negative results are as important as positive results." Also, "it is symptomatic of the situation that in OR and computer science one cannot publish reports that an algorithm does *not* perform well in computational tests." In our opinion, little has changed in the twenty years since Hooker wrote his paper, unfortunately.

In this current paper we apply this empirial approach as follows. We start with an exploratory study (section 4), in which we try to give ourselves as many different views as possible on the behavior of the heuristics, and their relation with robustness. We highlight the differences in the robustness properties of POSes created using the various chain selection heuristics, and also investigate the relation between a POS's ex ante and its ex post robustness, to see if we confirm Policella et al.'s intuition above. Then, we build an explanatory model in section 5. Since this model contains unobservable characteristics of chain selection heuristics and (their interaction with) the environment in which they operate, we can't rely on direct measurements to confirm this model. Instead, we derive predictions from the model about the behavior of the algorithm under unknown conditions. Then we perform systematic, controlled experiments designed to refute these predictions, by setting up the required conditions and recording the algorithm's behavior (section 6): only if the model withstands our active attempts to overthrow it, can we call it valuable and trust it. If we are unable to refute these predictions, then this increases our confidence in the validity of our model—although by no means does this *prove* that the model is correct.

Simulation Environment We used the PSPLIB benchmark set for the RCPSP [9] in both our exploratory and our controlled experiments for testing. This PSPLIB consists of four sets of instances: J30, J60, J90, and J120, concerning projects consisting of the indicated number of activities. For the "solve" step, we created fixed-time schedules using a very simple Serial Schedule Generation Scheme ([8], see [12] for a thorough empirical study of a scheduling heuristic for the RCPSP). Using the fixed-time schedule, the next step is to create a partial order schedule. We used the chaining algorithm with the (1) basic, (2) maxCC, (3) minID and (4) maxCCminID chain selection heuristics. We also have an implementation using Mixed Integer Programming [3] that we used for additional insight in one of our exploratory experiments.

For every POS we constructed, we computed the values for the ex ante robustness metrics *slack*, *flexibility*, and *pairwise float*. To compute a POS's value for the ex post robustness metric Δ*makespan*, we compared the makespan of the POS's earliest start schedule with the makespan of a simulation of its execution using an earliest start policy to dispatch the activities, under conditions of activities with extended durations ('delays'). We introduced delays by multiplying every activity's duration with a factor ≥ 1. Every activity's factor was drawn from a distribution; a set of factors for all activities in a project is called a 'delay pattern.' We generated several different delay patterns, the combination of which we call a delay set. We experimented with four distributions—again, to provide ourselves with the opportunity to make a wide range of observations of this algorithm 'in the wild.' We generated four delay sets (sets of delay patterns):

exp2 A delay set with factors chosen from an exponential distribution with $\lambda = 2$, with a minimum factor of 1. These delay patterns reflect the case where, if an activity is delayed, a small delay is more likely than a large delay. In this set, on average 14% of activities is delayed, with a (sample) mean factor of 1.51. This results in a mean delay of 7.1% over all activities.

fixed_50_30 A delay set where one half of the activities is delayed by 30 percent, or $Pr(x = 1) = 0.5$ and $Pr(x = 1.3) = 0.5$. This distribution is inspired by the delay distributions used in [22], and is meant as an example where many activities are delayed but only by a small amount.

unif_80_5 A delay set where twenty percent of activities is delayed, by a factor chosen from a uniform distribution between 1 and 5. This distribution has a slightly larger proportion of delayed activities than the *exp2* delay set, but a smaller proportion than the *fixed_50_30* set. The delays are quite large, to simulate a scenario with a lot of delay, testing the robustness of schedules under quite extreme conditions.

gauss_02 A delay set where delays are chosen according to a gaussian distribution with $\mu = 1$ and $\sigma = 0.2$. This is the only set where factors smaller than 1 occur, and it is chosen to capture the context of scheduling projects with external contractors. In this case the act of rescheduling an activity can be very costly, regardless of the difference between the old and new start time of the activity.

Our code is released in [19] and available on GitHub.

4 Exploratory Study

In this section we report on our exploratory study, where we observe the behavior of the algorithm under a variety of circumstances. In this study, we tested all combinations

of instance set (J30, J60, J90, or J120) and delay set (as described in Section 3), unless specified otherwise. Because we're only interested in observations that are not likely to be due to chance, we determine the significance of differences between heuristics using the Wilcoxon signed-rank test [18]. This test is used for paired data that are not normally distributed. The pairing comes from the fact that we compare the results of partial order schedules for each instance individually, as opposed to viewing all POSes as a group. This reduces the influences of differences between instances, and instead focuses on differences between algorithms. In this paper we will summarize our findings; more detailed results — in the form of 14 pages of graphs and tables — are in [20].

Robustness Differences between Algorithms Since the goal of the heuristics is to increase the robustness of the resulting Partial Order Schedule, we first compare the robustness of POSes constructed using different heuristics. For all combinations of instance set and delay set, the differences between heuristics in terms of $\Delta makespan$ were very clear. POSes created using maxCC had lower $\Delta makespan$ than POSes created using basic chaining, minID gave POSes with lower $\Delta makespan$ than maxCC, and finally maxCCminID produced POSes with the lowest $\Delta makespan$ under all tested circumstances. Although variation among POSes created using the same heuristic was larger than the difference between heuristics, all these differences proved statistically significant using the Wilcoxon signed-rank test with $p < 0.001$. We suspect that the large variety in instances contributes a large amount to the difference between POSes created using the same heuristic.

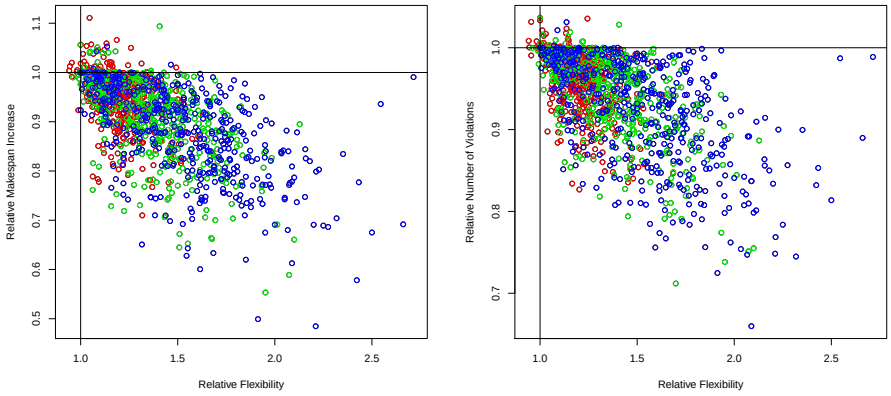
The differences in *number of delays*, follow the same pattern. Again, maxCCminID performed best, then minID, then maxCC, and basic chaining is the least robust according to this metric. For this measure, however, this pattern was not in all cases statistically significant: with three parameter combinations, the difference between maxCC and minID was not significant with a $p < 0.001$. All other differences were statistically significant, giving us confidence that the different heuristics are effective at their goal of creating more robust POSes.

4.1 Correlation between Flexibility and Robustness

We now investigate Policella et al.'s intuition, that the degree of flexibility is indicative of its robustness. We found not only a very strong correlation between flexibility and robustness but also a large variance among partial order schedules. Because of this variance, we investigated the differences between partial order schedules created for the same RCPSP instance using different heuristics. This is done using what we call a comparison plot.

The values in a comparison plot are relative values of a POS, created using one heuristic, compared to a baseline POS, created using another heuristic. To illustrate the creation of this plot: for one particular instance, maxCCminID produces a POS with flexibility 0.42 and basic chaining produces a POS with flexibility 0.21. During simulation, the normalized makespan increase for the baseline POS might be 0.2 while the POS created by maxCCminID measures 0.15. We then plot a point for maxCCminID at coordinates $(\frac{0.42}{0.21}, \frac{0.15}{0.2})$, or $(2, 0.75)$. Moreover, we also show data points for the other two heuristics (maxCC and minID), compared to basic chain selection. For ease of reading we draw two lines at $x = 1$ and $y = 1$. These denote equality of the val-

ues of the measured POS and the baseline POS, and provide a visible division between cases where the POS measured higher or lower than the baseline POS.



(a) Makespan Increase compared to Flexibility Increase

(b) Increase in *number of delays* compared to Flexibility Increase

Figure 1: Flexibility and Robustness of POSes created using maxCC (red), minID (green), and maxCCminID (blue) compared to basic chaining, for the J30 instance set and exp2 delay set.

Figure 1 shows two comparison plots for flexibility and the two robustness measures (Δ*makespan* and *number of delays*), each plot comparing the three heuristics to basic chaining. The plots show a majority of points with relative flexibility greater than one and the two metrics lower than one, suggesting that high values for flexibility are correlated with low values on these two metrics. Since these metrics actually measure *the absence of* robustness (low robustness means large makespan increase and a large number of start time delays), these plots that in a vast majority of cases flexibility and robustness both increased. This supports the idea that there is indeed, in general, a strong connection between flexibility and robustness.

To further test this idea, we used optimization for flexibility using Mixed Integer Programming, similar to the approach in [3], to construct POSes with the lowest possible number of constraints. Due to the long time needed for calculation, we limited these tests to instances of the J30 instance set. In the majority of cases, both flexibility and robustness increased, although there are also cases where flexibility increased and robustness decreased. This leads us to observe that flexibility indeed has a strong correlation with robustness, but that there are other factors that have to be taken into account.

Correlation between Slack and Robustness Another metric that is associated with robustness is *slack*. Recall that this metric measures, for an activity, the time by which it can be postponed without disturbing other activities. For a schedule, it is the summed slack of all activities, and for a partial order schedule S we define it as the slack of S's earliest start schedule. These metrics in [5] are based on three ways of measuring

slack, which we call slack, binary slack, and capped slack. The first is slack as defined in section 2, the second is a binary value which is 1 if and only if the absolute slack is larger than 0, and the third is the minimum of the slack and a fraction of the activity's processing time. In the original paper this fraction is defined as the expected delay, but here we fix it to 1 to ease computations and reasoning.

For summing these values for a schedule, Chtourou and Harouari use one of four weightings: unweighted, weighted by number of successors, weighted by resource usage, and weighted by number of successors multiplied by resource usage [5]. While unweighted and weighted by number of resources speak for themselves, the weighting by resource usage needs some explanation: this weighting counts the slack once for each item of each resource an activity requires. For each of the twelve possible metrics, we calculated the correlation between the metric and the two robustness measures. The first thing that we noticed, is that the metrics weighted by resource usage had a low or even positive correlation with the robustness metrics. Remember that the robustness metrics, Δ*makespan* and *number of delays*, actually measure the lack of robustness: an increase in some of these slack metrics weighted by resource usage therefore indicates a *decrease* in robustness. Secondly we noticed that the correlation strengths of the other six metrics—slack, binary slack, and capped slack, either unweighted or weighted by number of successors—with robustness varies quite a lot. The correlations are not as strong as the correlation between flexibility and robustness, but they are strong enough to notice: slack might be another factor affecting robustness besides flexibility.

5 Robustness Model

Having found these correlations in the exploratory study, we now want to understand why these correlations exist, and possibly (after sufficient empirical confirmation): how we can use this to our advantage? In other words: how does chaining produce robust partial order schedules, and how can we exploit this?

In this section we try to answer this question by proposing three models. The first model, the delay propagation model, describes how a single activity is affected by disturbances. The second model, which we call the robustness model, uses the delay propagation model to describe what makes a partial order schedule robust. The third and final model, called the chain selection model, uses the robustness model to describe the strategy that heuristics should use in order to create robust partial order schedules. If we understand this algorithm and the environment in which it operates well (if our models are correct, that is), then this implies that an improved heuristic we derive from our models should perform better than the current heuristics. This test is the subject of section 6—and it illustrates that it's not necessarily *bad* to compare algorithms, if this is done to test a prediction about how this comparison will turn out.

5.1 Delay Propagation Model

Our first model we call the Delay Propagation Model. This model describes how the exension of the duration of activities affects other activities' start times. In order to reason about this, we describe the earliest start schedule under delay as a function of the original earliest start schedule and the introduced delays.

$$s(a) = \begin{cases} 0 & : |pred(a)| = 0 \\ \max_{p \in pred(a)}(s(p) + d_p) & : \text{otherwise} \end{cases} \quad (2)$$

We can define the earliest start schedule using Equation 2. If we denote s' and d' to be the schedule and the duration under delay, respectively, we can substitute these in the formula to get the earliest start schedule under delay. However, we want to express s' as a function of s and the delay.

For that purpose, we introduce the notion of start delay, which is the difference between the original start time and the start time under delay: $SD_a = s'(a) - s(a)$. Rewritten, we get $s'(a) = s(a) + SD_a$. To further highlight the effects of the duration extension, we split d' into the original duration and the extension. We introduce Δ_a as the duration extension of an activity a: $\Delta_a = d'_a - d_a$. Subsituting these in Equation (2) for the predecessors' start time, we get the following.

$$s'(a) = \begin{cases} 0 & : |pred(a)| = 0 \\ \max_{p \in pred(a)}(s(p) + SD_p + d_p + \Delta_p) & : \text{otherwise} \end{cases}$$

If we introduce the margin between to subsequent activities as $m_{a,b} = s(b) - (s(a) + d_a)$, we can rewrite this equation as follows:

$$s'(a) = \begin{cases} 0 & : |pred(a)| = 0 \\ \max_{p \in pred(a)}(s(a) - m_{p,a} + SD_p + \Delta_p) & : \text{otherwise} \end{cases}$$

Now we can use Equation 2 to get the following.

$$s'(a) = s(a) + \max(0, \max_{p \in pred(a)}(SD_p + \Delta_p - m_{p,a})),$$

which means, that we can define the start delay SD_a as follows:

$$SD_a = \max(0, \max_{p \in pred(a)}(SD_p + \Delta_p - m_{p,a})) \quad (3)$$

This model is still quite opaque: it shows the start delay of an activity only based on its direct predecessors. We can, however, expand the scope of the model to include all predecessors (the set $pred^*$). We do this by focusing only on delayed activities — activities a with $\Delta_x > 0$. All other activities do not contribute to the start delay of subsequent activities but merely propagate their start delay, hence the name of the model.

As we can see in Equation 3, the influence on SD_a of its predecessor p is reduced by the margin between the activities, $m_{p,a}$. When two activities are not directly related by precedence constraints, but there is a path between them, we can use the Pairwise Float, or $PF_{p,a}$ (see section 2), as introduced in [3]. For two activities, directly connected through a precedence constraint, it is equal to the margin between these activities. For two activities which are connected through exactly one path, it is equal to the sum of the margins between the activities on the path. If there is more than one path connecting two activities, the pairwise float of these activities is equal to the minimum of the pairwise floats of the paths.

If we then define the set of delayed activities as $delayed = \{a \in A | \Delta_a > 0\}$, we can finally define the start delay SD_a as follows:

$$SD_a = \max(0, \max_{p \in (pred^*(a) \cap delayed)}(SD_p + \Delta_p - PF_{p,a})) \quad (4)$$

5.2 Robustness Model

By the ex post robustness measures, a robust POS is a schedule with as few start time delays and as little makespan increase as possible. Using the Delay Propagation model as a guidance, we now propose a model for how properties of partial order schedules influence its robustness.

*5.2.1 Size of pred**

In Equation 4 we can see that the start delay of an activity a depends on the elements of $pred^*(a) \cap delayed$. We reason that if this set contains more elements, the expected maximum increases. Reducing this set, by reducing the size of $pred^*(a)$, will therefore reduce the expected start delay. This, in turn, increases robustness by increasing stability and reducing $\Delta makespan$.

The size of $pred^*$ is closely related to $flex$, as defined in Equation (1) in Section 2. After all: flexibility is the fraction of unrelated pairs of activities, while $\sum_{a \in A} |pred^*(a)|$ equals the number of related pairs of activities. We can therefore state the following relationship: $\sum_{a \in A} |pred^*(a)| = \frac{1}{2}(1 - flex)(n - 1)$. Note that here we do not take ordering of pairs into account: the pairs (a, b) and (b, a) are equal.

This reasoning would explain the strong correlation between flexibility and robustness. Based on this, we also predict that increasing flexibility will increase robustness.

5.2.2 Pairwise Float

We can also observe in equation 4 that pairwise float reduces the start delay. Pairwise Float therefore contributes to robustness.

This can be used to explain the correlation between slack and robustness. Slack is the amount of time an activity can be postponed without disturbing any of its successors, and is therefore the minimum margin between the activity and any of its successors. Since the pairwise float is the sum of margins between activities, it is likely that an increase in slack is correlated with an increase in pairwise float.

Furthermore, pairwise float and flexibility are also connected. From Equation 4 we can see that a delay of activity $x \in pred^*(a)$ on a is completely negated if $PF_{x,a} \geq SD_x + \Delta_x$. For the start delay of a and its successors, this is effectively the same as x and a not being connected. A sufficient amount of pairwise float might therefore be used to simulate the effects of $flex$. This is what is used in [3] to incorporate flexibility into their pairwise float heuristic: by defining the pairwise float of an unrelated pair of activities as a large value.

How much pairwise float is sufficient depends on the delays and how these are propagated. We reason that, when delays are small and few, pairwise float might have an effect on robustness comparable to flexibility. When delays are larger or more numerous, more pairwise float is needed to yield the same effect on robustness compared to flexibility. We predict that an increase in pairwise float therefore increases robustness if it does not decrease flexibility, or if delays are small and few and the reduction in flexibility is comparable to the increase in pairwise float. It is uncertain what we can consider “small and few” or “comparable”, we need further investigations to determine these.

5.3 Chain Selection Model

The Robustness Model predicts that increasing flexibility, or reducing the average size of $pred^*(a)$, increases robustness. One method to approach this, is to minimize the size of $pred(a)$ for each activity individually during chaining. This method will not produce an optimal flexibility in all cases, but we suspect that it will produce flexible POSes without much computational cost.

This explains why the maxCC and minID heuristics produce more flexible POSes than basic chaining. Both heuristics only take the current activity into account when assigning schedules and reduce the size of $pred(a)$. MaxCC reduces the number of added constraints by choosing chains such that the activity under consideration shares as many chains as possible with its predecessors — thereby reducing the number of predecessors. MinID reduces the number of added constraints by choosing chains such that there already is a constraint from the last activity in the chain to the activity under consideration — thereby "re-using" the existing constraint.

However, the maxCCminID heuristic does not minimize the size of $pred(a)$. We predict that another heuristic which further reduces the size of $pred(a)$ or even minimizes it, can be used to produce more flexible partial order schedules.

6 Model Testing

In the previous section we proposed three models. In this section, we investigate the validity of these models. We will do this by deriving a new heuristic, which is predicted to increase robustness, and measuring if the predictions hold.

6.1 Proposed Heuristic

The Chain Selection Model predicts that if we optimize the size of $pred(a)$ for each activity a individually, we can increase flexibility. This should then increase robustness, if the Robustness Model is correct. We therefore propose a new heuristic, called maxChains, which groups all available chains by the last activity, and then picks a chain from the largest group.

Our proposal is to use this in combination with maxCCminID to form maxCCminIDmaxChains. We can describe it as follows:

- Select a chain x such that $last(x) = last(prev)$, where $prev$ is the previously selected chain (maxCC).
- If that is not possible, select a chain such that $last(x) \in pred^*(a)$ (minID).
- If that is not possible, determine for each activity in how many chains it is the latest activity. Of these activities, select the activities with the most chains of which it is the latest. Select a chain x such that $last(x)$ is one of these activities (maxChains).

If we use this heuristic, the algorithm will first select all "free chains", i.e. chains such that the constraints dictated by those chains are already present, using minID and maxCC. Then it will select chains such that it can get the most chains per added constraint, since it selects a chain from the largest group of chains with the same activity as latest activity, using maxChains and maxCC. This will produce, when only considering activity a, a minimum size $pred(a)$. According to the Chain Selection Model, this will

increase flexibility compared to maxCCminID. The Robustness Model predicts that this increase in flexibility should increase robustness compared to maxCCminID.

6.2 Measured Results

We performed several simulations with partial order schedules created using maxCCminID and maxCCminIDmaxChains. We first measured the difference in flexibility between partial order schedules created using the two different heuristics. In some cases flexibility decreased, but the mean flexibility increased by 2.8%, 2.5%, 2.6%, and 3.5% for the J30, J60, J90, and J120 instance sets, respectively. Although this might not seem like much, this increase in flexibility proved significant: using the Wilcoxon signed-rank test we found p-value orders of magnitude smaller than 0.001. This finding supports our confidence in the validity of the Chain Selection Model.

Having established that this heuristic does increase flexibility, we then tested whether robustness was increased as well — according to the Robustness Model, it should have. However, the data contradict our prediction: of the sixteen combinations of instance set and delay set, we found four combinations where both $\Delta makespan$ decreased and stability increased with a p-value of $p < 0.001$. In two more cases, either $\Delta makespan$ decreased or stability increased significantly, with the other metric not improving significantly or even deteriorating. We can therefore not conclude that the maxCCminIDmaxChains metrics produces more robust partial order schedules than maxCCminID. These observations therefore do not support the Robustness Model.

6.3 Discussion of Findings

The predictions made using the Chain Selection Model were matched by results: although variance is quite large, the mean flexibility has increased. We suspect that this varience in flexibility is largely due to the large diversity in the instances. However, since we completely minimized the size of $pred(a)$ for each individual activity a, this model no longer provides any hints on how to further increase flexibility. Finding correlations between instance properties and flexibility might provide new insights in how flexibility can be achieved, although that we will be part of future work.

The predictions made according to the robustness model were not matched by results: the predicted increase in robustness was lacking or insignificant in most cases. Because there are many factors influencing robustness — in contrast to flexibility, which is a product of the algorithm and the source instance — we investigated two other possible sources of change in robustness: a change in pairwise float or an inaccurate delay set. These proved not to be the source of this incongruency, however: pairwise float actually increased, which should have also increase robustness, and a delay set consisting of more delay patterns yielded the same result. We therefore conclude that the Robustness Model is incorrect or inaccurate.

Since it is based on the mathematically derived low-level description of delays given by the Delay Propagation Model, we suspect that the Robustness Model is not entirely incorrect but merely incomplete. The Delay Propagation Model and the Robustness Model both do not take the structure of the network into account, only the number of predecessors count. We suspect that this structure does have influence on robustness, but that this influence was overshadowed by the influence of the large changes in

flexibility caused by the maxCC, minID, and maxCCminID heuristics, which is why it was not noticed before.

7 Discussion

We found a strong correlation between flexibility and robustness. Although this correlation was already suspected and it not very surprising, the evidence was lacking—presenting this evidence is our first contribution. More interesting is the finding of the correlation between slack and robustness, which was not as strong as expected by other work, although circumstances were also different.

Our main contribution lies in the three models we proposed and the way we tested them. We first proposed the Delay Propagation model, which expresses the start delay of an activity as a function of the delays and pairwise float. Secondly we proposed the Robustness Model, which uses the Delay Propagation Model to describe two factors influencing robustness and their relative importance. This model proved to be lacking in predictory power, causing us to suspect that there are other factors at play, that we did not yet take into account, probably in the structure of the activity network. Finally we proposed a Chain Selection model, which explains how flexibility can be increased by optimizing a surrogate measure. Using this model we also proposed a new heuristic which produces slightly more flexible POSes than the existing chaining heuristics, although this increase did not result in more robustness.

The approach we used is one not often seen in the field of algorithmics. Often, when using empirical evaluation, the goal (if any) is merely to compare results. Our focus is not on good or bad results, but rather on understanding the chaining algorithm. This can inform us how better results can be achieved.

Based on our work, there is a lot more work that can be done. Our suggestions for future work are mainly based on two motivations: the motivation to increase the applicability of the work, and the motivation to increase the validity of the work.

For increasing applicability, we suggest performing similar experiments with instances and delay patterns from a wider variety of sources, and preferably from actual industrial applications. Furthermore, we have selected low makespan and high stability (low number of start time delays) as desired properties of schedules. We can imagine that there are applications in which these metrics are not the best tools for the job. It would therefore be very interesting to see if our findings still hold when other properties are needed for robustness.

To increase the validity of the work, we see a lot of opportunity in refining the models. The current models are not entirely accurate, and we suspect that more accuracy can be gained by studying the influence of the structure of the activity network.

References

1. M.A. Aloulou and M.C. Portmann. An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. In *Proceedings MISTA*, 2003.
2. J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Math.*, 1983.
3. K. Braeckmans, E. Demeulemeester, W. Herroelen, and R. Leus. Proactive resource allocation heuristics for robust project scheduling. *DTEW Report 0567*, 2005.

4. A. Cesta, A. Oddi, and S.F. Smith. Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings AIPS*, 1998.
5. H. Chtourou and M. Haouari. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Comp. & Indust. Eng.*, 55, 2008.
6. J.N. Hooker. Needed: An empirical science of algorithms. *Oper. Res.*, 42, 1994.
7. J.N. Hooker. Testing heuristics: We have it all wrong. *J. of Heuristics*, 1, 1995.
8. R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European J. OR*, 90, 1996.
9. R. Kolisch and A. Sprecher. PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 96, 1996.
10. M. Lombardi and M. Milano. A min-flow algorithm for Minimal Critical Set detection in Resource Constrained Project Scheduling. *Artificial Intelligence*, 2012.
11. C.C. McGeoch. Feature Article — Toward an Experimental Method for Algorithm Simulation. *INFORMS Journal on Computing*, 8, 1996.
12. F. De Nijs and T. Klos. A novel priority rule heuristic: Learning from justification. In *Proceedings ICAPS*, 2014.
13. N. Policella, A. Cesta, A. Oddi, and S.F. Smith. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications*, 20, 2007.
14. N. Policella, A. Cesta, A. Oddi, and S.F. Smith. Solve-and-robustify. *Journal of Scheduling*, 12, 2009.
15. N. Policella, A. Oddi, S.F. Smith, and A. Cesta. Generating Robust Partial Order Schedules. In *Proceedings Constraint Programming (CP)*. Springer, 2004.
16. N. Policella, S.F. Smith, and A. Oddi. Generating Robust Schedules through Temporal Flexibility. In *Proceedings ICAPS*, 2004.
17. M.E. Pollack and I. Tsamardinos. Efficiently dispatching plans encoded as simple temporal problems. In *Intelligent Techniques for Planning*. Idea Group, 2005.
18. F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6), 1945.
19. D. Wilmer. RCPSP Testing Framework, 2014. `https://github.com/dwilmer/rcpsp-testing-framework`.
20. D. Wilmer. Robust solutions for the resource-constrained project scheduling problem: Understanding and improving robustness in partial order schedules produced by the chaining algorithm. Master's thesis, Delft University of Technology, 2015. `http://discover.tudelft.nl:8888/recordview/view?recordId=TUD:oai:tudelft.nl:uuid:f403f220-e233-4f9f-8602-584d928adfc8`.
21. M. Wilson, T. Klos, C. Witteveen, and B. Huisman. Flexibility and Decoupling in Simple Temporal Networks. *Artificial Intelligence*, 214, 2014.
22. M. Wilson, C. Witteveen, T. Klos, and B. Huisman. Enhancing flexibility and robustness in multi-agent task scheduling. In *Proceedings OPTMAS*, 2013.

MISTA 2015

Heuristic Methods for Single Machine Scheduling with Periodic Maintenance

Paz Perez-Gonzalez · Manuel Dios · Victor Fernandez-Viagas · Jose M Framinan

Abstract In this paper we address the problem of scheduling jobs taking into account the existence of cyclical unavailability periods where no operation can be performed, a problem which is usually denoted in the literature as scheduling with periodic maintenance. More specifically, our research is focused onto the single machine scheduling problem with periodic maintenance and makespan minimisation as objective. This NP-hard problem has been studied previously in the literature and, although several approximate procedures have been proposed, no computational evaluation among them has been carried out. We conduct an exhaustive computational evaluation of the state-of-the-art heuristics, and propose a new heuristic for the problem that outperforms the existing ones both in terms of the quality of the solutions obtained and in the CPU time requirements.

1 Introduction

In many real-life manufacturing scenarios, scheduling of jobs must take into account the existence of –usually cyclical– periods where no operation can be performed. These unavailability periods may be seen as an special case of scheduling with deterministic machine availability constraints, in this case due to periodic maintenance, non-working shifts, holidays, etc. Although the motivation of our work is the natural interruption of operations in the factory during the weekends, the literature usually assumes that the unavailability periods are due to preventive maintenance activities that must be carried out cyclically and therefore the problem is denoted as scheduling with periodic maintenance (see e.g. [Angel-Bello et al., 2011, Yu et al., 2014]).

In our case, it is desirable that the jobs are completed within a shift, so no job should be left unfinished for the next shift. This is a usual practice in many manufacturing companies with relatively complex manual operations (such as the assembly of wiring harness in the aerospace industry, from which our case is taken), as shifts are formed by different teams of workers and having one worker to complete the tasks of a previous worker is not desirable in terms of efficiency and quality of the operation.

Paz Perez-Gonzalez, Manuel Dios, Victor Fernandez-Viagas, Jose M Framinan
Industrial Management, School of Engineering, University of Seville
E-mail: pazperez@us.es, mdios@us.es, vfernandezviagas@us.es, framinan@us.es

More specifically, our problem deals with scheduling jobs for a single operation (machine) where the jobs should start and be completed within a working shift. The goal is to minimise the maximum completion time of the jobs, or makespan. Note that, although the single machine scheduling problem with makespan objective is trivial if no availability constraints are considered, it turns to be NP-hard in the strong sense if periodic maintenance is included [Hsu et al., 2010, Low et al., 2010]. Several approximate solution procedures have been presented in the literature, including both constructive heuristics [Ji et al., 2007, Hsu et al., 2010, Yu et al., 2014] and metaheuristics [Low et al., 2010]. However, no exhaustive computational study has been carried out to compare the different solution procedures for different availability periods, so their practical suitability has not be established.

In this paper, we first carry out an exhaustive computational experimentation among the existing heuristics. In view of the excellent results obtained by one relatively fast constructive heuristic, we embed it in a fast local search schema and obtain a new heuristic which does not only outperform the rest of the constructive heuristic, but also the best up-to-now metaheuristic for the problem while consuming much less CPU time.

The rest of the paper is organised as follows. Section 2 presents the notation for the problem and the state-of-the-art. In Section 3 we present all heuristics identified in the literature, including some new constructive heuristics, and a new heuristic. In Section 4 we describe the test beds generated to carry out the computational experience. The results provided by all methods described in Section 3 are presented, and their statistical significance is tested. Finally, in Section 5 some conclusions are discussed along future research lines.

2 Background

A recent review on scheduling with availability constraints is due to [Ma et al., 2009]. In their work, the problems are classified according to the layout and the most critical assumption, i.e.: the effect of the unavailability constraints on the disrupted job. In the resumable case, if an operation cannot be finished before the unavailability period, it is preempted and it can continue when the machine is available again. In the non-resumable case, preemption is not allowed and the disrupted operation has to totally restart rather than continue. In the semi-resumable case the disrupted job will have to partially restart.

Our problem consists on scheduling n jobs, $J_1, \ldots, J_n$ in a single machine with processing times p_i, $i = 1, \ldots, n$. The machine is not continuously available due to deterministic causes, and the unavailability periods are fixed, with a common determined duration t which have to be carried out after T units of availability time. In the literature, the availability periods are denoted batches, bins or blocks. Jobs are non-resumable, so we assume that $T \geq p_i \ \forall i = 1, \ldots, n$ in order to guarantee the feasibility of the problem. The objective is to minimize the makespan or maximum completion time of the jobs. The problem can be denoted $1|nr - pm|C_{max}$ according to the standard notation [Ji et al., 2007].

To the best of our knowledge, [Ji et al., 2007] is the first paper considering the $1|nr-pm|C_{max}$ problem, although it is not included in the review by [Ma et al., 2009].

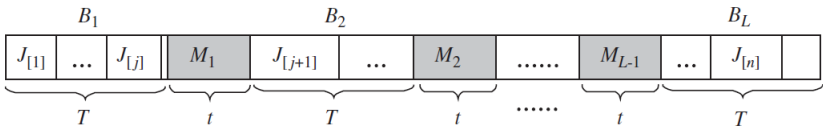


Fig. 1 Illustration of the problem [Ji et al., 2007]

As mentioned before, this problem is NP-hard in the strong sense [Hsu et al., 2010, Low et al., 2010]. In their paper, [Ji et al., 2007] describe the problem, and identify the similarity of this problem to the bin packing problem, since the interval between two consecutive maintenance periods can be considered as a batch with capacity T (see Figure 1). Its clear (see Property 1 in [Ji et al., 2007]), that the optimal schedule must have the minimum number of batches (i.e. it corresponds to an optimal solution for the bin-packing problem), denoted L. [Ji et al., 2007] show that the worst-case ratio of the LPT (Longest Processing Time first) algorithm is 2. The authors assume in their LPT heuristic that jobs are first sorted in descending order of their processing times and then are assigned one by one in the first block with sufficient slack, an assignment policy known in the bin-packing literature as First Fit. Note that other bin packing assignment policies could be possible, therefore in order to be precise we denote their algorithm as FFD, indicating a First-Fit assignment of jobs sorted in Descending order of their processing times.

[Hsu et al., 2010] consider an extended version of the problem, considering two maintenance strategies: a machine should stop to maintain after a periodic interval T, or to after a fixed amount of jobs processed K. This problem is $1|nr - pm|C_{max}$ when $K = n$. They provide a two-stage BIP (Binary Integer Programming) model: the first-stage model determines the minimum number of batches L required for processing the n jobs; the second-stage minimizes the total gap within the first $L - 1$ batches. The makespan can be calculated once the second-stage model is solved. Moreover, they present some heuristic approaches, the first one is called Decreasing order with Best Fit (denoted BFD in our paper), where jobs are arranged by LPT but the assignment to the batches is done according to the so-called Best Fit bin packing policy, i.e. jobs are assigned to the block with current minimum slack. The second one is the so-called Butterfly order with Best Fit (denoted BFHILO in our paper, in Section 3), which arranges the jobs according to the so-called butterfly order, i.e. given the jobs in LPT order, select first the largest one, then the smallest, the second largest, the second smallest, and so on.

They compare the results provided by solver LINGO for the BIP model to the heuristics in a test bed with $n = 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 500, 1000, 2000$ and 10000 for 10 instances per size, with p_i and t drawn from a uniform distribution $[1, 5, 10]$. Moreover $T = \max t\{\lceil a \sum_{i=1}^{n} p_i \rceil, \max_{1 \leq i \leq n} p_i\}$ where $a = \frac{1}{5}, \frac{1}{4}$ and $\frac{1}{3}$. Finally $K = \lceil bn \rceil$ with $b = 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}$. They show that BFHILO is better than BFD when $K \leq \lceil \frac{n}{2} \rceil$, being BFD better otherwise.

[Low et al., 2010] solve the $1|nr - pm|C_{max}$ problem using a Particle Swarm Optimization (PSO) algorithm. They use different heuristics to generate a initial population with 10 individuals, combining five arrangement rules with two policies of the bin-packing problem: best fit and first fit. The five arrangement rules are LPT, SPT, V-

Sharp, A-Sharp and Butterfly (see Section 3). They compare different versions of PSO algorithm with a test bed with $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300$, p_i uniformly distributed in $[1, 50]$, t uniformly distributed in $[5, 20]$ and T uniformly distributed in $[150, 200]$.

[Angel-Bello et al., 2011] study a generalization of the problem considering sequence dependent setup-times, denoted $1|nr-pm, s_{ij}|C_{max}$, with s_{ij} the setup-time of job j that depends on the job i processed just before j. They provide a MILP (Mixed Integer Linear Programming) model, but they show that it is not efficient and solve the problem using GRASP with an improvement phase based on Tabu Search. They use two test beds. The first test bed is generated randomly with 10, 12 and 15 jobs, and $s_{ij}+p_j$ generated by an uniform distribution in three different intervals: (2,8); (4,2) and (5,20). Finally $T = 2.25d_m, 2.5d_m, 3d_m, 4d_m$, with $d_m = \max_{1\leq i\leq n}\{(s_{0i} + p_i + s_{i0} + p_0)/2\}$, where $i = 0$ is the index for maintenance activities. For the second test bed they consider the published instances for the Asymmetrical Vehicle Routing Problem (AVRP), based on the idea about each batch of jobs is a route with a distance constraint. They use three instances for the AVRP created by [Fischetti et al., 1994] with 33, 38 and 44 clients as jobs, the distance between the clients as $s_{ij} + p_j$, and finally the distance constraint as T. They do not apply existing methods for solving the VRP since the objective of the problem corresponds to minimize the number of routes and the length of the shortest route, which is not a usual objective in routing problems.

[Pacheco et al., 2012] consider the same problem $1|nr - pm, s_{ij}|C_{max}$, providing a MILP and solving it by an algorithm called Multi-Start Tabu (MST). They generate a test bed with 10, 12, 15, 20, 30, 40 and 50 jobs, and the rest of the parameters in the same way that the test bed by [Angel-Bello et al., 2011]. They compare their results to those provided by [Angel-Bello et al., 2011], being the MST better than the GRASP for the biggest instances.

Finally, [Yu et al., 2014] consider the $1|nr - pm|C_{max}$ problem, and they provide three constructive heuristics, called LS, LPT and MLPT. The first one, LS, is the List Scheduling algorithm, which consists of generating a random order and apply the First Fit algorithm. In the following, we denote this algorithm as FFR (First Fit Random). LPT is the same FFD algorithm by [Ji et al., 2007]. Finally MLPT is a modified version of the FFD algorithm applied to the bin packing problem by [Yue and Zhang, 1995]. This heuristic is detailed in the Section 3 and denoted MFFD. [Yu et al., 2014] prove that the complexity of the three heuristics is $\mathcal{O}(n^2)$, show that the worst case bound in all cases is 2, and compare the performance bound, concluding that MFFD outperforms the other methods.

As a conclusion, to the best of our knowledge there are no comparison regarding the efficiency of the different methods that have been proposed for $1|nr - pm|C_{max}$. In the next section, we present these methods in detail so they are classified in order to carry out a computational evaluation in Section 4.

3 Heuristics

In this section we describe the heuristics found in the reviewed literature and some new heuristics to be tested in the experimental evaluation. First, we describe some constructive heuristics (CH) from different authors, including new proposals. Then, we describe a PSO (Particle Swarm Optimization) metaheuristic by [Low et al., 2010]. Finally, we

propose a new heuristic, based on BFD with a local insertion method. We have not included those methods presented by [Angel-Bello et al., 2011, Pacheco et al., 2012] since they consider setup times and the problem is very different.

3.1 Constructive Heuristics

Most of the constructive heuristics (CH) have the following structure:

Phase 1: Arrange the job according a given order, and form a sequencing priority list S.

Phase 2: Apply a bin-packing policy to assign the jobs from S to the blocks.

Let the index of the jobs be such us $p_1 \leq p_2 \leq \dots p_n$ (i.e. the processing times of the jobs to be scheduled given in non-decreasing order, SPT). Then, we consider the following different rules to sort the jobs:

- Random
- LPT: $p_n, \dots, p_1$
- SPT: $p_1, \dots, p_n$
- V-Sharp: $p_n, p_{n-2}, \dots, p_1, \dots, p_{n-3}, p_{n-1}$
- A-Sharp: $p_2, p_4, \dots, p_{n-1}, p_n, p_{n-2}, \dots, p_3, p_1$
- HILO: $p_n, p_1, p_{n-1}, p_2, p_{n-2}, p_3, \dots$ (denoted as Butterfly in the literature)
- LOHI: $p_1, p_n, p_2, p_{n-1}, p_3, p_{n-2}, \dots$

Moreover, we consider the following bin-packing policies. We call slack of a block to the difference between T and the sum of the processing times of all jobs assigned to that block:

1. First Fit: Insert the first job scheduled in S, j, into the first block where the slack is greater or equal to p_j. Remove j from S. Repeat the procedure until all jobs in S are assigned.
2. Best Fit: Insert the first job scheduled in S, j, into the block which has the minimum slack such us this slack is greater or equal to p_j. Remove j from S. Repeat the procedure until all jobs in S are assigned.

Not all the combinations of sorting rules and bin-packing policies have been tested in the literature. Table 1 shows the CH described previously, indicating whether it has been previously proposed and, if so, the reference where it can be found.

[Yu et al., 2014] develop a most sophisticated CH, denoted MFFD (Modified First Fit Decreasing). It is based on the construction of the following sets: $A = \{j : p_j > T/2\}$, $B = \{j : T/3 < p_j \leq T/2\}$, $C = \{j : \frac{T-p_n}{5} < p_j \leq T/3\}$ and $D = \{j : 0 < p_j \leq \frac{T-p_n}{5}\}$. MFFD is described as follows:

Step 1: Given the jobs in the LPT order, assign the jobs in A to the first $|A|$ bins in order, so that the levels of the bins form a non-increasing sequence (i.e., the level of a bin is the total processing times of the jobs it contains).

Step 2: From right to left (i.e., from bin $X_{|A|}$ to X_1): if the two smallest unscheduled jobs from C will not fit together in X_i , or if there is only one such job left, do nothing. Otherwise assign the smallest unscheduled job J_s from C in X_i, together with the largest remaining unscheduled job J_l from C that will fit, and then take J_s out of X_i and assign the largest remaining unscheduled job from C that will fit together with J_l into X_i.

Step 3: Use the BFD algorithm to assign the remaining jobs to bins starting from X_1.

Phase 1	Phase 2	Notation	Name	Reference
Random	FF	FFR	First Fit Random	[Yu et al., 2014]
LPT	FF	FFD	First Fit Decreasing	[Ji et al., 2007, Low et al., 2010, Yu et al., 2014]
LPT	MFF	MFFD	Modified Fisrt Fit Decreasing	[Yu et al., 2014]
SPT	FF	FFI	First Fit Increasing	[Low et al., 2010]
V-Sharp	FF	FFV	First Fit V-Sharp	[Low et al., 2010]
A-Sharp	FF	FFA	First Fit A-Sharp	[Low et al., 2010]
HILO	FF	FFHILO	First Fit HILO	[Low et al., 2010]
LOHI	FF	FFLOHI	First Fit LOHI	Not considered
Random	BF	BFR	Best Fit Random	Not considered
LPT	BF	BFD	Best Fit Decreasing	[Hsu et al., 2010, Low et al., 2010]
SPT	BF	BFI	Best Fit Increasing	[Low et al., 2010]
V-Sharp	BF	BFV	Best Fit V-Sharp	[Low et al., 2010]
A-Sharp	BF	BFA	Best Fit A-Sharp	[Low et al., 2010]
HILO	BF	BFHILO	Best Fit HILO	[Hsu et al., 2010, Low et al., 2010]
LOHI	BF	BFLOHI	Best Fit LOHI	Not considered

Table 1 Constructive Heuristics considered in the experimental evaluation

3.2 PSO

In this section we describe briefly the Particle Swarm Optimization (PSO) by [Low et al., 2010]. In this metaheuristic, each sequence is coded as a particle in the following way: $X_i^r = (x_{i11}^r, x_{i12}^r, \ldots, x_{inn}^r)$, with $x_{ijk}^r = 1$ if job j of particle i is in the position k in the iteration r, and 0 in other case. Each particle i in the iteration r has a velocity $V_i^r = (v_{i11}^r, v_{i12}^r, \ldots, v_{inn}^r)$. Let $P_i^r = (p_{i11}^r, p_{i12}^r, \ldots, p_{inn}^r)$ be the best particle i obtained in r iterations. Let $P_g^r = (p_{g11}^r, p_{g12}^r, \ldots, p_{gnn}^r)$ be the best particle of the population obtained in r iterations. The specified PSO applied by [Low et al., 2010] is the following:

Step 1: $r = 0$: Initialize a population of K particles with random velocities V_i^0.

Step 2: For each particle i, evaluate $C_{max}(X_i^r)$.

Step 3: For each particle i, if $C_{max}(X_i^r) < C_{max}(P_i^r)$ then $P_i^r = X_i^r$.

Step 4: For each particle i, if $C_{max}(X_i^r) < C_{max}(P_g^r)$ then $P_g^r = X_i^r$.

Step 5: Update the velocities as follows: $v_{ijk}^r = wv_{ijk}^{r-1} + c_1 rand_1(p_{ijk}^r - x_{ijk}^r) + c_2 rand_2(p_{gjk}^r - x_{ijk}^r)$, with $w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} r$, with $iter_{max}$ the maximal number of iterations. Positions of particles are updates by the algorithm LPV (Largest Position Value) as follows:

1. For each particle i, $\pi = \emptyset$, and $S = \{s(v_{i11}^r), s(v_{i12}^r), \ldots, s(v_{inn}^r)\}$ with $s(x) = \frac{1}{1+e^{-x}}$.

2. Select the LPV of S, $s(v_{ilm}^r)$ and place the unscheduled job l in the position m in π. Remove $s(v_{ilm}^r)$ $\forall l, m$ from S. Repeat this step until $S = \emptyset$.

Step 6: If all particles are identical and $e = \frac{C_{max}(P_g^r) - C_{low}}{C_{low}} \geq 0.01$, with C_{low} a lower bound of C_{max} for the instance, then one of particles is kept and the rest $k-1$ particles are randomly generated.

Step 7: Go to step 2 until one of the following criteria is met:

1. $C_{max}(P_g^r) = C_{low}$

2. All particles are identical and $e < 0.01$.

3. $r = iter_{max}$.

3.3 New heuristic

Finally, we propose a new heuristic based on the BFD with a local insertion, denoted HBF. The heuristic uses the excellent performance of the BF assignment policy to explore different starting orders. The steps are the following:

Step 1: Sort the jobs according to the LPT rule, obtaining sequence Π

Step 2: Assign the jobs to the block according to the Best Fit policy, thus obtaining a sequence $S_{BF}(\Pi)$ and make the corresponding makespan to be *best*, i.e. $best := C_{max}(S_{BF}(\Pi))$.

Step 3: Do:

For $i = 1$ to n:

1. Removes one job from Π at random and insert it in the rest of the positions. For each so-obtained solution Π', assign the jobs to the block according to the Best Fit policy and calculate the makespan, i.e. $curr := C_{max}(S_{BF}(\Pi'))$.

2. If $curr < best$, then $best := curr$, $\Pi := \Pi'$.

while there is improvement in *best*.

4 Experimental Results

As mentioned in Section 1, there are several test beds for our problem. However, we have created an extended test bed, with $n \in \{10, 20, 25, 50, 80, 100, 150, 200, 250, 300\}$, similar to [Low et al., 2010], where the processing times p_i uniformly distributed in $[1, 99]$ as usual in the scheduling literature. For each size, we have generated 20 instances, making a total of 200 instances.

In order to determine the influence of the availability period in the problem, we solve each instance for different values of T. Hence, $T = \max\{\lceil a \sum_{i=1}^{n} p_i \rceil, \max_{1 \leq i \leq n} p_i\}$, with $a \in [0, 1]$, in the same way that [Hsu et al., 2010]. With this method the problem is feasible ($T \geq \max_{1 \leq i \leq n} p_i\}$) and it is not trivial ($T \leq \sum_{i=1}^{n} p_i$). We consider $a = 0.16, 0.2, 0.25, 0.33$ and 0.5. Moreover, we have included the following fixed values for T: 200, 300 and 500. Note that we assume $t = 0$ and the length of the preventive maintenance activity has no influence on the solution procedure (although the value of the makespan would increase).

We have solved all instances in this testbed using the 15 CH described in Section 3, the PSO and the new heuristic HBF. The parameters used for the PSO are the same than in [Low et al., 2010]: $K = 20$ with the initial population composed by the ten sequences generated by FFD, FFI, FFHILO, FFV, FFA, BFD, BFI, BFHILO, BFV and BFA and ten sequences randomly generated. Moreover, $c_1 = c_2 = 2$, $w_{max} = 0.9$, $w_{min} = 0.4$ and $rand_1$, $rand_2$ uniformly generated in (0,1). For each instance, we have computed the makespan value, and it has been compared to $C_{low} = \sum_{j=1}^{n} p_j$, which is a lower bound of the optimal makespan value for the problem. Therefore, the RPD (relative percentage deviation) is computed as follows:

$$RPD = \frac{C_{max}(HEUR) - C_{low}}{C_{low}} \cdot 100$$

Table 2 shows the average RPD (ARPD) values obtained for each heuristic, and the average number of blocks needed by the methods (denoted by n*). Results are shown for the different cases of T. Finally, the average computation times are included in the last column. It can be observed that the four best methods are FFD, MFFD, PSO and

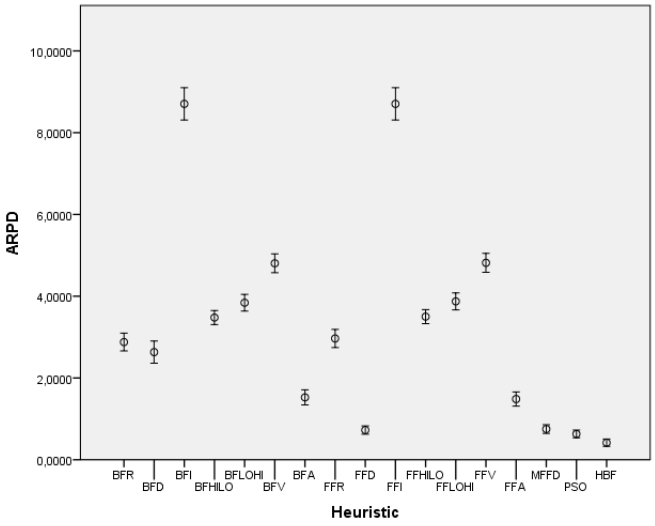


Fig. 2 95% Confidence Interval for ARPD: All heuristics

our proposal HBF, being the best results provided by HBF. Figure 2 shows these results graphically, by the 95% confidence intervals of ARPD for each heuristic. Additionally, in Table 2 we can seen that the heuristics have the worst values of ARPD for the case $T = 200$, and in this case the average number of blocks n* are the largest. It could be interpreted in the following way: as n* increases, it becomes harder for the heuristics to make the most of the capacity of the blocks. Regarding computation times, all CH are almost instantaneous. PSO and HBF need higher computation times, being HBF fastest.

In order to determine the differences among the four best heuristics, Figure 3 shows the 95% confidence intervals. MFFD by [Yu et al., 2014] is not better than FFD. Note that MFFD is the same method than FFD when $T/2 > \max\{p_j\}$. For our testbed, it provides the same results than FFD for the cases $T = 200, 300$ and 500. Additionally, according to the results presented in the Figure 3, there are not significant differences among FFD, MFFD and PSO, but there are significant differences among them and our proposal HBF, this last method providing the best ARPD values.

Finally, in order to specifically compare PSO and our proposed HBF, we have generated a new test bed along the lines described by [Low et al., 2010] so their algorithm is compared under the same conditions as in the original paper. More specifically, $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300\}$ and $p_j \sim U[1, 50]$. Fifty instances for each size (a total of 700 instances), have been generated and solved with $T \sim U[150, 200]$. Figure 4 confirms the previous results, being HBF the best method. Regarding CPU times, constructive heuristics are almost instantaneous, but results show that PSO requires an average of 2.68, while HBF requires 1.73 seconds.

Heuristics	a=0.16		a=0.2		a=0.25		a=0.33		a=0.5		T=200		T=300		T=500		Average	Average
	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	n*	ARPD	time
BFR	3.3825	7.0	3.5259	5.8	3.4694	4.7	2.8092	4.0	2.4731	2.5	3.9451	30.8	2.1841	20.5	1.2384	12.4	2.8785	0.0001
BFD	1.2466	7.0	1.2218	5.4	0.9411	4.4	0.6944	4.0	0.4072	2.3	15.9634	30.3	0.4207	20.3	0.1690	12.4	2.6330	0.0001
BFI	9.6460	7.2	8.7950	6.1	7.8438	5.1	5.4588	4.0	4.7500	3.0	15.9634	34.7	10.6959	22.3	6.4825	13.1	8.7044	0.0001
BFHILO	3.8921	7.0	3.2651	6.0	2.9470	5.0	2.3566	4.0	2.2595	2.9	6.5394	32.1	4.1869	21.1	2.3820	12.7	3.4786	0.0001
BFLOHI	4.3329	7.0	4.2417	6.0	3.1466	5.0	2.5641	4.0	2.5168	3.0	6.9871	32.1	4.3889	21.1	2.5424	12.6	3.8401	0.0002
BFV	4.9272	7.0	4.6706	6.0	4.5656	5.0	3.9297	4.0	2.6959	2.3	8.7384	32.4	5.5084	21.3	3.4011	12.7	4.8046	0.0002
BFA	2.5689	7.0	2.1418	5.6	2.3366	4.5	1.2538	4.0	0.7352	2.3	1.9924	30.5	0.8476	20.4	0.3254	12.4	1.5252	0.0002
FFR	3.5270	7.0	3.7463	5.8	3.4410	4.7	2.8092	4.0	2.4731	2.5	4.2155	30.9	2.2625	20.5	1.2549	12.4	2.9662	0.0002
FFD	1.2371	7.0	1.2183	5.4	0.9460	4.4	0.6936	4.0	0.4327	2.3	0.7044	30.3	0.4071	20.3	0.1706	12.4	0.7262	0.0002
FFI	9.6460	7.2	8.7950	6.1	7.8438	5.1	5.4588	4.0	4.7500	3.0	15.9634	34.7	10.6959	22.3	6.4825	13.1	8.7044	0.0001
FFHILO	3.9273	7.0	3.2651	6.0	2.9470	5.0	2.3566	4.0	2.2595	2.9	6.6333	32.1	4.2198	21.1	2.3955	12.7	3.5005	0.0001
FFLOHI	4.3587	7.0	4.3300	6.0	3.1466	5.0	2.5807	4.0	2.5168	3.0	7.0812	32.1	4.4284	21.1	2.5487	12.6	3.8739	0.0002
FFV	4.9780	7.0	4.6706	6.0	4.5656	5.0	3.9297	4.0	2.6959	2.3	8.7744	32.4	5.5200	21.3	3.4011	12.7	4.8169	0.0002
FFA	2.4669	7.0	2.1163	5.6	2.1177	4.5	1.2541	4.0	0.7395	2.4	1.9873	30.5	0.8694	20.4	0.3297	12.4	1.4851	0.0001
MFFD	1.2324	7.0	1.3537	5.4	1.0582	4.4	0.6496	4.0	0.4327	2.3	0.7044	30.3	0.4071	20.3	0.1706	12.4	0.7511	0.0045
PSO	1.2115	7.0	1.1129	5.3	0.8845	4.3	0.6274	4.0	0.1830	2.1	0.6235	30.3	0.2864	20.3	0.1146	12.4	0.6305	6.7118
HBF	0.9407	7.0	0.8743	5.2	0.7092	4.1	0.4367	4.0	0.0423	2.0	0.2363	30.2	0.0490	20.3	0.0447	12.4	0.4166	3.7607

Table 2 ARPD and n* for each heuristic depending on T.

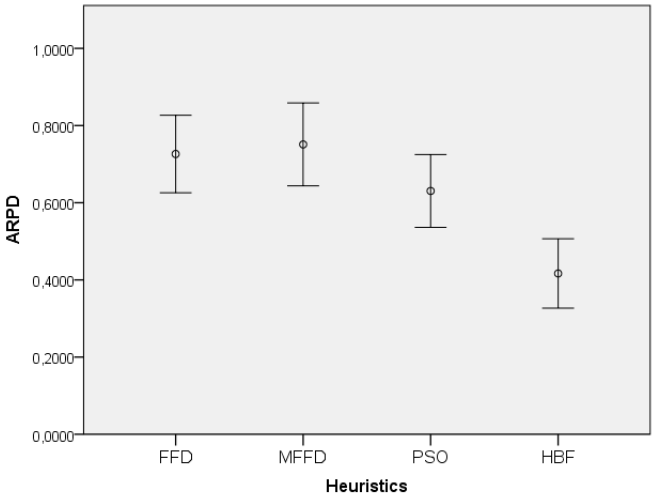


Fig. 3 95% Confidence Interval for ARPD: FFD, MFFD, PSO and HBF

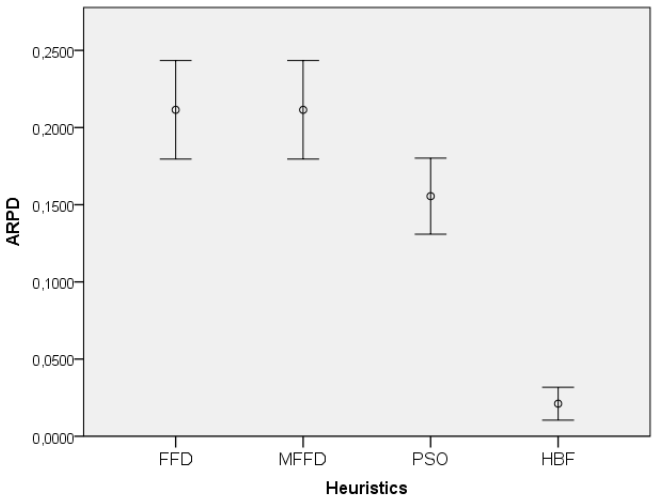


Fig. 4 95% Confidence Interval for ARPD: FFD, MFFD, PSO and HBF. Testbed [Low et al., 2010]

5 Conclusions

In this paper we address the single machine scheduling problem with periodic maintenance and makespan minimisation. This NP-hard problem has been studied previously in the literature and, although several approximate procedures have been proposed,

no computational evaluation has been carried out. We conduct an exhaustive computational evaluation of the state-of-the-art heuristics, and propose a new heuristic. The results show that the proposed heuristic outperforms existing ones in terms of quality of solutions and in CPU time requirements.

References

[Angel-Bello et al., 2011] Angel-Bello, F., Alvarez, A., Pacheco, J., and Martínez, I. (2011). A heuristic approach for a scheduling problem with periodic maintenance and sequence-dependent setup times. *Computers & Mathematics with Applications*, 61(4):797–808.

[Fischetti et al., 1994] Fischetti, M., Toth, P., and Vigo, D. (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5):846–859.

[Hsu et al., 2010] Hsu, C.-J., Low, C., and Su, C.-T. (2010). A single-machine scheduling problem with maintenance activities to minimize makespan. *Applied Mathematics and Computation*, 215(11):3929–3935.

[Ji et al., 2007] Ji, M., He, Y., and Cheng, T. (2007). Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research*, 34(6):1764–1770.

[Low et al., 2010] Low, C., Hsu, C.-J., and Su, C.-T. (2010). A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Systems with Applications*, 37(9):6429–6434.

[Ma et al., 2009] Ma, Y., Chu, C., and Zuo, C. (2009). A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211.

[Pacheco et al., 2012] Pacheco, J., Angel-Bello, F., and Alvarez, A. (2012). A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, 16(6):661–673.

[Yu et al., 2014] Yu, X., Zhang, Y., and Steiner, G. (2014). Single-machine scheduling with periodic maintenance to minimize makespan revisited. *Journal of Scheduling*, 17(3):263–270.

[Yue and Zhang, 1995] Yue, M. and Zhang, L. (1995). A simple proof of the inequality MFFD(L)71/60 OPT(L) + 1,L for the MFFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 11(3):318–330.

MISTA 2015

A novel Approximate/Exact objective based search technique for precedence constrained scheduling problems

Andrzej Kozik · Radosław Rudek

Abstract In this paper, we propose a novel Approximate/Exact objective based search technique, where a low computational cost approximation of the objective value is used to evaluate examined solutions during searching process. We apply the proposed approach for simulated annealing solving the single machine scheduling problem to minimize the total completion times with ready times, sequence dependent setups and precedence constraints. The numerical analysis reveals the advantages of the novel technique, which overwhelms the typical approach.

1 Introduction

Consider an algorithm that performs some move on the solution, where a move is a small change in the solution, and consider the evaluation of the objective function is computationally expensive (see [3]). On the other hand, there exists an approximation of the objective function that takes less computational time to be evaluated. Applying it to a local search algorithm (see [2]) results in better running times, but at the cost of worse solution delivered by the algorithm.

In this paper, we develop a novel Approximate/Exact (A/E) technique applied for local search algorithms, in which the algorithm uses approximation of the objective value to score examined solutions. By controlling the ratio between number of exact and approximate iterations, a family of algorithms with different trade-offs between computational-time and quality of delivered solutions can be obtained.

The remainder of this paper is organized as follows. The considered scheduling problem is formulated in the next section. The novel Approximate/Exact objective based search technique together with the tuned simulated annealing algorithm and the applied neighborhood are described subsequently. The numerical analysis is provided subsequently and the last section concludes the paper.

Andrzej Kozik
Institute of Mathematics and Informatics, Opole University, Poland
E-mail: andrzej.kozik@outlook.com

Radosław Rudek
Institute of Business Informatics, Wrocław University of Economics, Poland
E-mail: rudek.radoslaw@gmail.com

2 Problem formulation

At first, we will define a scheduling problem that is used to depict the idea of the proposed Approximate/Exact objective search technique. Related scheduling problems were considered in [3], [7], [8] and for a recent survey see [1].

There is given a single machine and a set $J = \{1, \ldots, n\}$ of n jobs. There are precedence constraints between jobs, which means that if job j must be processed before job k (denoted by $j \prec k$), then only such schedules are feasible. Let $G_p(V, E_p)$ be an acyclic directed graph representing precedence constraints between jobs, where V is a set of vertices representing jobs and E_p is a set of directed edges such that $(i, j) \in E_p$ if $i \prec j$. Jobs are available at their ready times r_j and the machine is continuously available and can process at most one job at a time. Once it begins processing a job, it will continue until this job is finished. Due to practical reasons, it is assumed that each job j may required additional setup time s_{ij} to prepare the machine after the previous job i is completed, i.e., it is sequence dependent.

Let $\pi = \langle \pi(1), ..., \pi(i), ..., \pi(n) \rangle$ denote the sequence of jobs (a permutation of the elements of the set J), where $\pi(i)$ is the job processed in position i in this sequence. By Π, we denote the set of all possible permutations, whereas $\Pi^+ \subset \Pi$ is the set of all such feasible permutations, which hold the defined precedence constraints. For the given sequence (permutation) $\pi \in \Pi^+$, we can easily determine the completion time $C_{\pi(i)}$ of a job placed in the ith position in π from the following formulae:

$$C_{\pi(i)} = \max\{r_{\pi(i)}, C_{\pi(i-1)} + s_{\pi(i-1),\pi(i)}\} + p_{\pi(i)}, \tag{1}$$

where $C_{\pi(0)} = 0$.

The objective is to find such a schedule (sequence) π of jobs on the single machine, which minimizes the total completion time criterion $TC(\pi) \triangleq \sum_{i=1}^{n} C_{\pi(i)}$. Formally the optimal schedule $\pi^* \in \Pi^+$ for the total completion time minimization problem is defined as $\pi^* \triangleq \text{argmin}_{\pi \in \Pi^+} \left\{ TC(\pi) \right\}$, where Π^+ is a set of all feasible schedules. For convenience and to keep an elegant description of the considered problem we will use the standard three field notation scheme $X \mid Y \mid Z$ (see [5]), where X describes the machine environment, Y describes job characteristics and constraints and Z represents the minimization objectives. According to this notation the problem analysed in the paper will be denoted as $1|r_j, s_{ij}, prec| \sum C_j$.

3 Approximate/Exact objective based search

In this section, we present the idea of the Approximate/Exact (A/E) objective based search technique applied for the well known simulated annealing algorithm ([6]). For more details concerning metaheuristic algorithms see [4], [**?**].

Let describe the general concept of the considered simulated annealing (SA). The search process of SA is a semi-random trajectory in the search space, biased towards the promising region of a solution space. In its basic form SA combines an intensification strategy (an iterative local search using a neighborhood concept) with a form of diversification (by so-called cooling-schedule allowing uphill moves to escape from the local minimum).

The considered SA starts the search process from a feasible initial solution. Next, at each iteration SA performs a random deletion of a job (say, k) from the actual solution

π followed by the evaluation of considered neighborhood, i.e. all possible insertion positions of k into solution π are evaluated, and the new solution π' is randomly sampled from this neighborhood. Let $TC(\pi)$ be the objective function to be minimized. During searching process SA with A/E uses alternately the exact (E) objective value or its approximate (A) value (which is defined in the further part of this paper). The new solution π' replaces the old solution π either with a probability computed following the Boltzmann distribution as follows $\min\left\{1, exp(-(TC(\pi') - TC(\pi))/T)\right\}$, where T is the temperature parameter, which is decreased after each iteration of an algorithm by a geometric cooling scheme, i.e., $T = \lambda T$ and $\lambda \in (0, 1)$.

Now, let us describe the neighborhood search and the approximation of the objective value applied for the considered scheduling problem. Let $\pi \in \Pi^+$ be a feasible permutation and π^D be π with removed job k. Let π' be permutation π^D with arbitrarily inserted job k. The neighborhood search procedure, given as Algorithm 1, computes all feasible insertions of job k into π^D. It returns a set $\mathcal{N}$ of pairs (i, TC), where i is an insertion position of job k and TC is its corresponding criterion value.

The insertion position i of job k partitions jobs in π^D into two subsets: on the left side of k in permutation π^D (in positions $1, \ldots, i$) and on the right size (in positions $i+1, \ldots, n$). Example insertion of k is presented in Figure 1.

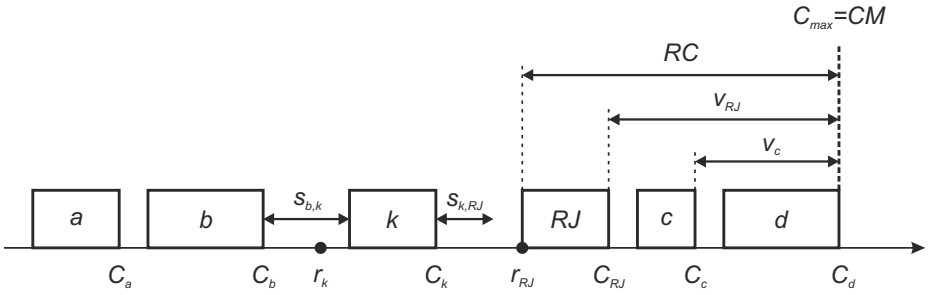


Fig. 1 Example insertion of job k into permutation $\langle a, b, RJ, c, d\rangle$.

Let RJ be a job in position $i+1$ in π^D on the right side (assume, $\pi(n+1) = 0$ and $s_{0,i} = 0$ and $s_{i,0} = 0$ for each $i \in J$). Let LS be a sum of completion times of jobs on the left ($LS = C_a + C_b$ in Figure 1), and let RS be a sum of times between C_{max} and completion of jobs on the right ($RS = v_c + v_{RJ}$ in Figure 1). Let RN be the number of jobs on the right of k and let RC be the time needed to complete jobs on the right. Let CM be a minimum value of the makespan C_{max} (the completion time of the last job) such that each job on the right of k starts its execution not earlier than its ready time – it is calculated according to (1).

The initial solution is feasible, thus, each considered permutation π is also feasible permutation. Therefore, in π^D all precedence constraints between jobs from π^D are met, and feasibility of π' depends only on satisfiability of constraints on job k. To efficiently verify the feasibility of a solution, the parameter Λ is applied. It denotes the number of unsatisfied constraints on job k. Since the insertion of job k starts from the last position, then initially, there are no jobs on the right of k. Thereby Λ is equal to the number of jobs that (according to the precedence constraints described by graph G_p) have to be processed after k, i.e., $\Lambda = |\{(k, i) \in G_p : i \in J\}|$.

Let us analyse formally the neighborhood search procedure (Algorithm 1), which for the input permutation π^D with deleted job k calculate the set $\mathcal{N}$ of feasible insertions of k with corresponding criterion values. The algorithm initializes the required values (lines 1-5) and it starts the analyse insertions of k starting from the last position (n) to the first in π^D (line 6). Note that the insertion into the first position is considered in lines 23-27. For each insertion position i, the algorithm first computes the approximated total completion time TC (lines 7-11). Next, the feasibility of insertion of k to position i is evaluated (line 12), i.e., if there are no unsatisfied constraints, then the insertion i and corresponding criterion value TC are added to the resulting set of feasible insertions $\mathcal{N}$. If job j in position before k must be processed after it, then the number of unsatisfied constraints is decreased (line 13), since in the next step, job j is after k. On the other hand, if job j must be processed before k, then the searching process is terminated, since next insertion are infeasible (line 14). In lines 15-21 the required parameters are updated.

Since each iteration of the loop (lines 7-21) is performed in the constant time, and the initialization part (lines 1-5) takes $O(n)$ time, the procedure takes $O(n)$ in total. Therefore, each solution in the neighborhood is evaluated in an amortized constant time.

Algorithm 1 Neighborhood search procedure (π^D,k)

1 : $\mathcal{N} = \emptyset$ *// the set of feasible insertions is empty*
2 : $\Lambda = |\{(k,i) \in G_p : i \in J\}|$ *// the number of jobs that must be proceeded after k*
3 : `For each` $j \in \pi^D$ `compute its completion time` C_j
4 : $LS = \sum_{j \in \pi^D} C_j$
5 : $RS = RC = RJ = RN = CM = 0$
6 : `For` $i = n$ `to 2 do`
7 : $j = \pi^D(i-1)$ *//a job in position* $i-1$ *in* π^D
8 : $C_k = \max\{C_j + s_{j,k}, r_k\} + p_k$
9 : $C_{max} = C_k + s_{k,RJ} + RC$
10: `If` $C_{max} < CM$ `then` $C_{max} = CM$
11: $TC = LS + C_k + RN \cdot C_{max} - RS$
12: `If` $\Lambda = 0$ `then` $\mathcal{N} = \mathcal{N} \uplus (i+1, TC)$ *//add insertion to feasible solutions*
13: `If` $(k,j) \in G_p$ `then` $\Lambda = \Lambda - 1$ *// k must be processed before j*
14: `If` $(j,k) \in G_p$ `then return` $\mathcal{N}$ *//next insertions are infeasible - terminate*
15: $LS = LS - C_j$
16: $S = RC + s(j, RJ)$
17: $RS = RS + S$
18: $RC = S + p_j$
19: `If` $RC + r_j > CM$ `then` $CM = RC + r_j$
20: $RN = RN + 1$
21: $RJ = j$
22: `End For`
23: $C_k = r_k + p_k$
24: $C_{max} = C_k + s_{k,RJ} + RC$
25: `If` $C_{max} < CM$ `then` $C_{max} = CM$
26: $TC = LS + C_k + RN \cdot C_{max} - RS$
27: `If` $\Lambda = 0$ `then` $\mathcal{N} = \mathcal{N} \uplus (1, TC)$
28: `Return` $\mathcal{N}$

4 Numerical analysis

In this section, we will analyse the simulated annealing algorithms based on A/E search to solve the considered problem. They were coded in C++ and simulations were run on PC, CPU Intel® Core^{TM}i7-2600K 3.40GHz and 8GB RAM.

The following algorithms are compared:

- SA_1 – simulated annealing that chooses a *first feasible solution* from a neighborhood, which is the reference algorithm for the proposed approach;
- SA_N – simulated annealing that chooses a random feasible solution from *all feasible solutions* in a neighborhood; it is slower than SA_1, since it calculates all feasible insertions, but on the other hand it provides better results;
- SA_{APR} – SA_N that uses an approximation of an objective value in each iteration;
- A/E_m – it alternately uses SA_{APR} during $100 \times (1-m)$ iterations and next SA_N for $100 \times m$ iterations.

Note that $SA_{APR} \equiv A/E_0$ and $SA_N \equiv A/E_1$.

Values of the parameters of simulated annealing were chosen empirically as follows: $Iterations = 25000$, $T_0 = 1000$ and $\lambda = 0.998$. The initial permutation is a random feasible solution.

The proposed approach is evaluated for the following problem sizes $n \in \{100, 500\}$ and in the given ranges of parameters: $p_j \in [1, 10]$ and $[1, 100]$ (i.e., $\{1, 2, \ldots, 10\}$ and $\{1, 2, \ldots, 100\}$), $s_{ij} \in [1, 10]$ and $[1, 100]$, $r_j \in [0, \tau \sum_{j=1}^{n} p_j]$, where $\tau \in \{0.5, 1.0, 2.0\}$ for $j = 1, \ldots, n$. For each combination of the parameters, 100 different random instances (replications) were generated from the uniform distribution over the integers (i.e., 2400 different instances).

The algorithms are compared in reference to an initial random solution, for each instance, according to the relative error δ_A that is calculated in the following way:

$$\delta_A = \Big(\frac{TC(\pi^A) - TC_0}{TC_0} \Big) \cdot 100\%, \tag{2}$$

where TC_0 is an initial random solution that is reference to evaluate efficiency of the algorithms and $TC(\pi^A)$ denotes the criterion value provided by algorithm $A \in \{SA_1, SA_N, A/E_0, A/E_{0.25}, A/E_{0.50}, A/E_{0.75}\}$ for each instance.

The results concerning the percentage values of mean relative errors provided by the analysed algorithms as well as their mean running times are presented in Table 1.

From Table 1 follows that SA_1 is about 10% faster than SA_N, but it delivers significantly worse solutions (20% to 30%). This clearly shows the advantage of the proposed neighborhood within simulated annealing algorithm.

Due to the low computational complexity of the neighborhood evaluation, $SA_{APR} \equiv A/E_0$ is 7 times faster than SA_N for $n = 100$ and 25 times faster for $n = 500$. However, it delivered solutions up to 15% worse than SA_N. It shows that criterion approximation errors can influence the search process more than the fast neighborhood evaluation. On the other hand, adding correction iterations to SA_{APR}, i.e., $A/E_{0.25}$, $A/E_{0.50}$ and $A/E_{0.75}$ results in run-time 30%, 60% and 90% of SA_N, respectively. Thus, solutions are significantly better than that of SA_{APR}, most frequently close to SA_N.

The impact of the problem size n on the running times of the algorithms is shown in Table 1 (for the fixed numbers of iterations). However, to conduct a fair comparison

Table 1 Percentage values of mean relative errors of algorithms and their mean running times (in square brackets) in seconds

n	p_j	s_{ij}	τ	SA_1	SA_N	SA_{APR}	$A/E_{0.25}$	$A/E_{0.50}$	$A/E_{0.75}$
100	10	10	0.5	-22.11	-52.43	-52.13	-52.46	-52.43	-52.48
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.25]
			1	-19.13	-55.27	-53.59	-54.67	-55.12	-55.19
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.24]
			2	-15.09	-50.02	-45.02	-49.63	-49.89	-49.98
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.25]
		100	0.5	-42.56	-73.12	-73.22	-73.29	-73.48	-73.24
				[0.28]	[0.31]	[0.04]	[0.11]	[0.19]	[0.26]
			1	-39.69	-71.91	-71.9	-71.88	-71.87	-71.90
				[0.28]	[0.31]	[0.04]	[0.11]	[0.18]	[0.25]
			2	-34.88	-68.83	-67.47	-68.25	-68.41	-68.69
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.25]
	100	10	0.5	-19.74	-48.09	-47.45	-47.92	-48	-48.06
				[0.28]	[0.30]	[0.04]	[0.11]	[0.17]	[0.24]
			1	-16.94	-50.07	-46.26	-49.76	-49.98	-50.04
				[0.28]	[0.30]	[0.04]	[0.11]	[0.17]	[0.24]
			2	-15.68	-44.79	-35.54	-44.51	-44.73	-44.76
				[0.28]	[0.31]	[0.04]	[0.11]	[0.18]	[0.25]
		100	0.5	-25.32	-54.63	-54.2	-54.31	-54.41	-54.68
				[0.28]	[0.31]	[0.04]	[0.11]	[0.18]	[0.25]
			1	-22.91	-57.05	-55.15	-56.37	-56.7	-56.91
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.24]
			2	-19.66	-52.2	-46.39	-51.78	-52.08	-52.14
				[0.28]	[0.30]	[0.04]	[0.11]	[0.18]	[0.25]
500	10	10	0.5	-18.88	-50.37	-48.8	-49.16	-49.65	-50.13
				[7.29]	[8.13]	[0.29]	[2.27]	[4.24]	[6.22]
			1	-15.45	-54.36	-47.51	-49.71	-51.93	-53.41
				[7.53]	[8.19]	[0.28]	[2.30]	[4.29]	[6.27]
			2	-12.51	-52.64	-38.23	-45.61	-50.03	-51.74
				[8.04]	[8.62]	[0.28]	[2.42]	[4.54]	[6.62]
		100	0.5	-40.68	-69.86	-69.8	-69.71	-69.61	-69.86
				[7.95]	[8.43]	[0.29]	[2.35]	[4.40]	[6.45]
			1	-38.58	-67.94	-67.32	-67.49	-67.54	-67.76
				[7.95]	[8.48]	[0.29]	[2.38]	[4.45]	[6.53]
			2	-33.76	-64.59	-61.77	-62.66	-63.31	-64.02
				[7.78]	[8.22]	[0.28]	[2.29]	[4.29]	[6.30]
	100	10	0.5	-14.53	-49.4	-46	-47.02	-48.02	-48.65
				[7.64]	[8.20]	[0.28]	[2.29]	[4.28]	[6.28]
			1	-11.35	-56.77	-42.58	-50.69	-54.48	-55.96
				[7.49]	[8.17]	[0.30]	[2.28]	[4.23]	[6.15]
			2	-10.14	-49.97	-36.11	-43.82	-47.68	-49.23
				[7.66]	[8.24]	[0.28]	[2.27]	[4.24]	[6.27]
		100	0.5	-22.47	-52.73	-51.31	-51.66	-51.94	-52.37
				[8.12]	[8.52]	[0.29]	[2.39]	[4.48]	[6.58]
			1	-19.3	-59.04	-50.88	-54.4	-56.63	-58.29
				[7.79]	[8.20]	[0.29]	[2.30]	[4.30]	[6.24]
			2	-16.45	-55.15	-43.96	-48.66	-51.8	-54.13
				[8.33]	[8.76]	[0.30]	[2.37]	[4.47]	[6.62]

between the algorithms, it is also worth analysing their accuracy in finding efficient solutions when their running times are the same. Therefore, an additional experiment is provided, where the stopping condition of the algorithms is the running time that is set to 1s. The results of this experiment are shown in Table 2.

Table 2 Percentage values of mean relative errors of algorithms; the running times of the algorithms are set to 1s

n	p_j	s_{ij}	τ	SA_1	SA_N	SA_{APR}	$A/E_{0.25}$	$A/E_{0.50}$	$A/E_{0.75}$
100	10	10	0.5	-21.94	-52.82	-53.76	-53.15	-53.01	-52.91
			1	-18.61	-55.74	-55.45	-55.92	-55.81	-55.72
			2	-14.03	-49.81	-46.37	-49.87	-49.85	-49.84
		100	0.5	-42.34	-74.16	-75.85	-75.06	-74.46	-74.19
			1	-39.83	-73.13	-74.61	-73.95	-73.43	-73.42
			2	-35.65	-70.28	-70.76	-70.56	-70.41	-70.29
	100	10	0.5	-18.28	-47.62	-47.74	-47.77	-47.65	-47.70
			1	-16.76	-50.25	-48.2	-50.27	-50.26	-50.26
			2	-15.45	-44.72	-36.47	-44.72	-44.72	-44.72
		100	0.5	-25.92	-55.83	-56.59	-56.19	-55.9	-55.93
			1	-23.29	-57.55	-57.2	-57.64	-57.59	-57.56
			2	-18.74	-51.4	-47.24	-51.46	-51.45	-51.45
500	10	10	0.5	-16.98	-27.73	-54.49	-42.41	-36.21	-31.41
			1	-13.32	-24.17	-53.48	-39.62	-32.57	-28.37
			2	-10.71	-21.76	-44.44	-34.03	-28.4	-24.86
		100	0.5	-38.65	-51.24	-74.37	-63.64	-58.64	-54.65
			1	-36.93	-49.4	-72.17	-61.54	-56.07	-52.52
			2	-32.69	-44.21	-66.54	-56.28	-51.25	-47.64
	100	10	0.5	-13.44	-24.14	-50.13	-38.37	-31.53	-27.12
			1	-11.28	-27.99	-48.28	-40.28	-34.16	-30.55
			2	-9.55	-20.98	-40.38	-31.41	-26.89	-23.50
		100	0.5	-21.48	-31.37	-56.61	-44.59	-38.51	-34.56
			1	-17.84	-30.74	-57.77	-43.88	-38.17	-34.00
			2	-15.39	-26.92	-51.38	-39.02	-33.6	-29.57

For $n = 500$, SA_{APR} is 20-30% better than SA_N. Similarly, A/E_m are also better than SA_N. This clearly shows the advantage of the power of examining the greater fraction of the solution space, neglecting approximation errors, whereas SA_N has no sufficient time to reach good solutions. This is visible for $n = 100$, where one second is still to short for SA_N to find good solutions, but enough for A/E_m to find solutions better than SA_{APR}. This reveals that using A/E_m, a trade-off between run-time of algorithm and quality of delivered solutions can be determine. It is especially important in the time-constrained applications.

5 Conclusions

In this paper, a novel Approximate/Exact objective based search technique is proposed. Its main idea is the low computational cost approximation of the objective value to evaluate examined solutions during searching process. We applied the proposed approach for simulated annealing solving the single machine scheduling problem to minimize the total completion times with ready times, sequence dependent setups and precedence constraints. The numerical analysis proved the A/E technique to be a new promising direction of research.

Therefore, our future research will focus on the further evolvement of the method, its application to other metaheuristics (e.g., tabu search) as well as the development of a general framework that covers different combinatorial optimization problems, especially in the field of scheduling with precedence constraints.

Acknowledgements This work was supported by The National Science Centre, Poland, under project no 2012/05/D/HS4/01129.

References

1. Allahverdi A., The third comprehensive survey on scheduling problems with setup times/costs, European Journal of Operational Research, 246, 345–378 (2015)
2. Blum C., Roli A., Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys, 35, 268-308 (2003)
3. Driessel R., Mönch L., Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times, Computers & Industrial Engineering, 61, 336–345 (2011)
4. Eglese R.W., Simulated annealing: a tool for operational research, European Journal of Operational Research, 46, 271–281 (1990)
5. Graham, R. L. and Lawler, E. L. and Lenstra, J. K. and Rinnooy Kan, A. H. G., Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 5, 287–326 (1979)
6. Kirkpatrick, S. and Gelatt, C. D. and Vecchi, M. P., Optimization by simulated annealing, Science, 220, 671–680 (1983)
7. Tanaka S., Sato S., An exact algorithm for the precedence-constrained single-machine scheduling problem, European Journal of Operational Research, 229, 345–352 (2013)
8. Wang J.-B., Wang J.-J., Single-machine scheduling problems with precedence constraints and simple linear deterioration, Applied Mathematical Modelling, 39, 1172–1182 (2015)

MISTA 2015

Tropical optimization problems in project scheduling

Nikolai Krivulin

Abstract We consider a project that consists of activities operating in parallel under various temporal constraints, including start-to-start, start-to-finish and finish-to-start precedence relations, early-start, late-start and late-finish time boundaries, and due dates. Scheduling problems are formulated to find optimal schedules for the project with respect to different objective functions to be minimized, including the project makespan, the maximum deviation from the due dates, the maximum flow-time, and the maximum deviation of finish times. We represent the problems as optimization problems in terms of tropical mathematics, and then solve these problems by applying direct solution methods of tropical optimization. As a result, new direct solutions of the problems are obtained in a compact vector form, which is ready for further analysis and practical implementation.

1 Introduction

Tropical optimization problems, which are formulated and solved in the framework of tropical mathematics, find increasing use in various fields of operations research, including project scheduling. As an applied mathematical discipline concentrated on the theory and applications of idempotent semirings, tropical (idempotent) mathematics dates back to a few seminal papers by Pandit [30], Cuninghame-Green [8], Giffler [11], Hoffman [15], Vorob'ev [35], and Romanovskiĭ [31], including the papers [8] and [11] concerned with optimization problems drawn from machine scheduling.

In succeeding years, tropical optimization problems were investigated in a number of publications, in which scheduling issues frequently served to motivate and illustrate the study. Specifically, various scheduling problems are examined in terms of tropical optimization by Cuninghame-Green [7], U. Zimmermann [38], K. Zimmermann [36, 37], Bouquard et al. [4], Fiedler et al. [10], Butkovič and Tam [6], Butkovič [5], Houssin [16], and Aminu and Butkovič [2]. Many optimization problems are formulated in the tropical mathematics setting to minimize a linear or nonlinear function defined on vectors over an idempotent

Nikolai Krivulin
Faculty of Mathematics and Mechanics
Saint Petersburg State University
St. Petersburg, 198504, Russia
E-mail: nkk@math.spbu.ru

semifield (a semiring with multiplicative inverses), and may have constraints in the form of vector equalities and inequalities (see, e.g., an overview in [22]). For some problems, complete direct solutions are obtained in a closed form under fairly general assumptions. Other problems are solved algorithmically by means of iterative computational procedures.

In this paper, we examine several problems drawn from project scheduling, to provide new solutions on the basis of recent results in tropical optimization. For details on the models and methods of project scheduling, one can consult the monographs by Demeulemeester and Herroelen [9], Neumann et al. [29], T'kindt and Billaut [33], and Vanhoucke [34].

We consider scheduling problems, which are to find an optimal schedule for a project that involves a set of activities operating in parallel under various temporal constraints, including start-to-start, start-to-finish, finish-to-start, early-start, late-start, late-finish, and due-date constraints. As optimization criteria to minimize, we take the project makespan, the maximum deviation from due dates, the maximum flow-time, and the maximum deviation of finish times. The problems under study are known to have algorithmic solutions in the form of iterative computational procedures, and can also be solved as linear programming problems by an appropriate linear programming algorithm [9, 29, 33, 34].

We represent the scheduling problems as tropical optimization problems, which are then solved by applying direct solution methods developed by Krivulin in [20, 21, 23, 24, 27]. As a result, we offer new direct solutions to the scheduling problems considered, which, in contrast to the conventional algorithmic solutions, provide results in a compact explicit vector form, ready for further development and practical implementation. Specifically, the new solutions allow various constraints to be incorporated in a unified and constructive way. The calculation of the solutions involves simple matrix-vector computations according to explicit formulae, which forms a basis for efficient computational algorithms and software.

The rest of the paper is organized as follows. In Section 2, we present scheduling problems that motivate and illustrate the study, and formulate these problems by using the ordinary notation. Section 3 includes a brief overview of preliminary definitions and results of tropical mathematics to be used in the subsequent sections. In Section 4, we describe some tropical optimization problems together with their solutions. In Section 5, we first rewrite the scheduling problems as tropical optimization problems, and then solve them by applying the results from Section 4.

2 Project scheduling model and example problems

We start with the description of a project scheduling model and the formulation of example problems of optimal scheduling in a general form (see, e.g., [9, 29, 33, 34] for further details on the common terminology and notation used in the area).

Consider a project that consists of n activities operating in parallel under start-to-start, start-to-finish and finish-to-start precedence relations, due dates, and time boundaries in the form of early-start, late-start and late-finish constraints. To describe the temporal constraints and scheduling objectives, we use, for each activity $i = 1, \ldots, n$, the notation x_i and y_i to represent the unknown start and finish times of the activity, respectively.

2.1 Temporal constraints

We now examine constraints on the start and finish times of each activity $i = 1, \ldots, n$. First, we represent precedence relations, which link activity i with other activities. Let a_{ij} be the

minimum time lag between the start of activity j and the finish of i. The time lag a_{ii} specifies the minimum duration of the activity (the duration provided that no other constraints are imposed). Note that a negative a_{ij} can be interpreted as the maximum time lag between the finish of j and the start of i. If there is no lag defined, we assume $a_{ij} = -\infty$.

The start-to-finish constraints take the form of the inequalities $a_{ij} + x_j \leq y_i$ holding for all $j = 1, \ldots, n$. The activity is assumed to finish immediately after all start-to-finish constraints are satisfied, and thus at least one of the inequalities holds as an equality. Then, these inequalities are equivalent to one equality

$$\max(a_{i1} + x_1, \ldots, a_{in} + x_n) = y_i.$$

Furthermore, we denote by b_{ij} the minimum time lag between the start of activity j and the start of i, and put $b_{ij} = -\infty$ if the lag is not indicated. The start-to-start constraints are given by the inequalities $b_{ij} + x_j \leq x_i$ for all j, which can be rewritten as one inequality

$$\max(b_{i1} + x_1, \ldots, b_{in} + x_n) \leq x_i.$$

Let the minimum time lag between the finish of activity j and the start of i be denoted by c_{ij}, with $c_{ij} = -\infty$ if undefined. The finish-to-start constraints are written as the inequalities $c_{ij} + y_j \leq x_i$ for all j, or as one inequality

$$\max(c_{i1} + y_1, \ldots, c_{in} + y_n) \leq x_i.$$

Finally, we introduce due dates and time boundary constraints. The due date indicates the time when the activity is expected to finish, and therefore, it is not actually a strict constraint. For activity i, we denote the due date by d_i.

Let g_i and h_i be the earliest and latest possible times to start, and f_i be the latest possible time to finish. The early-start, late-start, and late-finish constraints provide strict boundaries for the start and finish times, given by

$$g_i \leq x_i \leq h_i, \qquad y_i \leq f_i.$$

2.2 Optimization criteria

To describe different scheduling objectives, we use several criteria that commonly arise in the development of optimal schedules in real-world problems. The criteria are written below in the form ready for further translation into terms of tropical mathematics.

We begin with the makespan, which is the interval between the earliest start time and the latest finish time of activities in a project. The makespan describes the total duration of the project and finds wide application as a natural objective function to be minimized in scheduling problems. With the notation introduced above, the makespan is given by

$$\max_{1 \leq i \leq n} y_i - \min_{1 \leq i \leq n} x_i = \max_{1 \leq i \leq n} y_i + \max_{1 \leq i \leq n} (-x_i).$$

Another important criterion takes the form of the maximum absolute deviation of finish times of activities from given due dates for a project. The minimum value of this criterion corresponds to the minimal violation of due dates, which can be accomplished in the project. The maximum deviation from due dates is written as

$$\max_{1 \leq i \leq n} |y_i - d_i| = \max_{1 \leq i \leq n} \max(y_i - d_i, d_i - y_i).$$

The flow-time of an activity (also known as the system, throughput and turn-around time) is defined as the difference between its start and finish times, and can determine expenses related to undertaking the activity in the project. The flow-time of activity i is bounded from below by the value of a_{ii}, which is commonly assumed to be nonnegative. In general, the flow-time may be greater than this bound due to other temporal constraints.

In many real-world problems, the objective is formulated to minimize the maximum flow-time taken over all activities. The maximum flow-time is described by the expression

$$\max_{1\leq i\leq n}(y_i - x_i).$$

Finally, we consider the maximum deviation of completion times of all activities. The minimization of this criterion is equivalent to finding a schedule, where all activities have to finish simultaneously as much as possible. Such a problem can arise in just-in-time manufacturing, when certain delivery operations must be completed at once. The maximum deviation of completion times is given by

$$\max_{1\leq i\leq n} y_i - \min_{1\leq i\leq n} y_i = \max_{1\leq i\leq n} y_i + \max_{1\leq i\leq n}(-y_i).$$

2.3 Examples of scheduling problems

We conclude with typical examples of scheduling problems, which are to serve to both motivate and illustrate the results in the rest of the paper. To formulate the problems, we use the notation and formulae introduced above to represent the unknown variables and given parameters, as well as to write the constraints and objectives for scheduling.

2.3.1 Minimization of maximum flow-time

First, we consider a problem of minimization of maximum flow-time under start-to-finish, start-to-start, finish-to-start and early-start temporal constraints. Given a_{ij}, b_{ij}, c_{ij} and g_i for all $i, j = 1,\ldots,n$, the problem is to find the start and finish times x_i and y_i for each activity $i = 1,\ldots,n$, that

$$\begin{aligned} \text{minimize} \quad & \max_{1\leq i\leq n}(y_i - x_i), \\ \text{subject to} \quad & \max_{1\leq j\leq n}(a_{ij} + x_j) = y_i, \quad \max_{1\leq j\leq n}(b_{ij} + x_j) \leq x_i, \\ & \max_{1\leq j\leq n}(c_{ij} + y_j) \leq x_i, \quad g_i \leq x_i, \qquad i = 1,\ldots,n. \end{aligned} \tag{1}$$

2.3.2 Minimization of maximum deviation from due dates

We now formulate a problem to minimize the maximum deviation from due dates under start-to-finish, start-to-start, finish-to-start and due dates constraints. Given parameters a_{ij}, b_{ij}, c_{ij} and d_i, the problem seeks to obtain the unknowns x_i and y_i that

$$\begin{aligned} \text{minimize} \quad & \max_{1\leq i\leq n} \max(y_i - d_i, d_i - y_i), \\ \text{subject to} \quad & \max_{1\leq j\leq n}(a_{ij} + x_j) = y_i, \quad \max_{1\leq j\leq n}(b_{ij} + x_j) \leq x_i, \\ & \max_{1\leq j\leq n}(c_{ij} + y_j) \leq x_i, \qquad i = 1,\ldots,n. \end{aligned} \tag{2}$$

2.3.3 Minimization of makespan

Suppose that we need to minimize the makespan under start-to-finish, early-start, late-start and late-finish constraints. Given parameters a_{ij}, g_i, h_i and f_i, we find x_i and y_i that solve the problem

$$\begin{aligned} &\text{minimize} && \max_{1\le i\le n} y_i + \max_{1\le i\le n}(-x_i), \\ &\text{subject to} && \max_{1\le j\le n}(a_{ij}+x_j) = y_i, \quad g_i \le x_i \le h_i, \\ & && y_i \le f_i, \qquad i = 1,\dots,n. \end{aligned} \tag{3}$$

2.3.4 Minimization of maximum deviation of finish times

Consider the problem of minimizing the maximum deviation of finish times under start-to-finish, start-to-start, finish-to-start, and late-finish constraints: given a_{ij}, b_{ij}, c_{ij} and f_i, find x_i and y_i, that

$$\begin{aligned} &\text{minimize} && \max_{1\le i\le n} y_i + \max_{1\le i\le n}(-y_i), \\ &\text{subject to} && \max_{1\le j\le n}(a_{ij}+x_j) = y_i, \quad \max_{1\le j\le n}(b_{ij}+x_j) \le x_i, \\ & && \max_{1\le j\le n}(c_{ij}+y_j) \le x_i, \quad y_i \le f_i, \qquad i = 1,\dots,n. \end{aligned} \tag{4}$$

Note that such problems can normally be solved using iterative computational algorithms (see [9, 29, 33, 34] for overviews of available solutions). In addition, these problems can be reformulated as linear programming problems to solve them by applying one of the computational methods of linear programming. Below, we provide new solutions to the problems, which are based on optimization methods in tropical mathematics, and offer results in a compact explicit vector form rather than in the form of a numerical algorithm.

3 Preliminary algebraic definitions and results

In this section, we give a brief overview of preliminary definitions, notation and results of tropical algebra to provide a formal basis for describing and solving tropical optimization problems as well as for applying them in project scheduling in the next sections. Both introductory and advanced material on tropical mathematics is provided in many publications, including [1, 3, 5, 12–14, 17, 28, 32] to name only a few. The overview below is mainly based on the presentation of results in [21, 23, 24, 27], which provides a useful framework to obtain direct solutions to the problems under study in a compact vector form.

3.1 Idempotent semifield

We consider a system $(\mathbb{X}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$, where $\mathbb{X}$ is a set closed under addition $\oplus$ and multiplication $\otimes$ with zero $\mathbb{0}$ and identity $\mathbb{1}$, such that $(\mathbb{X}, \oplus, \mathbb{0})$ is a commutative idempotent monoid, $(\mathbb{X}\setminus\{\mathbb{0}\}, \otimes, \mathbb{1})$ is an Abelian group, multiplication is distributive over addition, and $\mathbb{0}$ is absorbing for multiplication. This system is usually called the idempotent semifield.

Addition is idempotent, which means that $x \oplus x = x$ for each $x \in \mathbb{X}$. The idempotent addition induces on $\mathbb{X}$ a partial order such that $x \le y$ if and only if $x \oplus y = y$. It follows

directly from the definition that $x \leq x \oplus y$ and $y \leq x \oplus y$ for all $x,y \in \mathbb{X}$. Moreover, the inequality $x \oplus y \leq z$ appears to be equivalent to the two inequalities $x \leq z$ and $y \leq z$, and both addition and multiplication are monotone in each argument. Finally, we assume that the partial order is extendable to a total order, and thus consider $\mathbb{X}$ to be linearly ordered.

Multiplication is invertible to let each nonzero $x \in \mathbb{X}$ have the inverse x^{-1} such that $x \otimes x^{-1} = \mathbb{1}$. The inverse operation is antitone, which implies that, for all nonzero x and y, the inequality $x \leq y$ yields $x^{-1} \geq y^{-1}$ and vise versa.

The power notation with integer exponents is used to represent repeated multiplication defined as $x^0 = \mathbb{1}$, $x^p = x^{p-1} \otimes x$ and $x^{-p} = (x^{-1})^p$ for all nonzero x and positive integer p. The integer power is assumed to extend to rational exponents to make $\mathbb{X}$ algebraically closed (algebraically complete, radicable). In the algebraic expressions below, we omit the multiplication sign to save writing, and read the exponents only in the above sense.

A typical example of the idempotent semifield under consideration is the real semifield $\mathbb{R}_{\max,+} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$, where the addition $\oplus$ is defined as maximum and the multiplication $\otimes$ as ordinary addition, with the zero $\mathbb{0}$ given by $-\infty$ and the identity $\mathbb{1}$ by 0. Each number $x \in \mathbb{R}$ has the inverse x^{-1}, which is equal to the opposite number $-x$ in the conventional notation. For all $x,y \in \mathbb{R}$, the power x^y is well-defined and coincides, in ordinary arithmetic, with the product xy. The partial order induced by the idempotent addition corresponds to the standard linear order given on $\mathbb{R}$.

As another example, consider the semifield $\mathbb{R}_{\min,\times} = (\mathbb{R}_+ \cup \{+\infty\}, \min, \times, +\infty, 1)$, where $\mathbb{R}_+$ is the set of positive real numbers, $\oplus = \min$, $\otimes = \times$, $\mathbb{0} = +\infty$ and $\mathbb{1} = 1$. In this semifield, the notation of inverses and exponents has the standard interpretation. The partial order defined by addition extends to a linear order that is opposite to the standard order on $\mathbb{R}$.

3.2 Matrices and vectors

We now examine matrices and vectors over the idempotent semifield introduced above. The set of matrices that have m rows and n columns with entries from $\mathbb{X}$ is denoted $\mathbb{X}^{m \times n}$. A matrix, in which all entries are $\mathbb{0}$, is the zero matrix. A matrix is called row-regular (column-regular), if it has no row (column) consisting entirely of $\mathbb{0}$. Provided that a matrix is both row-regular and column-regular, it is called regular.

For any matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{X}^{m \times n}$ and $\boldsymbol{C} \in \mathbb{X}^{n \times l}$, and a scalar $x \in \mathbb{X}$, the matrix addition, matrix multiplication and scalar multiplication follow the standard rules with the scalar operations $\oplus$ and $\otimes$ in the place of the ordinary addition and multiplication, given by

$$\{\boldsymbol{A} \oplus \boldsymbol{B}\}_{ij} = \{\boldsymbol{A}\}_{ij} \oplus \{\boldsymbol{B}\}_{ij}, \qquad \{\boldsymbol{AC}\}_{ij} = \bigoplus_{k=1}^{n} \{\boldsymbol{A}\}_{ik} \{\boldsymbol{C}\}_{kj}, \qquad \{x\boldsymbol{A}\}_{ij} = x\{\boldsymbol{A}\}_{ij}.$$

The partial order associated with the idempotent addition and its properties extend to those on the set of matrices, where the relations are expanded entry-wise.

Consider any matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{X}^{m \times n}$. The transpose of $\boldsymbol{A}$ is the matrix $\boldsymbol{A}^T \in \mathbb{X}^{n \times m}$.

The multiplicative conjugate transpose of $\boldsymbol{A}$ is the matrix $\boldsymbol{A}^- = (a_{ij}^-)$, where $a_{ij}^- = a_{ji}^{-1}$ if $a_{ji} \neq \mathbb{0}$, and $a_{ij}^- = \mathbb{0}$ otherwise.

Consider square matrices of order n, which form the set $\mathbb{X}^{n \times n}$. A matrix which has the diagonal entries equal to $\mathbb{1}$ and all off-diagonal entries equal to $\mathbb{0}$ is the identity matrix $\boldsymbol{I}$. The power notation with nonnegative integer exponent serves to represent iterated products as $\boldsymbol{A}^0 = \boldsymbol{I}$ and $\boldsymbol{A}^p = \boldsymbol{A}^{p-1}\boldsymbol{A}$ for any square matrix $\boldsymbol{A}$ and positive integer p.

For any matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{X}^{n\times n}$, the trace is given by

$$\operatorname{tr} \boldsymbol{A} = \bigoplus_{i=1}^{n} a_{ii}.$$

Every matrix that has only one row (column) is considered a row (column) vector. All vectors below are column vectors unless otherwise specified. The set of column vectors of order n is denoted $\mathbb{X}^n$. A vector with all elements equal to $\mathbb{0}$ is the zero vector. If a vector has no zero elements, it is called regular. The vector of all ones is $\mathbf{1} = (\mathbb{1}, \ldots, \mathbb{1})^T$.

Let $\boldsymbol{A} \in \mathbb{X}^{n\times n}$ be a row regular matrix and $\boldsymbol{x} \in \mathbb{X}^n$ be a regular vector. Then, the vector $\boldsymbol{A}\boldsymbol{x}$ is regular. If the matrix $\boldsymbol{A}$ is column regular, then the row vector $\boldsymbol{x}^T \boldsymbol{A}$ is also regular.

The multiplicative conjugate transpose of a nonzero column vector $\boldsymbol{x} = (x_i) \in \mathbb{X}^n$ is a row vector $\boldsymbol{x}^- = (x_i^-)$ with the entries $x_i^- = x_i^{-1}$ if $x_i \neq \mathbb{0}$, and $x_i = \mathbb{0}$ otherwise.

It is not difficult to verify that the conjugate transposition has the following useful properties. First, the equality $\boldsymbol{x}^-\boldsymbol{x} = \mathbb{1}$ is valid for any nonzero vector $\boldsymbol{x}$.

Furthermore, suppose that $\boldsymbol{x}$ and $\boldsymbol{y}$ are regular vectors of the same order. Then, it is easy to see that the element-wise inequality $\boldsymbol{x} \leq \boldsymbol{y}$ is equivalent to $\boldsymbol{x}^- \geq \boldsymbol{y}^-$. In addition, the matrix inequality $\boldsymbol{x}\boldsymbol{y}^- \geq (\boldsymbol{x}^-\boldsymbol{y})^{-1}\boldsymbol{I}$ holds, and becomes $\boldsymbol{x}\boldsymbol{x}^- \geq \boldsymbol{I}$ when $\boldsymbol{y} = \boldsymbol{x}$.

Finally, consider a square matrix $\boldsymbol{A} \in \mathbb{X}^{n\times n}$. A scalar $\lambda \in \mathbb{X}$ is an eigenvalue of $\boldsymbol{A}$, if there exists a nonzero vector $\boldsymbol{x} \in \mathbb{X}^n$ to satisfy the equality $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. The maximum eigenvalue with respect to the order defined on $\mathbb{X}$ is called the spectral radius and given by

$$\lambda = \bigoplus_{k=1}^{n} \operatorname{tr}^{1/k}(\boldsymbol{A}^k).$$

3.3 Solution to linear inequalities

We conclude the overview of the preliminary definitions and results with the solution of linear inequalities to be used below in the analysis of tropical optimization problems.

Suppose that, given a matrix $\boldsymbol{A} \in \mathbb{X}^{m\times n}$ and a regular vector $\boldsymbol{d} \in \mathbb{X}^m$, we need to find vectors $\boldsymbol{x} \in \mathbb{X}^n$ that satisfy the inequality

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{d}. \tag{5}$$

A direct complete solution to the problem under various assumptions is obtained in [23, 27] in the following form.

Lemma 1 *For any column-regular matrix $\boldsymbol{A}$ and regular vector $\boldsymbol{d}$, all solutions to inequality* (5) *are given by*

$$\boldsymbol{x} \leq (\boldsymbol{d}^-\boldsymbol{A})^-.$$

Furthermore, we consider the problem: given a matrix $\boldsymbol{A} \in \mathbb{X}^{n\times n}$, find regular vectors $\boldsymbol{x} \in \mathbb{X}^n$ to solve the inequality

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{x}. \tag{6}$$

To represent a solution to the problem for any matrix $\boldsymbol{A} \in \mathbb{X}^{n\times n}$, we define a function that takes $\boldsymbol{A}$ to produce the scalar

$$\operatorname{Tr}(\boldsymbol{A}) = \bigoplus_{k=1}^{n} \operatorname{tr} \boldsymbol{A}^k,$$

and use the asterisk operator (the Kleene star), which maps $\boldsymbol{A}$ to the matrix

$$\boldsymbol{A}^* = \bigoplus_{k=0}^{n-1} \boldsymbol{A}^k.$$

The next result obtained in [24, 26, 27] by using various arguments offers a direct, complete solution to inequality (6).

Theorem 1 *For any matrix $\boldsymbol{A}$, the following statements hold:*

1. *If* $\mathrm{Tr}(\boldsymbol{A}) \leq \mathbb{1}$, *then all regular solutions to* (6) *are given by* $\boldsymbol{x} = \boldsymbol{A}^* \boldsymbol{u}$, *where* $\boldsymbol{u}$ *is any regular vector.*
2. *If* $\mathrm{Tr}(\boldsymbol{A}) > \mathbb{1}$, *then there is no regular solution.*

4 Tropical optimization problems

Tropical optimization problems present an area in tropical mathematics, which is of both theoretical interest and practical importance (see, e.g., [22] for an overview, and [5, 7, 38] for further details on particular problems). Many problems are formulated in the framework of tropical mathematics to minimize or maximize nonlinear functions defined on vectors over idempotent semifields and calculated using multiplicative conjugate transposition of vectors. These problems may have constraints given by linear inequalities and equalities.

There are problems that can be solved directly in a rather general setting. For other problems, only algorithmic solution are known in the form of an iterative computational scheme to produce a particular solution, if there is any, or signify that no solutions exist.

The purpose of this section is twofold: first, to offer representative examples to demonstrate a variety of optimization problems under study, and second, to provide an efficient basis for the solution of scheduling problems in the next section. We consider examples of both unconstrained and constrained optimization problems with different objective functions defined in the common setting in terms of a general idempotent semifield. For all problems, direct solutions are given in a compact vector form ready for further analysis and straightforward computations. For some problems, the solutions obtained are complete solutions.

We start with the following problem: given matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{X}^{n \times n}$ and a vector $\boldsymbol{g} \in \mathbb{X}^n$, find regular vectors $\boldsymbol{x} \in \mathbb{X}^n$ that

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{x}^- \boldsymbol{A} \boldsymbol{x}, \\ &\text{subject to} \quad \boldsymbol{B}\boldsymbol{x} \oplus \boldsymbol{g} \leq \boldsymbol{x}. \end{aligned} \tag{7}$$

A direct complete solution to the problem is given in [21, 24] as follows.

Theorem 2 *Let $\boldsymbol{A}$ be a matrix with spectral radius $\lambda > \mathbb{0}$ and $\boldsymbol{B}$ a matrix with* $\mathrm{Tr}(\boldsymbol{B}) \leq \mathbb{1}$. *Then, the minimum value in problem* (7) *is equal to*

$$\theta = \lambda \oplus \bigoplus_{k=1}^{n-1} \bigoplus_{1 \leq i_1 + \cdots + i_k \leq n-k} \mathrm{tr}^{1/k}(\boldsymbol{A}\boldsymbol{B}^{i_1} \cdots \boldsymbol{A}\boldsymbol{B}^{i_k}),$$

and all regular solutions are given by

$$\boldsymbol{x} = (\theta^{-1}\boldsymbol{A} \oplus \boldsymbol{B})^* \boldsymbol{u}, \qquad \boldsymbol{u} \geq \boldsymbol{g}.$$

Furthermore, suppose that, given a matrix $\boldsymbol{A} \in \mathbb{X}^{n\times n}$ and vectors $\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h} \in \mathbb{X}^n$, we need to find regular vectors $\boldsymbol{x} \in \mathbb{X}^n$ that solve the problem

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{x}^- \boldsymbol{A}\boldsymbol{x}, \\ &\text{subject to} \quad \boldsymbol{g} \le \boldsymbol{x} \le \boldsymbol{h}. \end{aligned} \tag{8}$$

The following direct exact solution is proposed in [21].

Theorem 3 *Let $\boldsymbol{A}$ be a matrix with spectral radius $\lambda > \mathbb{0}$, and $\boldsymbol{h}$ be a regular vector such that $\boldsymbol{h}^- \boldsymbol{g} \le \mathbb{1}$. Then the minimum value in problem* (8) *is equal to*

$$\theta = \lambda \oplus \bigoplus_{k=1}^{n-1} (\boldsymbol{h}^- \boldsymbol{A}^k \boldsymbol{g})^{1/k},$$

and all regular solutions are given by

$$\boldsymbol{x} = (\theta^{-1}\boldsymbol{A})^* \boldsymbol{u}, \qquad \boldsymbol{g} \le \boldsymbol{u} \le (\boldsymbol{h}^- (\theta^{-1}\boldsymbol{A})^*)^-.$$

Given a matrix $\boldsymbol{A} \in \mathbb{X}^{m\times n}$ and a vector $\boldsymbol{d} \in \mathbb{X}^m$, consider the problem to find regular vectors $\boldsymbol{x} \in \mathbb{X}^n$ that

$$\text{minimize} \quad \boldsymbol{d}^- \boldsymbol{A}\boldsymbol{x} \oplus (\boldsymbol{A}\boldsymbol{x})^- \boldsymbol{d}. \tag{9}$$

A direct solution proposed to the problem in [18, 25, 27] is as follows.

Theorem 4 *Let $\boldsymbol{A}$ be a row-regular matrix and $\boldsymbol{d}$ be a regular vector. Then, the minimum value in problem* (9) *is equal to*

$$\Delta = ((\boldsymbol{A}(\boldsymbol{d}^- \boldsymbol{A})^-)^- \boldsymbol{d})^{1/2},$$

and the maximum solution is given by

$$\boldsymbol{x} = \Delta (\boldsymbol{d}^- \boldsymbol{A})^-.$$

Finally, we present a solution to the problem: given matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{X}^{m\times n}$ and vectors $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{X}^m$, find regular vectors $\boldsymbol{x} \in \mathbb{X}^n$ to

$$\text{minimize} \quad \boldsymbol{q}^- \boldsymbol{B}\boldsymbol{x} (\boldsymbol{A}\boldsymbol{x})^- \boldsymbol{p}. \tag{10}$$

The next statement offers a direct solution to the problem [19].

Theorem 5 *Let $\boldsymbol{A}$ be row-regular and $\boldsymbol{B}$ column-regular matrices, $\boldsymbol{p}$ be nonzero and $\boldsymbol{q}$ regular vectors. Then, the minimum value in problem* (10) *is equal to*

$$\Delta = (\boldsymbol{A}(\boldsymbol{q}^- \boldsymbol{B})^-)^- \boldsymbol{p},$$

and attained at any vector

$$\boldsymbol{x} = \alpha (\boldsymbol{q}^- \boldsymbol{B})^-, \qquad \alpha > \mathbb{0}.$$

5 Application to project scheduling

We are now in a position to derive solutions to the scheduling problems formulated in the beginning of the paper. In this section, we first represent each problem in the framework of the idempotent semifield $\mathbb{R}_{\max,+}$ in both scalar and vector forms, and then solve this problem by reducing to an optimization problem of the previous section.

5.1 Minimization of maximum flow-time

Consider problem (1) and describe it in terms of the semifield $\mathbb{R}_{\max,+}$. By replacing the usual operations by those of $\mathbb{R}_{\max,+}$, we obtain the problem to find the unknowns x_i and y_i for all $i = 1,\dots,n$, which

$$\begin{aligned} &\text{minimize} && \bigoplus_{i=1}^{n} x_i^{-1} y_i, \\ &\text{subject to} && \bigoplus_{j=1}^{n} a_{ij} x_j = y_i, \quad \bigoplus_{j=1}^{n} b_{ij} x_j \le x_i, \quad \bigoplus_{j=1}^{n} c_{ij} y_j \le x_i, \\ & && g_i \le x_i, \qquad i = 1,\dots,n. \end{aligned}$$

To put the problem in a vector form, we introduce the following matrix-vector notation:

$$\boldsymbol{A} = (a_{ij}), \qquad \boldsymbol{B} = (b_{ij}), \qquad \boldsymbol{C} = (c_{ij}), \qquad \boldsymbol{g} = (g_i), \qquad \boldsymbol{x} = (x_i).$$

With this notation, the problem is to find vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ that

$$\begin{aligned} &\text{minimize} && \boldsymbol{x}^{-}\boldsymbol{y}, \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{B}\boldsymbol{x} \le \boldsymbol{x}, \quad \boldsymbol{C}\boldsymbol{y} \le \boldsymbol{x}, \\ & && \boldsymbol{g} \le \boldsymbol{x}. \end{aligned} \tag{11}$$

A direct complete solution of the problem is given as follows.

Theorem 6 *Let $\boldsymbol{A}$ be a matrix with spectral radius $\lambda > \mathbb{0}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ be matrices such that* $\mathrm{Tr}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A}) \le \mathbb{1}$. *Then, the minimum value in problem* (11) *is equal to*

$$\theta = \lambda \oplus \bigoplus_{k=1}^{n-1} \bigoplus_{1 \le i_1 + \cdots + i_k \le n-k} \mathrm{tr}^{1/k}(\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{i_1} \cdots \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{i_k}),$$

and all regular solutions are given by

$$\boldsymbol{x} = (\theta^{-1}\boldsymbol{A} \oplus \boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}\boldsymbol{u}, \qquad \boldsymbol{y} = \boldsymbol{A}(\theta^{-1}\boldsymbol{A} \oplus \boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}\boldsymbol{u}, \qquad \boldsymbol{u} \ge \boldsymbol{g}.$$

Proof By substitution of the first equality constraint $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, we eliminate the vector $\boldsymbol{y}$. Then, we combine all inequality constraints into one, to write the problem

$$\begin{aligned} &\text{minimize} && \boldsymbol{x}^{-}\boldsymbol{A}\boldsymbol{x}, \\ &\text{subject to} && (\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})\boldsymbol{x} \oplus \boldsymbol{g} \le \boldsymbol{x}. \end{aligned}$$

Clearly, this problem has the form of that at (7), where $\boldsymbol{B}$ is replaced by $\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A}$. Thus, a direct application of Theorem 2 yields the desired result. □

5.2 Minimization of maximum deviation from due dates

Consider problem (2), which, in terms of the semifield $\mathbb{R}_{\max,+}$, takes the form

$$\begin{aligned} &\text{minimize} && \bigoplus_{i=1}^{n}(d_i^{-1}y_i \oplus y_i^{-1}d_i), \\ &\text{subject to} && \bigoplus_{j=1}^{n} a_{ij}x_j = y_i, \quad \bigoplus_{j=1}^{n} b_{ij}x_j \le x_i, \quad \bigoplus_{j=1}^{n} c_{ij}y_j \le x_i, \qquad i = 1,\dots,n. \end{aligned}$$

In addition to the previously introduced matrix-vector notation, we define the vector $\boldsymbol{d} = (d_i)$, and write the problem as

$$\begin{aligned} &\text{minimize} && \boldsymbol{d}^{-}\boldsymbol{y} \oplus \boldsymbol{y}^{-}\boldsymbol{d}, \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{B}\boldsymbol{x} \le \boldsymbol{x}, \quad \boldsymbol{C}\boldsymbol{y} \le \boldsymbol{x}. \end{aligned} \tag{12}$$

The next result offers a solution to the problem.

Theorem 7 *Let $\boldsymbol{A}$ be a row-regular matrix, $\boldsymbol{B}$ and $\boldsymbol{C}$ matrices such that* $\mathrm{Tr}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A}) \le \mathbb{1}$*, and $\boldsymbol{d}$ be a regular vector. Then, the minimum value in problem* (12) *is equal to*

$$\Delta = ((\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}(\boldsymbol{d}^{-}\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*})^{-})^{-}\boldsymbol{d})^{1/2},$$

and the maximum solution is given by

$$\boldsymbol{x} = \Delta(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})(\boldsymbol{d}^{-}\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*})^{-}, \qquad \boldsymbol{y} = \Delta\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})(\boldsymbol{d}^{-}\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*})^{-}.$$

Proof After substitution $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, we combine both inequality constraints into one inequality $(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})\boldsymbol{x} \le \boldsymbol{x}$. Furthermore, application of Theorem 1 to the last inequality yields $\boldsymbol{x} = (\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}\boldsymbol{u}$, where $\boldsymbol{u}$ is any regular vector.

By substitution of this solution, we reduce problem (12) to the unconstrained problem

$$\text{minimize} \quad \boldsymbol{d}^{-}\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}\boldsymbol{u} \oplus (\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}\boldsymbol{u})^{-}\boldsymbol{d}.$$

This problem has the form of (9) with $\boldsymbol{A}$ replaced by $\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^{*}$. Therefore, we can apply Theorem 4 to obtain a solution in terms of the vector $\boldsymbol{u}$. Turning back to the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we complete the proof. □

5.3 Minimization of makespan

In the framework of the semifield $\mathbb{R}_{\max,+}$, problem (3) is rewritten as

$$\begin{aligned} &\text{minimize} && \bigoplus_{i=1}^{n} y_i \bigoplus_{j=1}^{n} x_j^{-1}, \\ &\text{subject to} && \bigoplus_{j=1}^{n} a_{ij}x_j = y_i, \quad g_i \le x_i \le h_i, \quad y_i \le f_i, \qquad i = 1,\dots,n. \end{aligned}$$

By adding the vector notation $\boldsymbol{f} = (f_i)$ and using $\mathbf{1}$ to indicate the vector of ones, we represent the problem in the form

$$\begin{aligned} &\text{minimize} && \mathbf{1}^{T}\boldsymbol{y}\boldsymbol{x}^{-}\mathbf{1}, \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{g} \le \boldsymbol{x} \le \boldsymbol{h}, \\ & && \boldsymbol{y} \le \boldsymbol{f}. \end{aligned} \tag{13}$$

Theorem 8 *Let $\boldsymbol{A}$ be a nonzero matrix, $\boldsymbol{h}$ and $\boldsymbol{f}$ be regular vectors satisfying the condition $(\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})\boldsymbol{g} \leq \mathbb{1}$. Then, the minimum makespan in problem* (13) *is equal to*

$$\theta = \mathbf{1}^T \boldsymbol{A}(\boldsymbol{I} \oplus \boldsymbol{g}\boldsymbol{h}^-)\mathbf{1},$$

and all regular solutions are given by

$$\boldsymbol{x} = (\boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})\boldsymbol{u}, \qquad \boldsymbol{y} = \boldsymbol{A}(\boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})\boldsymbol{u},$$

where

$$\boldsymbol{g} \leq \boldsymbol{u} \leq ((\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})(\boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A}))^-.$$

Proof As before, we first substitute $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. An application of Lemma 1 to solve the inequality $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{f}$ yields $\boldsymbol{x} \leq (\boldsymbol{f}^- \boldsymbol{A})^-$. Then, we take the two upper boundaries $\boldsymbol{x} \leq \boldsymbol{h}$ and $\boldsymbol{x} \leq (\boldsymbol{f}^- \boldsymbol{A})^-$, and apply conjugate transposition to rewrite them as $\boldsymbol{x}^- \geq \boldsymbol{h}^-$ and $\boldsymbol{x}^- \geq \boldsymbol{f}^- \boldsymbol{A}$. By coupling both inequalities into one, and again taking the conjugate transposition, we obtain one upper bound $\boldsymbol{x} \leq (\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})^-$.

Finally, we represent the objective function $\mathbf{1}^T \boldsymbol{A}\boldsymbol{x}\boldsymbol{x}^-\mathbf{1}$ in its equivalent form $\boldsymbol{x}^-\mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{x}$ to write the problem as

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{x}^-\mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{x}, \\ &\text{subject to} \quad \boldsymbol{g} \leq \boldsymbol{x} \leq (\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})^-. \end{aligned} \tag{14}$$

The problem obtained is of the form of (8), where $\boldsymbol{A}$ is replaced by $\mathbf{1}\mathbf{1}^T \boldsymbol{A}$ and $\boldsymbol{h}$ by $(\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})^-$. To apply Theorem 3, we first calculate

$$(\mathbf{1}\mathbf{1}^T \boldsymbol{A})^k = (\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{k-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A}, \qquad \mathrm{tr}(\mathbf{1}\mathbf{1}^T \boldsymbol{A})^k = (\mathbf{1}^T \boldsymbol{A}\mathbf{1})^k, \qquad k = 1, \ldots, n;$$

from which it directly follows that the spectral radius of the matrix $\mathbf{1}\mathbf{1}^T \boldsymbol{A}$ is equal to

$$\lambda = \mathbf{1}^T \boldsymbol{A}\mathbf{1} > \mathbb{0}.$$

Furthermore, we consider the minimum, which is given by

$$\theta = \mathbf{1}^T \boldsymbol{A}\mathbf{1} \oplus (\mathbf{1}^T \boldsymbol{A}\mathbf{1}) \bigoplus_{k=1}^{n-1} ((\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{-1} \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g})^{1/k}.$$

First, suppose that $\mathbf{1}^T \boldsymbol{A}\mathbf{1} \leq \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g}$. Since the inequality $(\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{-1} \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g} \geq \mathbb{1}$ holds, we have

$$((\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{-1} \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g})^{1/k} \leq (\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{-1} \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g},$$

and hence, $\theta = \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g}$. On the other hand, if $\mathbf{1}^T \boldsymbol{A}\mathbf{1} > \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g}$, then we immediately find that $\theta = \mathbf{1}^T \boldsymbol{A}\mathbf{1}$. By combining both results, we finally obtain

$$\theta = \mathbf{1}^T \boldsymbol{A}\mathbf{1} \oplus \boldsymbol{h}^- \mathbf{1}\mathbf{1}^T \boldsymbol{A}\boldsymbol{g} = \mathbf{1}^T \boldsymbol{A}\mathbf{1} \oplus \mathbf{1}^T \boldsymbol{A}\boldsymbol{g}\boldsymbol{h}^-\mathbf{1} = \mathbf{1}^T \boldsymbol{A}(\boldsymbol{I} \oplus \boldsymbol{g}\boldsymbol{h}^-)\mathbf{1}.$$

To describe the solution set according to Theorem 3, we examine the matrix

$$(\theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})^* = \bigoplus_{k=0}^{n-1} (\theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})^k = \boldsymbol{I} \oplus \theta^{-1} \bigoplus_{k=1}^{n-1} (\theta^{-1}\mathbf{1}^T \boldsymbol{A}\mathbf{1})^{k-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A}.$$

Considering that $\theta \geq \mathbf{1}^T \boldsymbol{A}\mathbf{1}$, we obtain $(\theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})^* = \boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A}$. Substitution into the solution provided by Theorem 3 yields

$$\boldsymbol{x} = (\boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A})\boldsymbol{u}, \qquad \boldsymbol{g} \leq \boldsymbol{u} \leq ((\boldsymbol{h}^- \oplus \boldsymbol{f}^- \boldsymbol{A})(\boldsymbol{I} \oplus \theta^{-1}\mathbf{1}\mathbf{1}^T \boldsymbol{A}))^-.$$

Finally, we represent the vector $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, which completes the proof. □

5.4 Minimization of maximum deviation of finish times

After rewriting problem (4) in terms of $\mathbb{R}_{\max,+}$, the problem becomes

$$\begin{aligned}
\text{minimize} \quad & \bigoplus_{i=1}^{n} y_i \bigoplus_{j=1}^{n} y_j^{-1}, \\
\text{subject to} \quad & \bigoplus_{j=1}^{n} a_{ij} x_j = y_i, \quad \bigoplus_{j=1}^{n} b_{ij} x_j \leq x_i, \\
& \bigoplus_{j=1}^{n} c_{ij} y_j \leq x_i, \quad y_i \leq f_i, \qquad i = 1, \ldots, n.
\end{aligned}$$

Switching to matrix-vector notation puts the problem in the form

$$\begin{aligned}
\text{minimize} \quad & \mathbf{1}^T \boldsymbol{y} \boldsymbol{y}^{-} \mathbf{1}, \\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \quad \boldsymbol{B}\boldsymbol{x} \leq \boldsymbol{x}, \quad \boldsymbol{C}\boldsymbol{y} \leq \boldsymbol{x}, \\
& \boldsymbol{y} \leq \boldsymbol{f}.
\end{aligned} \tag{15}$$

The following result offers a solution to the problem.

Theorem 9 *Let $\boldsymbol{A}$ be row-regular and $\boldsymbol{B}$ column-regular matrices, $\boldsymbol{p}$ be nonzero and $\boldsymbol{q}$ regular vectors. Then, the minimum value in problem* (15) *is equal to*

$$\Delta = (\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-})^{-} \mathbf{1},$$

and attained if

$$\boldsymbol{x} = \alpha(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-}, \qquad \boldsymbol{y} = \alpha \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-},$$

where

$$\alpha \leq (\boldsymbol{f}^{-} \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-})^{-1}.$$

Proof After substitution $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, we combine the first two inequalities in the constraints into one inequality $(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})\boldsymbol{x} \leq \boldsymbol{x}$. This inequality is then solved by using Theorem 1 to obtain $\boldsymbol{x} = (\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* \boldsymbol{u}$, where $\boldsymbol{u}$ is a regular vector.

Furthermore, we write the last inequality in the constraints as $\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* \boldsymbol{u} \leq \boldsymbol{f}$, and apply Lemma 1 to find $\boldsymbol{u} \leq (\boldsymbol{f}^{-} \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-}$. The problem takes the form

$$\begin{aligned}
\text{minimize} \quad & \mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* \boldsymbol{u} (\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* \boldsymbol{u})^{-} \mathbf{1}, \\
\text{subject to} \quad & \boldsymbol{u} \leq (\boldsymbol{f}^{-} \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-}.
\end{aligned}$$

First, we remove the constraints and solve the obtained unconstrained problem. By applying Theorem 5, where both matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are replaced by $\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*$, and both vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ by $\mathbf{1}$, we find the minimum

$$\Delta = (\boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-})^{-} \mathbf{1},$$

which is attained at the vector

$$\boldsymbol{u} = \alpha(\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{C}\boldsymbol{A})^*)^{-}, \qquad \alpha > \mathbb{0}.$$

To find the values of α, which satisfy the constraint $\boldsymbol{u} \leq (\boldsymbol{f}^- \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{CA})^*)^-$, we solve the inequality

$$\alpha(\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{CA})^*)^- \leq (\boldsymbol{f}^- \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{CA})^*)^-.$$

By applying Lemma 1 with α as the unknown, we have

$$\alpha \leq (\boldsymbol{f}^- \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{CA})^* (\mathbf{1}^T \boldsymbol{A}(\boldsymbol{B} \oplus \boldsymbol{CA})^*)^-)^{-1}.$$

It remains to turn back to vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ to complete the proof. □

Acknowledgements The author is very grateful to three referees for their careful reading of a previous draft of this paper, and for their constructive suggestions, which have been incorporated in the final version.

References

1. Akian, M., Bapat, R., Gaubert, S.: Max-plus algebra. In: L. Hogben (ed.) Handbook of Linear Algebra, Discrete Mathematics and Its Applications, pp. 25-1–25-17. Taylor and Francis, Boca Raton, FL (2007). DOI 10.1201/9781420010572.ch25
2. Aminu, A., Butkovič, P.: Non-linear programs with max-linear constraints: A heuristic approach. IMA J. Manag. Math. **23**(1), 41–66 (2012). DOI 10.1093/imaman/dpq020
3. Baccelli, F.L., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Wiley Series in Probability and Statistics. Wiley, Chichester (1993)
4. Bouquard, J.L., Lenté, C., Billaut, J.C.: Application of an optimization problem in max-plus algebra to scheduling problems. Discrete Appl. Math. **154**(15), 2064–2079 (2006). DOI 10.1016/j.dam.2005.04.011
5. Butkovič, P.: Max-linear Systems: Theory and Algorithms. Springer Monographs in Mathematics. Springer, London (2010). DOI 10.1007/978-1-84996-299-5
6. Butkovič, P., Tam, K.P.: On some properties of the image set of a max-linear mapping. In: G.L. Litvinov, S.N. Sergeev (eds.) Tropical and Idempotent Mathematics, *Contemp. Math.*, vol. 495, pp. 115–126. American Mathematical Society (2009). DOI 10.1090/conm/495/09694
7. Cuninghame-Green, R.: Minimax Algebra, *Lecture Notes in Economics and Mathematical Systems*, vol. 166. Springer, Berlin (1979)
8. Cuninghame-Green, R.A.: Describing industrial processes with interference and approximating their steady-state behaviour. Oper. Res. Quart. **13**(1), 95–100 (1962). DOI 10.2307/3007584
9. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. International Series in Operations Research and Management Science. Kluwer Acad. Publ., New York (2002)
10. Fiedler, M., Nedoma, J., Ramík, J., Rohn, J., Zimmermann, K.: Linear Optimization Problems with Inexact Data. Springer, Berlin (2006). DOI 10.1007/0-387-32698-7
11. Giffler, B.: Scheduling general production systems using schedule algebra. Naval Res. Logist. Quart. **10**(1), 237–255 (1963). DOI 10.1002/nav.3800100119
12. Golan, J.S.: Semirings and Affine Equations Over Them: Theory and Applications, *Mathematics and Its Applications*, vol. 556. Kluwer Acad. Publ., Dordrecht (2003)
13. Gondran, M., Minoux, M.: Graphs, Dioids and Semirings: New Models and Algorithms, *Operations Research / Computer Science Interfaces*, vol. 41. Springer, New York (2008). DOI 10.1007/978-0-387-75450-5
14. Heidergott, B., Olsder, G.J., van der Woude, J.: Max-plus at Work: Modeling and Analysis of Synchronized Systems. Princeton Series in Applied Mathematics. Princeton Univ. Press, Princeton, NJ (2006)
15. Hoffman, A.J.: On abstract dual linear programs. Naval Res. Logist. Quart. **10**(1), 369–373 (1963). DOI 10.1002/nav.3800100131
16. Houssin, L.: Cyclic jobshop problem and (max,plus) algebra. In: S. Bittanti, A. Cenedese, S. Zampieri (eds.) Proceedings of the 18th IFAC World Congress, 2011, *World Congress*, vol. 18, pp. 2717–2721. IFAC (2011). DOI 10.3182/20110828-6-IT-1002.03095
17. Kolokoltsov, V.N., Maslov, V.P.: Idempotent Analysis and Its Applications, *Mathematics and Its Applications*, vol. 401. Kluwer Acad. Publ., Dordrecht (1997)
18. Krivulin, N.: A solution of a tropical linear vector equation. In: S. Yenuri (ed.) Advances in Computer Science, *Recent Advances in Computer Engineering Series*, vol. 5, pp. 244–249. WSEAS Press (2012)

19. Krivulin, N.: Explicit solution of a tropical optimization problem with application to project scheduling. In: D. Biolek, H. Walter, I. Utu, C. von Lucken (eds.) Mathematical Methods and Optimization Techniques in Engineering, pp. 39–45. WSEAS Press (2013)
20. Krivulin, N.: Complete solution of a constrained tropical optimization problem with application to location analysis. In: P. Höfner, P. Jipsen, W. Kahl, M.E. Müller (eds.) Relational and Algebraic Methods in Computer Science, *Lecture Notes in Computer Science*, vol. 8428, pp. 362–378. Springer, Cham (2014). DOI 110.1007/978-3-319-06251-8_22
21. Krivulin, N.: A constrained tropical optimization problem: Complete solution and application example. In: G.L. Litvinov, S.N. Sergeev (eds.) Tropical and Idempotent Mathematics and Applications, *Contemp. Math.*, vol. 616, pp. 163–177. AMS, Providence, RI (2014). DOI 10.1090/conm/616/12308
22. Krivulin, N.: Tropical optimization problems. In: L.A. Petrosyan, J.V. Romanovsky, D.W.K. Yeung (eds.) Advances in Economics and Optimization: Collected Scientific Studies Dedicated to the Memory of L. V. Kantorovich, Economic Issues, Problems and Perspectives, pp. 195–214. Nova Science Publ., New York (2014)
23. Krivulin, N.: Extremal properties of tropical eigenvalues and solutions to tropical optimization problems. Linear Algebra Appl. **468**, 211–232 (2015). DOI 10.1016/j.laa.2014.06.044
24. Krivulin, N.: A multidimensional tropical optimization problem with nonlinear objective function and linear constraints. Optimization **64**(5), 1107–1129 (2015). DOI 10.1080/02331934.2013.840624
25. Krivulin, N.K.: On solution of linear vector equations in idempotent algebra. In: M.K. Chirkov (ed.) Mathematical Models. Theory and Applications. Issue 5, pp. 105–113. Saint Petersburg University, St. Petersburg (2004). (in Russian)
26. Krivulin, N.K.: Solution of generalized linear vector equations in idempotent algebra. Vestnik St. Petersburg Univ. Math. **39**(1), 16–26 (2006)
27. Krivulin, N.K.: Methods of Idempotent Algebra for Problems in Modeling and Analysis of Complex Systems. Saint Petersburg Univ. Press, St. Petersburg (2009). (in Russian)
28. Litvinov, G.: Maslov dequantization, idempotent and tropical mathematics: A brief introduction. J. Math. Sci. (NY) **140**(3), 426–444 (2007). DOI 10.1007/s10958-007-0450-5
29. Neumann, K., Schwindt, C., Zimmermann, J.: Project Scheduling with Time Windows and Scarce Resources, 2 edn. Springer, Berlin (2003). DOI 10.1007/978-3-540-24800-2
30. Pandit, S.N.N.: A new matrix calculus. J. SIAM **9**(4), 632–639 (1961). DOI 10.1137/0109052
31. Romanovskiĭ, I.V.: Asymptotic behavior of dynamic programming processes with a continuous set of states. Soviet Math. Dokl. **5**(6), 1684–1687 (1964)
32. Speyer, D., Sturmfels, B.: Tropical mathematics. Math. Mag. **82**(3), 163–173 (2009)
33. T'kindt, V., Billaut, J.C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Berlin (2006)
34. Vanhoucke, M.: Project Management with Dynamic Scheduling. Springer, Berlin (2013). DOI 10.1007/978-3-642-40438-2
35. Vorob'ev, N.N.: The extremal matrix algebra. Soviet Math. Dokl. **4**(5), 1220–1223 (1963)
36. Zimmermann, K.: Some optimization problems with extremal operations. In: B. Korte, K. Ritter (eds.) Mathematical Programming at Oberwolfach II, *Mathematical Programming Studies*, vol. 22, pp. 237–251. Springer, Berlin (1984). DOI 10.1007/BFb0121020
37. Zimmermann, K.: Disjunctive optimization, max-separable problems and extremal algebras. Theoret. Comput. Sci. **293**(1), 45–54 (2003). DOI 10.1016/S0304-3975(02)00231-1
38. Zimmermann, U.: Linear and Combinatorial Optimization in Ordered Algebraic Structures, *Annals of Discrete Mathematics*, vol. 10. Elsevier, Amsterdam (1981)

MISTA 2015

Thesis Defense Timetabling

Michele Battistutta · Sara Ceschia · Fabio De Cesco · Andrea Schaerf

Abstract The *thesis defense timetabling* problem consists in composing the suitable committee for a set of graduation sessions and assigning each candidate to one of the sessions.

In this work, we define the problem formulation that applies to some Italian universities, and we provide two solution methods based on local search and constraint satisfaction, respectively. In addition, we perform an experimental analysis and comparison on both real-world and artificial instances.

1 Introduction

The thesis defense and the graduation ceremony are ineludible activities of the management of a university. Large departments may have many students graduating at the same time, and consequently need to split the graduation procedure into several sessions, with separate committees and possibly running in different days.

The corresponding timetabling problem consists in both assigning each candidate to one session and composing the suitable committees, satisfying various constraints and objectives.

This is an interesting NP-hard problem that, up to our knowledge, has been little studied in the relatively-large literature on educational timetabling problems (see [6, 8, 11] for surveys).

In this work, we propose a problem formulation obtained by modeling the real-world problem of an Italian university (Department of Psychology of the University of Milano-Bicocca). We develop both a MiniZinc [9] model and a local search technique to solve the problem. We also develop an instance generator that

Michele Battistutta, Sara Ceschia, and Andrea Schaerf
DIEGM, University of Udine, Via delle Scienze 206, 33100 Udine, Italy
E-mail: {michele.battistutta,sara.ceschia,schaerf}@uniud.it

Fabio De Cesco
EasyStaff s.r.l., Via Adriatica, 278 - 33030 Campoformido (UD), Italy.
E-mail: fabio@easystaff.it

produces realistic cases, in order to have enough instances available to test our solvers. We compare the results of some available CP-based MiniZinc engines and our local search solver on a set of instances produced by our generator and a few real-world ones, on a fixed timeout (5 minutes).

The outcome has been that, on the given dataset and timeout, the local search solver clearly outperformed the MiniZinc engines working on the proposed model.

All instances and results are available on the website `https://bitbucket.org/satt/tdtt-instances`.

2 Problem formulation

The thesis defense timetabling (TDTT) problem consist on the assignment of graduation candidates (students) to different sessions. In addition, for every session the committee must be composed by selecting appropriate faculty members. Usually there are two sessions per day (morning and afternoon) and the duration depends on the number of students assigned. If these are not enough to schedule all students, the university adds new sessions, typically in parallel to the ones already scheduled.

Students must be assigned exactly to one session, whereas faculty members can participate to more than one committee (or none). The number of members of the committee is not fixed, but it is bound by limits prescribed by university rules and traditions. In addition, there are limits on the composition of the committee in terms of qualified members, such as full and associate professors.

The presence of students and faculty members is interconnected because each student has a supervisor, that must be in the committee examining the student. In addition, the committee should include also an *opponent* (or *challenger*) that is a faculty member with expertise in the area of the thesis.

Summarizing, the main entities of the problem are:

Faculty members: The list of university staff that can be assigned as committee members. They are characterized by their role (e.g., full professor) and a list of sessions which they are unavailable to attend.

Students: For each student, it is given his/her supervisor and a list of the potential opponents. Supervisors and opponents are necessarily faculty members.

As customary, constraints are divided into hard and soft ones. The former must be always satisfied, the latter compose the objective function.

The hard constraints are:

Supervision: For each student, the supervisor must be present at the session assigned to the student.

Students per session: The number of students assigned to a session must be less than or equal to the given maximum.

Overlapping sessions: In case that two sessions overlap in time, no member can be assigned to the committee of both sessions.

Committee composition: Each committee must be formed respecting the minimum and maximum number of professor for each *academic level* (or role). In detail, there are four levels, which are (starting from the highest): full professor, associate professor, assistant professor, and external teacher.

Minimum and maximum values for *Committee composition* must be interpreted as minimum and maximum number of members with the given level or a higher one. For example, if we require a minimum of four associate professors, this can be also fulfilled with two associate professors and two full professors. Consequently, the limits for the lowest level represent the limits for the total number of members.

Notice that it is not explicitly defined a minimum number of students per session because the number of sessions is already regulated to fit the given number of students.

The soft constraints are:

Multiple duties: Each time a faculty member has to be in more than one committee, a penalty is assigned. In order to avoid high loads, the penalty of the violations is quadratic. That is, if p is the number of presences of a faculty member, the associated penalty is $(p-1)^2$.

Opponent's presence: For every student, there must be at least one of the suggested opponents in the corresponding session. Each student without opponent counts as one violation of this constraint.

Regarding the size of real cases, they can get to 20 sessions, and 150 candidates and 150 faculty members. The typical size of a committee is between 7 and 10 members. The number of possible opponents for each student normally ranges between 1 and 3. However, in some rare cases, it is possible that no opponent is suggested for a student. In this situation, all faculty members are considered suitable opponents for the student.

3 Computational complexity

We now prove that the decision problem underlying TDTT is NP-complete. The following proof considers only one single session, in which all students are assigned to it. This means that the subproblem of selecting the appropriate faculty members for a single session is already NP-complete.

Consider a *set covering* problem with universe U, with $|U| = m$, the sets $S_1, \ldots, S_n \subseteq U$, and an integer c, with $c < n$. It is well known (see [4], problem SP4) that checking if there is a set of c sets that covers the whole U is an NP-complete problem.

We now show that we can build an instance of TDTT such that it has a 0 cost solution if and only if there is a covering of U composed of c sets.

We create a TDTT instance with m students and $n+1$ faculty members, such that all students have as supervisor the faculty member number $n+1$. For each faculty member i from 1 to n, we assign her/him as opponent of all students in S_i. All faculty members are available for the session. Finally we set a maximum number of members equal to $c+1$ (one spot is for the supervisor $n+1$). It is easy to see that if there is a committee in which all students have an opponent, this corresponds to a selection of the sets that covers all elements in U.

4 Solution techniques

4.1 Local search

We developed a solver based on local search that works on a reduced search space: only students (and consequently supervisors) and opponents are assigned to sessions. Then the procedure has a post-processing step that uses a greedy technique to complete the committee with available members, satisfying committee composition constraints.

The main features of our local search algorithm are:

Search space: The search space consists in the assignment of all students to any session, and the selection of one opponent to each of them. The supervisor and the selected opponent are inserted in the committee of the candidate. Infeasible solutions are part of the search space and the violation of hard constraints are penalized in the cost function with a high weight.

Initial solution: The algorithm start from an initial solution in which students are assigned to a random session and the opponent is chosen randomly from the list of potential ones.

Neighborhood relation: The neighborhood relation is composed by the union of four different moves:

1. Assign the student to a different session and/or change the assigned opponent.
2. Swap the sessions of two students.
3. Assign to a different session a group of student that are assigned to the same session and have the same supervisor.
4. Swap the sessions for two groups of students: each group is formed by students in the same session and with the same supervisor.

In neighborhoods 2-4 the selected opponent remains unchanged, and she/he moves to the new session along with the student.

Neighborhoods 3 and 4 were added for a better space exploration since the solver, affected by the constraint of the maximum number of members for committee and by the objective of limit the presence of each member, tends to reach solutions where students with the same supervisor are assigned to the same session. All four types of move have the same probability of being chosen when the algorithm generates a random move.

Since in our search space, only the supervisor and the opponent are assigned to the committee, it is normally necessary to add other members to complete the committee reaching the minimum number. In this case, the solution is processed with a greedy algorithm that assigns to the committee the missing members minimizing multiple duties.

As metaheuristic technique, we use Simulated Annealing, in its “standard” version as proposed in [1].

4.2 Constraint programming

We developed a constraint model in MiniZinc that uses as decision variables:

– an array of integers that assigns to each student the selected session,

– an array of sets that assigns to each session the set of faculty members that compose the committee.

```
array [1..Candidates] of var 1..Sessions: StudentSession;
array [1..Sessions] of var set of 1..FacultyMembers: SessionMembers;
```

Using set variables for the committee composition makes automatically satisfied the constraint that all members must be distinct. In addition, the constraints regarding simultaneous sessions can be easily expressed using the built-in `disjoint` operator between sets.

The constraints about the composition of the committees based on the level of the participants are expressed as follows, where the component `1` of the elements of the array `LimitMembersForLevel` is the minimum and the component `2` is the maximum.

```
constraint forall (s in 1..Sessions, lv in 1..Levels)
    (((sum (p in 1..FacultyMembers)(AcademicLevel[p] <= l
     /\ p in SessionMembers[s]))
        >= LimitMembersForLevel[lv,1]) /\
     ((sum (p in 1..FacultyMembers)(AcademicLevel[p] <= l
     /\ p in SessionMembers[s]))
        <= LimitMembersForLevel[lv,2]));
```

In order to define the objective function to be minimized, we introduce new decision variables that are functionally related to the main ones. We show here the ones defined for the objective component *Opponent's presence* (corresponding ones are used for *Multiple duties*).

In detail, we introduce the following variables, where `missing_opponent` is the variable that is included in the objective function with the given weight.

```
array [1..Candidates] of var 0..FacultyMembers: NumOpponents;
var 0..Candidates: missing_opponent;
```

These new (redundant) variables are linked to the main ones by the following constraints

```
constraint forall (s in 1..Candidates)
     (NumOpponents[s] = sum (op in Opponents[s])
       (op in SessionMembers[StudentSession[s]]));

constraint
   missing_opponent = sum (s in 1..Candidates)
        (NumOpponents[s] == 0 /\ card(Opponents[s]) > 0);
```

The full model is available along with the instances and the results on the website.

5 Experimental analysis

The local search code is written in C++, using the framework EasyLocal++ [3], compiled using `gcc` v. 4.9.1, and parameter tuning has been performed using JSON2RUN [13].

All experiments ran on an Ubuntu Linux 13.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20 GHz) physical cores, hyper-threaded to 32 virtual cores. A single virtual core has been dedicated to each experiment.

5.1 Instances

We used three real-world instances, coming from the Department of Psychology of the University of Milano-Bicocca. In addition, we developed an instance generator parametrized by the number of sessions.

The generator randomly selects the number of students and faculty members based on the limits fixed for the composition of sessions. The maximum number of students for session is a value between 8 and 10 and is used along with the number of sessions to define the total number of students, so that each session has at least 6 students. The generator also pairs a random number of session for the simultaneous constraint, the number of pair simultaneous sessions in a generated instance ranges from 0 to one fourth of the number of sessions.

The committee size and composition are fixed: the committee goes from 7 to 10 members with at least 1 faculty member of rank 1 and three of rank lower or equal to 2. We choose to use these fixed values since they were common for all the real cases that we analyzed. A student normally has up to 3 opponents.

In order to create realistic data for the opponents, all faculty members are assigned to an area. Once the supervisor is randomly selected, the potential opponents are selected among the faculty member of the area of the supervisor and the two adjacent ones. If no member, besides the supervisor, exists in these three areas, the student is left without suggested opponents, so that the opponent can be anyone in the committee.

We have created 20 instances with a number of sessions ranging from 5 to 15, and experiments are performed on these instances. Instances are written in MiniZinc data format and they are available on the website (`https://bitbucket.org/satt/tdtt-instances`), along with the MiniZinc model.

5.2 Tuning

Simulated Annealing has several control parameters: the cooling rate (α), the number of neighbors sampled at each temperature (N), and the starting and final temperatures (T_0 and T_{min}). In order to find the best configuration of these parameters using a statistically-principled approach, we resort to the F-Race procedure [2] with a 95% confidence.

With the aim of equalizing approximately the running times for all different configurations, we fix the total number of iterations $I = 10^6$ and compute the parameter N from the others so that the total I is always the same. This results in an average running time of 5 minutes on our machine.

Preliminary results demonstrate that our solution method is not sensitive to small variations of the cooling rate, thus we decided to set $\alpha = 0.99$ and focus on the other control parameters T_0 and T_{min}.

We test 30 different configurations generated according to the *Hammersley point set* [5] for the ranges whose bounds are $T_0 = [10^0, 10^2]$ and $\rho = [10^1, 10^3]$, with $\rho = T_0/T_{min}$. The winning configuration turned out to be $T_0 = 1.18$ and $T_{min} = 0.104$. All the following experiments have been performed using these values for the parameters.

Instance	MiniZinc *fd*	Gecode	Opturion CPX	Local search avg	Local search best
gtt-art01	–	–	210	3.7	1
gtt-art02	346	386	293	1.2	0
gtt-art03	169	204	167	0.3	0
gtt-art04	347	–	258	6.2	2
gtt-art05	–	–	268	0.2	0
gtt-art06	–	–	265	0.5	0
gtt-art07	–	–	333	1.7	0
gtt-art08	–	–	296	1.1	0
gtt-art09	74	–	67	4.1	2
gtt-art10	–	–	310	1.5	0
gtt-art11	394	450	277	2.0	0
gtt-art12	88	–	118	1.0	1
gtt-art13	227	–	219	8.2	5
gtt-art14	–	–	342	3.3	2
gtt-art15	–	–	224	11.0	7
gtt-art16	461	447	417	7.3	4
gtt-art17	–	–	288	1.7	0
gtt-art18	78	123	111	7.0	6
gtt-art19	–	314	302	1.6	0
gtt-art20	402	–	326	6.4	4
gtt-real01	–	567	289	61.1	59
gtt-real02	68	–	80	0.0	0
gtt-real03	–	904	440	54.1	52

Table 1 Comparison between MiniZinc engines and local search on the testbed.

5.3 Comparison of results

We experimented several engines for MiniZinc. Among those available in the MiniZinc distribution v.2.0 only *fd* and *fdmip* support the *set* construct which is used by our model. However, in Table 1 we decided to show only the results obtained by *fd*, which is the default evaluation algorithm for MiniZinc, given that they have the same performances. In addition, we compare our results with Gecode v.4.3.3 [12], which is a pure constraint programming (CP) solver, and Opturion CPX v.1.0.2 [10], which combines CP and SAT techniques.

In Table 1 we show the cost of the best solution obtained by each solver within 5 minutes of running time. For our local search solver, we also report the average cost of 30 repetitions, obtained with the parameter configuration described in Section 5.2. The dash symbol states that the solver was not able to obtain any solution in the timeout.

The outcome is that the local search solver clearly outperforms all the other solution methods both on generated and real cases within the granted computational time. In detail, it was able to find the perfect solution for 11 instances; in addition in one case (gtt-real02) the perfect solution was obtained in all 30 trials.

6 Discussion and conclusions

We have modeled and solved a timetabling problem that, up to our knowledge, has not been formalized yet in the scientific literature. A similar problem has been considered in [7], in which however constraints and objectives are different; for

example, in their case the committee changes for each single student, so that the order of presentation is also important to minimize the time spent by committee members.

For this new problem, we have collected 3 real cases and developed an instance generator in order to abstain from overtuning on such a small number of available instances.

We have developed both a local search method and a MiniZinc model and tested several MiniZinc engines. The experimental analysis shows that the local search solver outperforms indisputably the MiniZinc engines, at least on the ground (instances and timeout) defined in the work. This is however not a fair comparison, given that the MiniZinc engines are exact solvers and they would need more time to explore their search tree effectively for instances of this size.

This model has been used to generate the graduation calendar for the Faculty of Psychology of the University of Milano-Bicocca and it has been used by the university with satisfactory results.

For the future, we plan to investigate on the management of the thesis defense procedure in other universities, in order to design a more general formulation. In addition, we plan to try to improve the MiniZinc model, with the aim of making it more suitable to be solved with the available engines.

Another future development is to collect more real-life instances, and to use them for the improvement of the generator, in such a way that it can create more realistic instances.

References

1. Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.
2. Mauro Birattari, Z. Yuan, P. Balaprakash, and Thomas Stützle. *F-Race and iterated F-race: An overview*. Springer, Berlin, 2010.
3. Luca Di Gaspero and Andrea Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience*, 33(8):733–765, 2003.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.
5. John Michael Hammersley, David Christopher Handscomb, and George Weiss. Monte Carlo methods. *Physics today*, 18:55, 1965.
6. Jeffrey H. Kingston. Educational timetabling. In A. Sima Uyar, Ender Ozcan, and Neil Urquhart, editors, *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 91–108. Springer Berlin Heidelberg, 2013.
7. Beáta Kochaniková and Hana Rudová. Student scheduling for bachelor state examinations. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-13)*, pages 762–766, 2013.
8. Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.
9. Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming–CP 2007*, pages 529–543. Springer, 2007.
10. Opturion. Opturion CPX website. URL: `http://www.opturion.com/cpx.html`. Viewed: 2nd February 2015.
11. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
12. Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode/Flatzinc website. URL: `http://www.gecode.org/flatzinc.html`. Viewed: 2nd February 2015.
13. Tommaso Urli. json2run: a tool for experiment design & analysis. *CoRR*, abs/1305.1112, 2013.

MISTA 2015

Bacteria Swarm Optimisation Approach for Enrolment-Based Course Timetabling Problems

Khalid Shaker • Salwani Abdullah • Arwa Alqudsi

Abstract The university course timetabling problem is known as a NP-hard problem. It is also sometimes known as a class/teacher timetabling that refers to a set of courses that need to be scheduled into a given number of rooms and timeslots within a week and, at the same time, students and teachers are assigned to courses so that the meetings can take place. A novel population based approach is proposed in this paper called a Bacteria Swarm Optimisation (BSO) algorithm. BSO is developed based on the behavior of bacteria when it searches for the nutrients. The search space is divided into three regions namely, risk, null and rich regions. Differential Evolution algorithm (DE) is embedded within BSO to move the solutions toward the best solution. The performance of our approach is tested over eleven benchmark datasets (representing one large, five medium and five small problems). Experimental results show that our approach is able to generate competitive results when compared with previous available approaches. Possible extensions upon this simple approach are also discussed.

Keywords: Bacteria swarm optimisation, Differential Evolution, Timetabling.

1 Introduction

The course timetabling problem deals with the assignment of a set of courses to specific timeslots and rooms within a working week subject to a variety of hard and soft constraints. In this paper, a Bacteria Swarm Optimisation (BSO) algorithm for university course timetabling is presented. BSO was proposed by Passino [17]. It is inspired by the chemotactic behavior of the E. Coli. Chemotaxis is the phenomenon in which bacteria direct their movements according to certain chemicals substances in their environment. Bacteria move toward places with large concentrations of nutrients and they move away from places with low concentrations of nutrients. The bacterial system consists of four principal mechanisms, namely chemotaxis, elimination, reproduction and swarming [17]. This work attempts to apply the behaviour of bacteria swarm to solve university course timetabling problems.
Various methods have been investigated to solve university course timetabling problems. Carter and Laporte [10] divided these methods into four categories such as sequential, metaheuristics, constraint-based and cluster methods. A few years later Petrovic and Burke [18] included some other types such as case-based reasoning techniques, multi-criteria approaches and hyper-heuristic methods. Graph colouring heuristics were the earliest

Khalid Shaker
Department of Information Technology,
College of Information Technology, Ahlia University, Kingdom of Bahrain
E-mail: khalidalhity@gmail.com

Salwani Abdullah • Arwa Alqudsi
Center for Artificial Intelligence Technology,
FTSM, Universiti Kebangsaan Malaysia,
43600 Bangi, Selangor, Malaysia
E-mail: salwani@ukm.edu.my,
arwa.alqudsi81@gmail.com

approaches in solving timetabling problems. Currently, metaheuristic methods have shown a success in solving this problem. For example, population-based methods have been applied to course timetabling problems i.e. genetic algorithm [12]; ant algorithm [22]; artificial immune system [14] and harmony search [7]. Single solution-based methods that have employed on the same problem are such as dual simulated annealing [4]; variable neighbourhood search [2]; graph-based hyper heuristic [9]; non-linear great deluge [11] and extended great deluge [16].The hybridisation method with an aim to take the best feature from one approach and to incorporate it with another good (or better) feature from another (or more) approach(es) have also shown some success. For example the hybridisation between genetic and sequential local search [5]; iterated local search with mutation operator [3] and electromagnetic-like mechanism with great deluge [25]. Overviews of the previous approaches for course timetabling problems are available in [8, 13, 15, 20, and 21].

2 Problem Description

The Enrolment-based Course Timetabling Problem considered in this work was initially defined by the Metaheuristics Network[1]. This problem was discussed as an assignment of lecture events to timeslots and rooms according to a variety of hard and soft constraints. The problem description that is employed in this paper is adapted from the description presented in [22]. This problem includes four hard constraints and three soft constraints as follows:
Hard constraint:

- *Event conflict* i.e. no student can be assigned to more than one course at the same time (coded as H_1).
- *Room features* i.e. the room should satisfy the features required by the event (coded as H_2)
- *Room capacity* i.e. the number of students attending the event should be less than or equal to the capacity of the room (coded as H_3).
- *Room occupancy* i.e. no more than one event is allowed at a timeslot in each room (coded as H_4).

Soft constraints:

- *Event in the last timeslot* i.e. a student shall not have to sit a course that is scheduled in the last timeslot of the day (coded as S_1)
- *Two consecutive events* i.e. a student shall not have more than 2 consecutive events (coded as S_2).
- *One event a day* i.e. a student shall not have to sit a single course on a day (coded as S_3).

Hard constraints act an inviolable requirement. A timetable which meets the hard constraints is recognized as a *feasible* solution.

The problem has:

- A set of N courses, $e = \{e_1,\ldots,e_N\}$
- 45 timeslots
- A set of R rooms
- A set of F room features
- A set of M students.

The main objective is to minimise the violation of the soft constraints in a *feasible* solution that later represents the quality of the obtained solution. A solution consists of an ordered list of length $|N|$ where the position corresponds to the events i.e. position i corresponds to event e_i for $i = 1,\ldots,|N|$. The values for each position are a number between 0 to 44 corresponding to

[1] http://www.metaheuristics.net

the timeslot index and 0 to $|R\text{-}1|$ corresponding to the room index. For example, a timeslot vector is given as (0,17,30,…,10) and a room vector is given as (4,3,0,…,3) means that event e_1 is scheduled in timeslot 0 at room 4. Event e_2 is scheduled in timeslot 17 at room 3 and finally event $e_{|N|}$ is scheduled in timeslot 10 at room 3.

The objective function for the problem is defined in the formula below.

$$min \sum_{i=1}^{v} S_1 + S_2 + S_3 \quad (1)$$

where S_1, S_2 and S_3 represent the relevant soft constraints.

3 Proposed method: Bacteria Swarm Optimisation (BSO)

The proposed algorithm consists of two phases i.e. build a feasible initial population using a constructive heuristic; and an improvement algorithm with an aim to optimise the violation of the soft constraints while maintaining the feasibility of the solutions.

3.1 Constructive Heuristic

The initial population is produced using a constructive heuristic called a least saturation degree which starts from an empty timetable [16]. This feasible solution is obtained by adding or removing appropriate events (courses) from the schedule based on room availability (we attempt to schedule those courses with the least room availabilities earlier on in the process), without taking into account any of the soft constraints violations, until the hard constraints are met. If a feasible solution is found, the algorithm stops. Otherwise, the neighbourhood moves

infeasible to feasible solution. Nbs1 is applied for a certain number of iterations. If a feasible solution is met, then the algorithm stops. Otherwise the algorithm continues by applying a Nbs2 neighbourhood structure for a certain number of iterations. In this work, across all instances tested, the schedules are made feasible before starting the improvement algorithm.

3.2 Improvement algorithm using a Bacteria Swarm Optimisation algorithm

In this work we divide the search space into three regions ("risk" region, "null" region and "*rich*" region) based on the cardinality-based method that was proposed by Zäpfel et al. (2010).

$$\mu = g_{min} + \alpha\,(g_{max} + g_{min}) \quad \textbf{(2)}$$

From equation (2), we count how many solutions will be included in each region, where $\alpha \in [0, 1]$. For example, suppose we have 50 initial solutions in the population that are sorted in ascending order based on the quality of solutions. Assume that the lowest penalty cost among these solutions is 100, the highest is 500, and α=0.5. Then we get the interval μ=100+0.5(500-100) =300. That means the boundary of first region (*rich* region) will be in between 100 and 300. For the rest of solutions assume that the lowest penalty cost among these solutions is 310, of course, the highest is 500. Then we get the interval μ=310+0.5(500-310) =405. That means the boundary of second region (*null* region) will be in between 310 and 405 and the remained of solutions will be in the last region (*risk* region), the lowest penalty cost is 410 for example and the highest is 500. The solutions in the *risk* region and *null* region have to move toward the *rich* region. Also the solutions in the *rich* region have to move toward the global solution (the best solution in the *rich* region). At the improvement phase, after initialized all parameters, the algorithm identifies four mechanisms.

3.2.1 Chemotaxis

This process simulates the movement of bacteria through swimming via flagella. In this work we use DE (see subsection 3.3) to move the solutions toward the optimal solution.

During improvement, within this mechanism, the search starts with a randomly selected two parent solutions (p_1, p_2) from the population P based on D value which is set to 10. The D value represents the distance between two solutions to select the parents; which is calculated from the difference between the penalty costs of two solutions in the population. The purpose of D value is to achieve the exploitative of solutions as the algorithm selects only the parent timetables which are close to each other. The *Chemotaxis* mechanism has number procedures:

A. Mutation Operator: the mutation operator is carried out on the selected parents (p_1, p_2). Randomly selected neighbourhood structures (discussed in Section 7.4.2) will be applied on p_1 and p_2 to generate a new parent solutions denoted as p_1* and p_2*.

B. Crossover operator: by taking the two parent timetables (p_1*, p_2*) and exchanging two timeslots selected randomly between p_1*and p_2*, new two off-springs will be generated called $child_1$ and $child_2$. This will be disused in details in Section 7.4.2. .

C. Evaluation and Selection Operator: the evaluation and selection operators are carried out to evaluate and select the best new offspring. The quality of new offspring solutions ($f(child_1)$, $f(child_2)$) is calculated. The best solution among $child_1$ and $child_2$ (called *child**) will be compared with the best solution, *Solbest*. If there is an improvement, the counter of number of consecutive non-improving solution ($life\_\ p_i$) will be decreased by one and *child** and its parents (*p1, p2*) will be replaced with three worse solutions in P. Otherwise, $life\_\ p_i$ will be increased by one. Note that $life\_\ p_i$ is between 0 and 3 ($0< life\_\ p_i <3$). Note the best solution so far will be kept in the next generation.

3.2.2 Elimination

The least healthy bacteria eventually die, while each of the healthier bacteria (those yielding lower value of the objective function in this work) is split into two bacteria, which are then placed in the same region. This keeps the swarm size constant. In our work, if $life\_\ parent_i \geq 3$, that mean the parent p_i unable to generate a better *child* after 3 generations and should be eliminated from the population P. Then the population size needs to be fixed, a new solution need to replace the eliminated solution. This will be achieved next step (Reproduction).

3.2.3 Reproduction

The best solution in the same region that parent p_i was eliminated from (in the previous process) will be selected to apply a neighbourhood structure (randomly selected) on it to generate a new solution. This keeps the swarm size is fixed.

3.2.4 Swarming

An adaptive process could be occurred by rearranging the search space regions (based on equation 2 in subsection 3.2), thus, a different size of search regains based on the quality of population solutions. This could contribute a small diversification of solutions to the search. Where, improved solutions may move to another region, which diversify the collection of solutions once were in the region. As well as the range of regions will become different. The algorithm stops when the maximum number of iterations is reached or the penalty cost is zero. The pseudo code for the algorithm implemented in this work is given in Fig.1.

Step1: Initialisation:
Set S: total number of bacteria in the population P;
Set NumOfgenBSO: number of generations;
Set Solbest: the best solution in initial population;
Set P_i: the solution in P;
Set life_ p_i: number of consecutive non-improving solution;
Distribute the search space to three regions;

Step2: Procedures:
Repeat
A: Chemotaxis: *apply DE algorithm:*
***Selection**: Randomly select 2 parents from P, (p_1, p_2) based on D;*
***Mutation**: select neighbourhood structure N_i randomly and apply it on p_1 and p_2 to generate p_1*, p_2*;*
***Crossover**: generate $child_1$ and $child_2$ via exchanging the time slots (selected randomly) between p_1* and p_2* ;*
***Evaluation & Selection**: Choose the best between $child_1$ and $child_2$, called child**
***if** (f(child*) < f(Solbest))*
life_ p_i --;
replace the child and its parents (p_1,p_2) with three worse solutions in P;*
else
life_ p_i ++;
B: Elimination: *if life_ $p_i \geq 3$*
eliminate p_i (the parent which unable to generate better child after 3 generations will be eliminated from P);
C: Reproduction: *select the best solution in the same region of the eliminated p_i, and apply a randomly selected neighbourhood structure togenerate a new solution to keep the cardinality of the population size fixed;*
D: Swarming: rearrange the search space to three regions

Until (termination criterion is met);

Fig.1 The pseudo code of Bacteria swarm optimisation algorithm

3.3 Differential Evolution Algorithm (DE)

DE is a basic algorithm of the population as genetic algorithms using similar perators, crossover, mutation and selection. The main difference in obtaining better solutions is that genetic algorithms rely on crossover while DE is based on the operation of mutation. The main operation is based on differences of pairs of random solutions in the population. The algorithm uses mutation operation as a search mechanism and selection operation to direct the search toward the potential regions in the search space.

This population then is improved by applying mutation, crossover and selection operators. The main steps of the differential evolution algorithm are given in Fig. 2.

Initialization
Evaluation
do while *(termination criteria are met)*
Mutation
Crossover
Evaluation
Selection
do

Fig. 2 Differential Evolution Algorithm

3.3.1 Chromosome Representation

A simple chromosome representation can improve the crossover and mutation operations (Pongcharoen et al. 2008). Fig. 3 shows an example of the chromosome representation that is composed of a string of genes. Each chromosome represents a feasible timetable, where the gene represents timeslot t_i, room r_i, and course c_i. For example, courses c_1, c_5, c_7, c_{12} are scheduled at timeslot t_1 in the rooms r_1, r_2, r_3 and r_4, respectively.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
r_1	c_1	c_{10}	.	.	.	.	.	.	c_3
r_2	c_5	c_{11}	c_8	.	.	.	.	.	c_6
r_3	c_7	.	c_{13}	.	.	.	.	.	c_9
r_4	c_{12}	.	c_{16}	.	.	.	.	.	.

Fig. 3 Representation of chromosome for course timetabling problems

3.3.2 Mutation and Crossover Operations

The mutation is considered as a main operation for DE algorithm , as it relies on mutation operation (Storn and Price 1997). Random selections of neighborhood structures listed below are used in a mutation process which is based on a mutation rate.

Nbs_1: Choose a single course at random and move to a feasible timeslot that can generate the lowest penalty cost.

Nbs_2: Select two courses at random from the same room (the room is randomly selected) and swap timeslots.

The crossover operation is illustrated as in Fig. 4. We use the same crossover mechanism that is proposed by (Abdullah et al. 2008).

Parent1

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
r_1	c_{18}		.	c_4	c_{10}	c_{17}	c_2	c_{19}	c_{13}
r_2	c_{14}	c_6	c_5	c_{12}	c_1	.	.	.	c_{15}
r_3	c_{20}	c_7	c_3	.	.	c_9	c_{16}	.	.
r_4	c_{21}	.	c_8	.	c_{11}	.	.	c_{22}	.

Parent2

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
r_1	c_{19}	c_4	.	.	c_{20}	c_{17}	c_2	c_8	c_5
r_2	c_{13}	c_{16}	.	c_{21}	c_{14}	.	.	.	c_{16}
r_3	.	.	c_3	c_6	.	c_{12}	.	c_7	.
r_4	c_9	c_{10}	.	.	c_1	c_{18}	.	c_{22}	c_{11}

Offspring1

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
r_1	c_{18}	c_8	.	c_4	c_{10}	c_{17}	c_2	c_{19}	c_{13}
r_2	c_{14}	c_6	c_5	c_{12}	c_1	.	.	.	c_{15}
r_3	c_{20}	c_7	c_3	.	.	c_9	c_{16}	.	.
r_4	c_{21}	c_{22}	~~c_8~~	.	c_{11}	.	.	~~c_{22}~~	.

Offspring2

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
r_1	c_{19}	c_4	.	.	c_{20}	c_{17}	c_2	c_8	c_5
r_2	c_{13}	c_{16}	.	c_{21}	c_{14}	.	.	c_6	c_{16}
r_3	.	.	c_3	~~c_6~~	.	c_{12}	.	c_7	.
r_4	c_9	c_{10}	.	.	c_1	c_{18}	.	c_{22}	c_{11}

Fig. 4 Crossover operation

The crossover operator exchanges the shaded timeslots between p_1 and p_2 to form the offspring. The shaded timeslots, i.e. t_3 and t_7 are randomly selected. In this case, the feasibility of the solution is likely will be violated. In order to maintain the feasibility of the offspring, two conditions are considered:

1. Course can be moved only if the corresponding timeslots is empty. For example, c_6 can be moved from timeslot t_2 in Parent1 to timeslot t_8 in Parent2.
2. No conflicts occur between moved course and scheduled courses. For example, in Offspring1, there should be no conflict between moved course c_8 from t_8 (Parent2) and scheduled courses c_6, c_7 in t_2 (Parent1).

4 Experimental Results

The algorithm was implemented on a Pentium 4 Intel Core i3 1.8 GHz PC Machine using Matlab on a Windows XP Operating System. We have evaluated our results on the instances taken from Socha *et al*. [17] and which are available at http://iridia.ulb.ac.be/~msampels/tt.data/. Table 1 shows the Parameters setting for the proposed algorithm after some preliminary experiments and almost similar with the papers in the literature (e.g. Yang and Jat 2011). We ran the experiments for 200000 iterations with 11 test-runs to obtain an average value

Table 1 Parameters setting of BSO algorithm

Parameters	Value
Number of Generations	200000
Population Size	50
Crossover rate	0.8
Mutation rate	0.5

We have evaluated our results on the instances taken from Socha et al. (2002). The best results out of 11 runs obtained are presented in Table 2. The table shows the comparison of the approach in this work against other available approaches reported in the literature.

Table 2 Comparison Results

Dataset	Our method	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
small1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	3	**0**	**0**	**0**	1	1
small2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	4	**0**	**0**	**0**	3	2
small3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	6	**0**	**0**	**0**	1	**0**
small4	**0**	**0**	**0**	**0**	**0**	**0**	**0**	6	**0**	**0**	**0**	1	1
small5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
medium1	87	175	242	84	**73**	99	80	140	96	139	124	195	146
medium2	78	197	161	82	79	**73**	105	130	96	92	117	184	173
medium3	146	216	265	123	132	135	139	189	135	**122**	190	248	267
medium4	**55**	149	181	62	69	112	88	112	79	98	132	164	169
medium5	**60**	190	151	75	61	87	88	141	87	116	73	219	303
Large	589	912	-	590	462	498	730	876	683	615	**424**	851	1166

Where:

M1 Genetic algorithm and local search by Abdullah and Turabieh (2008).
M2 Randomised iterative improvement algorithm by Abdullah et al. (2007).
M3 An Elitist-Ant System by Jaradat and Ayob (2010)
M4 Hybridized artificial bee colony with hill climbing by Asaju et al. (2014).
M5 Hybrid harmony search metaheuristic algorithm by Al-Betar et al. (2014).
M6 Extended great deluge by McMullan (2007).
M7 Non-linear great deluge by Landa-Silva and Obit (2008).
M8 Electromagnetic-like mechanism with great deluge by Turabieh et al. (2009).
M9 Genetic Algorithms and Local Search by Yang and Naseem Jat: et al. (2011).
M10 Harmony search by Al-Betar et al. (2010).
M11 The max-min ant algorithm by Socha et al. (2002).
M12 The tabu hyper heuristics by Burke et al. (2003).

Note that the best results are presented in bold. The term "-" in Table 2 indicates a percentage of runs that failed to obtain feasible solution. It can be seen that in general, our approach is better than other approaches reported in the table and is able to generate two best solutions i.e. *medium*4 and *medium*5 datasets. It can be clearly seen that the BSO approach performs well for the *small* datasets where the algorithm is able to obtain the optimal solutions with zero penalty, because the *small* datasets might have more feasible solution points in the

search space compared to the *medium* and *large* datasets. We believe that the way of the algorithm explore the search space helps significantly the algorithm to avoid the stagnation state which occurs when there is no update to the population during the improvement process. In addition, the novel strategy used to replace the new child and it is parents (not only the child) with the worse three solutions in the population *P* gives a chance to deal only with flexible solutions which could generate better solutions.

Fig. 5 (a), (b), (c) and (d) illustrate the effectiveness of the algorithm when it explores the search space on *small*1, *medium*3, *medium*5 and *large* datasets, respectively. The x-axis represents the number of iterations, while the y-axis represents the penalty cost. The figures show that the approach is able to produce fast convergence. This due to that the algorithm performs the mutation operator before the crossover operator. That is, the mutation directs the search toward the prospective regions in the search space. The BSO significantly reduces the search space by considering feasible solution spaces, as well as by determining the boundaries of the nutrition regions. Where, these regions are restricted to boundaries based on the maximum and minimum penalty cost of solutions which are close to each other in term of distances.

In case of *small* datasets, the way the algorithm explores the search space clearly indicates that further improvement is achieved. The algorithm is also effective for the *medium* and *large* datasets, in the graphs clearly indicating that a further reduction of almost 70% of initial solutions in the cost evaluation can be achieved.

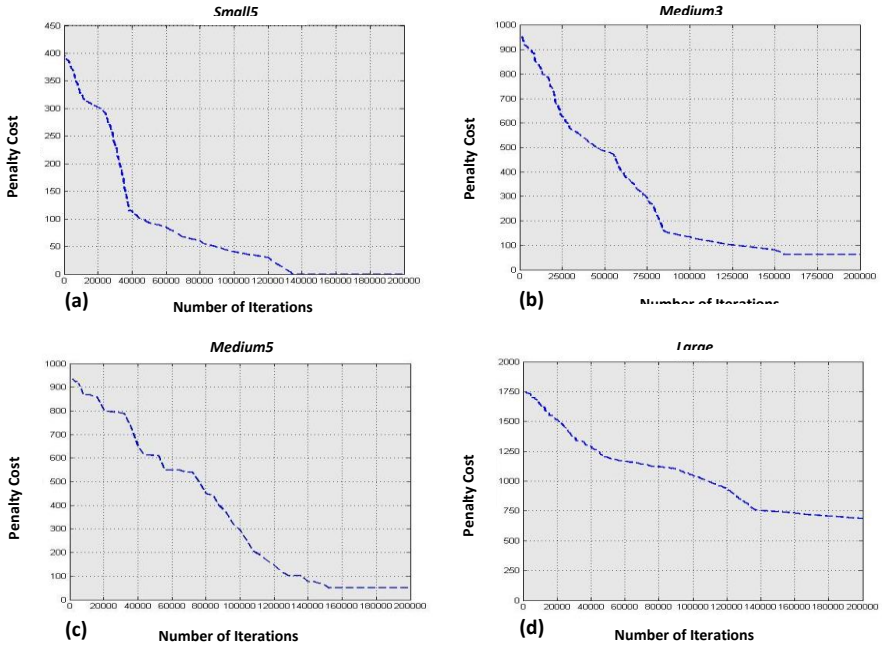


Fig. 5 Convergences results of (a) s*mall*5, (b) *medium*3, (c) *medium*5, and (d) *large* datasets

The quality of the results illustrated in Fig. 6 (a) and (b) which show the box plots of the penalty costs when solving *small*, *medium* and *large* instances, respectively.

Fig. 6 (a) and (b) show that the results of small datasets are high dispersed than medium datasets, but it can be clearly seen that in *small*1, *small*2 and *small*3 the median is closer to the *best* than the *worst*. However, the figures show a deceived illustration of the algorithm's runs over the small datasets. Where, it can be seen that the dispersion of the datasets lies within a short interval (e.g. between 0 and 3 for small2). Thus, we need to refer to the standard deviation values of the datasets in order evaluate correctly whether if there is a high dispersion of the small datasets or not. This will be conducted in the following paragraph. In the other hand, the results for the *medium* datasets are less dispersed compared to l*arge* and

small datasets as the *median* is closer to the *best* than the *worst* in all medium datasets. This indicates that the algorithm deals only with the much closed solutions, and this clarifies that our algorithm is effective, consistent, and able to generate competitive results when compared to other state-of-the-art techniques. For high dispersed results obtained for *large* datasets, we believe that the size of the search space is different from one problem to another, thus the dispersion of solution points are significantly different from one to another.

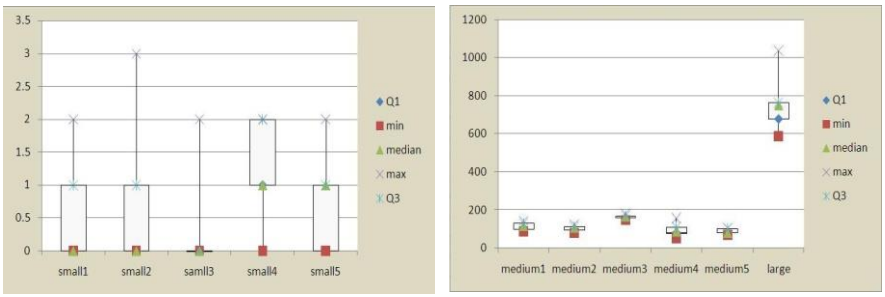


Fig. 6 (a) and (b) Box plots of the penalty costs for small, medium and large datasets.

5 Conclusion

This work in this paper presents an effective Bactria Swarm Optimisation Algorithm (BSO) for the university course timetabling problem, showing that evolutionary computation can deal successfully with the problem. The use of the effective Differential Evolution Algorithm (DE) is an important element of its high performance.

There are four advantages from using DE within BSO; first, move the solutions toward the best solutions, secondly, finding a true global minimum regardless the parameter values, thirdly, fast convergence, and lastly, a few control parameters are used.

The performance of the BSO has been compared to that of some other well-known heuristic algorithms. From the simulation results, it is observed that the convergence speed of the BSO is significantly better than other approaches in the literature. Therefore, the BSO seems to be a promising approach for course timetabling problems. Our future work will try to apply BSO on curriculum-based course timetabling problems ITC2007.

References

1. Asaju La'aro Bolaji, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, Mohammed A. Awadallah: University course timetabling using hybridized artificial bee colony with hill climbing optimizer. J. Comput. Science 5(5): 809-818 (2014)
2. Abdullah, S., Burke, E.K., McCollum, B.: A hybrid evolutionary approach to the university course timetabling problem. IEEE Congress on Evolutionary Computation, ISBN: 1-4244-1340-0, pp 1764-1768 (2007)
3. Abdullah, S., Burke, E.K., McCollum, B.: An investigation of variable neighbourhood search for university course timetabling. The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), pp. 413–427 (2005)
4. Abdullah, S., Burke, E.K., McCollum, B.: Using a randomised iterative improvement algorithm with composite neighbourhood structures for university course timetabling. In Metaheuristics: Progress in complex systems optimization (Operations Research / Computer Science Interfaces Series), Chapter 8. Published by Springer, ISBN:978-0-387-71919-1 (2007)

5. Abdullah, S., Shaker, K., McCollum, B., McMullan, P.: Dual Sequence Simulated Annealing with Round-Robin Approach for University Course Timetabling. EVOCOP 2010, LNCS 6022, Springer-Berlin /Heidelberg, pp 1–10 (2010)
6. Abdullah, S., Turabieh, H.: Generating university course timetable using genetic algorithms and local search. The Third 2008 International Conference on Convergence and Hybrid Information Technology ICCIT, vol. I, pp 254-260 (2008)
7. Khalid Shaker, Salwani Abdullah, Arwa Alqudsi, Hamid Jalab: Hybridizing Meta-heuristics Approaches for Solving University Course Timetabling Problems. RSKT 2013: 374-384, (2013)
8. Khalid Shaker, Salwani Abdullah, Arwa Hatem: A Differential Evolution Algorithm for the University course timetabling problem. DMO 2012: 99-102, (2012)
9. Khalid Shaker & Salwani Abdullah: Controlling Multi Algorithms Using Round Robin for University Course Timetabling Problem. FGIT-DTA/BSBT 2010: 47-55, (2010)
10. M.A. Al-Betar, A.T. Khader, M. Zaman: University course timetabling using a hybrid harmony search metaheuristic algorithm, IEEE Trans. Syst. Man Cyber- net. C, Appl.Rev. doi.org/10.1109/TSMCC.2011.2174356, (2014)
11. Ghaith M. Jaradat, Masri Ayob: An Elitist-Ant System for Solving the Post-Enrolment Course Timetabling Problem. FGIT-DTA/BSBT 2010: 167-176 (2010)
12. Shengxiang Yang, Sadaf Naseem Jat: Genetic Algorithms with Guided and Local Search Strategies for University Course Timetabling. IEEE Transactions on Systems, Man, and Cybernetics, Part C 41(1): 93-106 (2011)
13. Al-Betar, M., Khader, A., Yi Liao, I.: A Harmony Search with Multi-pitch Adjusting Rate for the University Course Timetabling. In: Z.W. Geem: Recent Advances in Harmony Search Algorithm, SCI 270, pp. 147–161. Springer, Heidelberg (2010)
14. Burke, E., Eckersley, A., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. Technical Report NOTTCS-TR-2006-2, University of Nottingham, School of CSiT (2006)
15. Burke, E.K., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for timetabling problems. European Journal of Operational Research 176, pp 177-192 (2007)
16. Landa-Silva, D., Obit, J.H.: Great deluge with non-linear decay rate for solving course timetabling problem. The fourth international IEEE conference on Intelligent Systems. Varna, Bulgaria, pp. 8.11–8.18 (2008)
17. Lewis, R., Paechter, B.: New crossover operators for timetabling with evolutionary algorithms. Proceedings of the 5th International Conference on Recent Advances in Soft Computing (ed. Lotfi), UK, December 16th-18th, pp 189-194 (2004)
18. Lewis, R.: A survey of metaheuristic-based techniques for University Timetabling problems. OR Spectrum 30, 167–190 (2008)
19. Malim, M.R., Khader, A.T., Mustafa, A.: Artificial Immune Algorithms for University Timetabling. In: Burke, E.K., Rudova, H. (eds.) The 6th International Conference on Practice and Theory of Automated Timetabling, Brno, Czech Republic, pp. 234–245 (2006)
20. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A., Di Gaspero, L., Qu, R., Burke, E.: Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition, Accepted for publication to INFORMS Journal on Computing. doi 10.1287/ijoc.1090.0320 (2009)
21. McMullan, P.: An extended implementation of the great deluge algorithm for course timetabling, Computational Science – ICCS, Part I, LNCS 4487, Springer-Verlag Berlin Heidelberg, pp 538–545 (2007)
22. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., and Lee, S.Y.: A Survey of Search Methodologies and Automated System Development for Examination Timetabling. Journal of Scheduling, 12(1): pp 55-89 (2009)
23. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intelli. Rev. 13, 87–127 (1999)

24. Socha, K., Knowles, J., Samples, M.: A max-min ant system for the university course timetabling problem. The Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002), LNCS 2463, Springer-Verlag, pp 1-13 (2002)
25. Thompson, J., Dowsland, K.: A robust simulated annealing based examination timetabling system. Computers & Operations Research, 25:637–648 (1998)
26. Turabieh, H., Abdullah, S., McCollum, B.: Electromagnetism-like Mechanism with Force Decay Rate Great Deluge for the Course Timetabling Problem. In: RSKT 2009. LNCS, vol. 5589, pp. 497–504. Springer, Heidelberg (2009)

MISTA 2015

A lexicographic goal programming approach for staff assignment with acceptance levels

Tom Rihm · Philipp Baumann

Abstract We present a real-world staff-assignment problem that was reported to us by a provider of an online workforce scheduling software. The problem consists of assigning employees to work shifts subject to a large variety of requirements related to work laws, work shift compatibility, workload balancing, and personal preferences of employees. A target value is defined for each requirement, and deviations from the target values are associated with acceptance levels. The objective is to minimize the total number of deviations in lexicographical order of the acceptance levels. This objective cannot be represented in existing goal programming approaches straightforwardly. We develop a lexicographic goal programming formulation that models this objective efficiently, and we introduce aggregation techniques to reduce the number of constraints. To evaluate the performance of the proposed formulation, we derive a test set of 27 instances from real-world data. The approach is able to devise optimal or near-optimal solutions for small- and medium-sized instances in short running times. The solutions obtained by our approach are used by the software provider as benchmark results to evaluate and improve its software.

1 Introduction

A large number of companies and organizations in the service industries have to assign employees to work shifts on a regular basis. Typical examples are hospitals, hotels, call centers, airports and police departments. As people are the most critical resource for such service providers, a careful and proper planning can lead to significant improvements in productivity. In general, a large number of potentially conflicting requirements related to work regulations, work shift compatibility, workload balance, and personal preferences of employees must be considered. A major challenge lies in balancing the requirements of different stakeholders such as management, employees and customers.

Tom Rihm
University of Bern
E-mail: tom.rihm@pqm.unibe.ch

Philipp Baumann
University of California, Berkeley
E-mail: philipp.baumann@berkeley.edu

In this paper, we present a particular type of staff assignment problem that was reported to us by a provider of an online workforce scheduling software. This provider has developed a framework that helps decision makers to specify trade-offs between different requirements. The decision maker is asked to specify a target value for each requirement and assign acceptance levels to deviations from the target value. In general, decision makers assign lower acceptance levels to larger deviations. For the sake of simplicity and transparency, the mapping of deviations to acceptance levels is piecewise constant, and less-accepted deviations are considered to be strictly more important than more-accepted deviations. According to the software provider, this framework is well received by their clients. The resulting staff assignment problem consists of assigning employees to work shifts such that the number of deviations from target values is minimized, with a reduction in the number of less-accepted deviations being always preferred to any reduction in the number of more-accepted deviations. A typical problem instance has a planning horizon of four weeks and comprises between 10 and 15 different types of requirements.

The literature on staff-assignment problems with multiple and potentially conflicting requirements concentrates on goal programming approaches (cf. [15]). In goal programming, each requirement is assigned a target value, and deviations from the target values are minimized (cf. [7,6]). The most widely-used variants of goal programming in practical applications are lexicographic, Chebyshev and weighted goal programming. Lexicographic goal programming minimizes deviations sequentially and is therefore used for applications with a predefined ranking of requirements. Chebyshev goal programming minimizes the maximum deviations across all requirements and is therefore used when a balanced achievement of the requirements is desired. Only weighted goal programming is used to model trade-offs between requirements. In weighted goal programming, the deviations are normalized, and a weighted sum of deviations is minimized. Various extensions of weighted goal programming have been proposed which enable decision makers to specify trade-offs between requirements more accurately (cf., e.g., [10,13]). These extensions include penalty functions that assign larger weights to larger deviations. However, weighted goal programming is not applicable to the problem considered here because the range of weights required to ensure that less-accepted deviations are always minimized before more-accepted deviations grows rapidly with the number of different acceptance levels and may become large enough to cause numerical problems for solvers.

In this paper, we develop a lexicographic goal programming formulation for the staff assignment problem under study. The proposed formulation is able to efficiently account for trade-offs between requirements. The key idea is to decompose each requirement into a set of sub-requirements, each of which is associated with an acceptance level and has its own target value. The formulation contains one constraint and one binary deviational variable for each sub-requirement. The binary deviational variables indicates whether or not a sub-requirement is violated. To determine the assignment, we iteratively minimize the total number of sub-requirement violations that map to the same acceptance level, starting with the lowest acceptance level in the first iteration. We increase the computational efficiency of the formulation by introducing aggregation techniques that reduce the number of constraints. The performance of the proposed approach is evaluated based on a test set of 27 problem instances. Our approach devises optimal or near-optimal solutions for small- and medium-sized instances in short CPU times. The aggregation techniques are particularity effective for instances with a large number of sub-requirements.

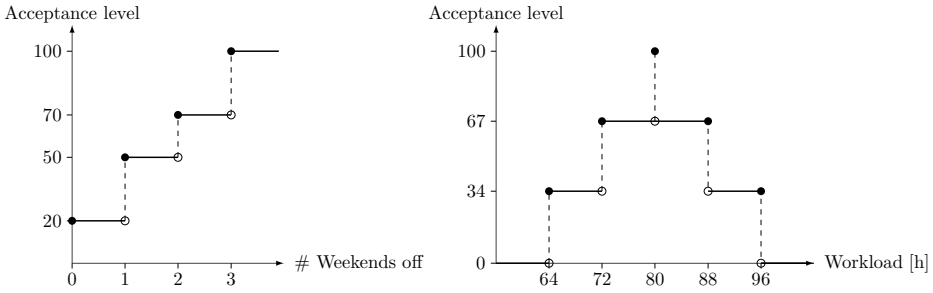


Fig. 1 Two examples of functions that map deviations from target values to acceptance levels.

The remainder of the paper is structured as follows. In Section 2, we describe the planning problem in more detail. In Section 3, we review the related literature. In Section 4, we present our solution approach. In Section 5, we illustrate the proposed solution approach by means of an illustrative example. In Section 6, we report on the computational results. Finally, Section 7 concludes the paper.

2 Problem description

The staff assignment problem considered in this paper can be described as follows. Given are a set of employees, a set of work shifts with predefined start times and durations, and a set of requirements. Some requirements are hard requirements, i.e., they must be satisfied by all feasible assignments. They are related to general work laws, contract specifications and the availability and qualifications of employees. Other requirements are soft requirements and are concerned with personal preferences of employer and employees. Examples of soft requirements are the achievement of predefined workloads for employees, the consideration of employee's requests for days off and the equal assignment of night shifts. These soft requirements are goals to be reached. Hence, for each soft requirement, a target value is specified. In addition, a piecewise-constant function is given that maps deviations from the target value to integer acceptance levels in the interval [0,100]. An acceptance level of zero indicates that the corresponding deviation is unacceptable and leads to an infeasible assignment. An acceptance level of 100 indicates that the target value or an even better value was achieved.

Figure 1 shows the mapping functions for two different requirements. The function on the left corresponds to the requirement that an employee has at least three weekends off during the planning horizon. Hence, if the employee has three or more weekends off, an acceptance level of 100 is achieved. Having only two or fewer weekends off results in an acceptance level below 100. The function on the right corresponds to the requirement of an employee to work exactly 80 hours during the planning horizon. Working more or less than 80 hours results in a lower acceptance level. A workload of less than 64 hours or more than 96 hours is infeasible, which is indicated by an acceptance level of zero. The assignment problem consists of finding an assignment of employees to work shifts such that all hard requirements are satisfied and the number of deviations from target values of soft requirements is minimized. Thereby a reduction in the number of less-accepted deviations is always preferred to any number of reductions in more-accepted deviations.

3 Literature review

In this section, we provide a brief overview of goal programming variants related to the planning problem described in Section 2 and discuss their use in practical applications.

In goal programming, each requirement is associated with a target value and deviations from target values are captured by deviational variables (cf. [7,6]). A so-called achievement function minimizes the deviational variables according to the preferences of the decision maker. The most widely used variants of goal programming are lexicographic, Chebyshev and weighted goal programming.

Lexicographic goal programming requires a ranking of the requirements that reflects their importance. The unwanted deviations from the target values are minimized sequentially according to the given ranking. This variant has been used by decision makers who do not need to model trade-offs between requirements because they have a clear ranking of the requirements in mind (cf., e.g., [2]). Chebyshev goal programming minimizes the maximum unwanted deviation across all requirements. It has been used by decision makers who are interested in a balanced achievement of requirements (cf., e.g., [9]).

Weighted goal programming allows for direct trade-offs between requirements. A weight is defined for each deviational variable that quantifies its relative importance. The achievement function is the weighted sum of the deviational variables. In the traditional form of weighted goal programming it is assumed that the weights are constant and do not change at further distances from the target value (cf., e.g., [1]). As this assumption is too restrictive to fit the preferences of many decision makers, various extensions have been proposed.

Charnes and Collomb [5] introduced interval goal programming which allows decision makers to specify a target interval instead of a target value. Deviations from either end of the interval are penalized in the achievement function. Later [8] introduced penalty functions that penalize large deviations with a higher weight than small deviations. This idea was extended by [12] and also [10] who proposed more complex penalty functions including decreasing functions, functions with discontinuities and non-linear functions. Romero [13] consolidates U-Shaped Penalty Functions in an achievement function with a general structure. This achievement function also encompasses the basic variants of lexicographic and Chebyshev goal programming. The use of complex penalty functions requires the introduction of binary variables and additional constraints which increases computational cost. To address this drawback of interval goal programming models [3] and [4] proposed techniques to reduce the number of variables and constraints required to model specific penalty functions.

The extensions described above allow a more accurate modelling of the decision maker's preferences and have been applied to many real-world applications (cf. the reviews of [14] and [11]). However, in order to apply weighted goal programming to the problem considered here, the penalty functions need to assign weights in such a way that the weight of a less-accepted deviation is always greater than the sum of all weights of more-accepted deviations. For a small problem instance with 40 different acceptance levels and ten deviational variables per acceptance level, the largest weight has to be more than 10^{40} times bigger than the smallest weight. Such large numbers may slow down commercial solvers or even cause numerical problems. We therefore present in the next section a lexicographic goal programming approach that is able to consider trade-offs between requirements.

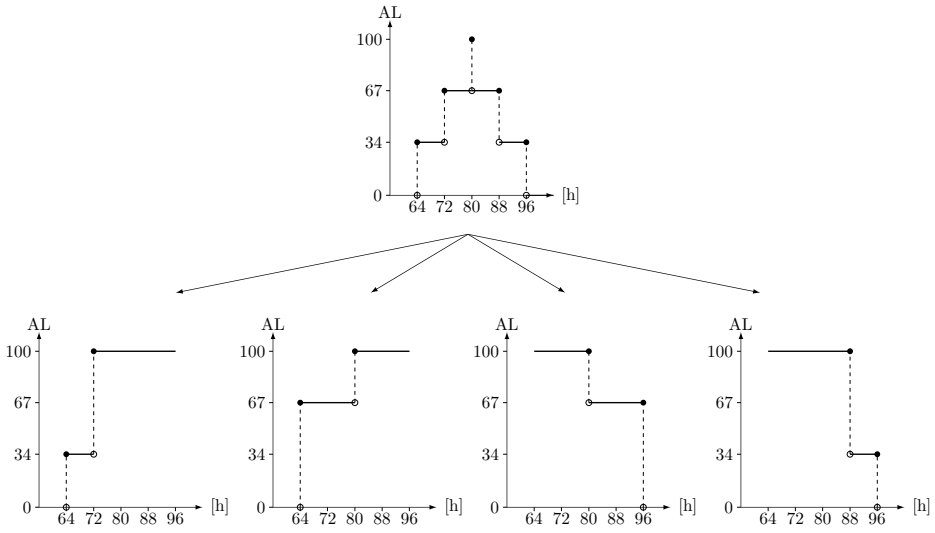


Fig. 2 Decomposition of a requirement into four sub-requirements.

4 Proposed solution approach

Our solution approach comprises two phases. In the first phase, we decompose each requirement into a set of sub-requirements. The number of sub-requirements is equal to the number of kinks in the mapping function of the original requirement. Each sub-requirement gets the target value and the acceptance level from the corresponding kink in the mapping function. Figure 2 shows the decomposition of a requirement that has four kinks in the corresponding mapping function.

In the second phase, we formulate and solve a lexicographic goal program (LGP). For each sub-requirement, we introduce a binary deviational variable that is equal to one when the sub-requirement is violated. The LGP is solved iteratively as a series of binary linear programs (BLPs). The first BLP minimizes the number of sub-requirement violations associated with the lowest acceptance level. The second BLP minimizes the number of sub-requirement violations associated with the second lowest acceptance level, etc. In each iteration, an additional constraint is added to ensure that the number of violations from the previous optimization will not be exceeded in subsequent iterations.

Note that the additional constraints do not fix assignments as the corresponding deviational variables can still change in subsequent iterations as long as the total number of violations does not exceed the prescribed upper bound. A distinctive feature of our approach is that all deviational variables are binary, which provides great flexibility for extensions. E.g., a balanced distribution of sub-requirement violations among employees could be incorporated easily.

5 Illustrative example

For the purpose of illustration, we apply the proposed solution approach to an illustrative example. We introduce the notation in Subsection 5.1, present the data of the example in Subsection 5.2, and formulate the lexicographic goal program in Subsection 5.3. Finally, we present aggregation techniques for reducing the number of constraints in Subsection 5.4.

5.1 Notation

We use the following notation.

Indices

a	Acceptance level	r	Sub-requirement
d	Combination of indices (domain)	s	Shift
i	Employee	t	Day
q	Requirement	w	Weekend

Sets

A	Acceptance levels
D_r	Domain of sub-requirement r
D_{ra}	Domain of sub-requirement r at acceptance level a
I	Employees
I_s	Employees compatible with shift s
R^{sub}	Sub-requirements
R^{sub}_q	Sub-requirements of requirement q
$R^{sub}_{q,i}$	Sub-requirements of requirement q relevant for employee i
S	Shifts
S_{it}	Compatible shifts of employee i starting on day t
S^k_{it}	Pair of shifts between which the rest period is less than k hours
T	Days
T_w	Days of weekend w
W	Weekends

Parameters

$g_{rd}/\widehat{g}_{rd}$	Factor of deviational variable z_{rd} for formulation (BF)/(AF)
l_s	Length of shift s
b_{rd}	Target value of sub-requirement r for index combination d
v_a	Allowed number of violations at acceptance level a

Variables

x_{is}	$= 1$, if employee i is assigned to shift s; $= 0$, else
y_{iw}	$= 1$, if employee i has weekend w off; $= 0$, else
z_{rd}	$= 1$, if LHS of constraint rd exceeds/falls below target value; $= 0$, else

Name	**Start**	**End**	**Qualified employees**	Week 1							Week 2						
				Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Shift A	8am	4pm	Ann, Dan, Eva	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}
Shift B	11am	7pm	Ann, Bob, Dan	B_1		B_3		B_5	B_6	B_7		B_9		B_{11}		B_{13}	B_{14}
Early Shift	4am	12pm	Dan, Eva, Gil	E_1	E_2	E_3	E_4	E_5			E_8	E_9	E_{10}	E_{11}	E_{12}		
Late Shift	3pm	11pm	Bob, Dan, Gil		L_2		L_4				L_8		L_{10}		L_{12}		
Admin Shift	9am	5pm	Ann, Bob			AS_3						AS_9		AS_{11}			

Fig. 3 Illustrative example: shifts to be assigned to employees

5.2 Data

Over a planning horizon of two weeks, five different types of shifts need to be assigned to employees. Figure 3 shows for each shift type the name, the start and end times, all employees with the required qualification and the days on which the shift should be performed. There are five employees (Ann, Bob, Dan, Eva, and Gil) who can be assigned to the shifts. The assignment of shifts must consider two hard requirements and should consider nine soft requirements. Table 1 provides a short description of all eleven requirements.

Table 2 shows the result of the decomposition phase. The soft requirements have been decomposed into twelve sub-requirements. The third column of Table 2 contains the domain of each sub-requirement. The acceptance levels and target values associated with sub-requirements are listed in the fourth and fifth column of Table 2. The last column specifies the maximum positive or negative undesired deviation from the target value that is still considered feasible.

The domain defines the set of employees, shifts or days for which the sub-requirement applies to. For example, according to requirements R_3 each shift should be assigned once. The domain of sub-requirement $r = R_3$–1 thus applies to all shifts and the target value for each shift is 1. The maximal allowed deviation in this case is $g_{rs} = -1$. Sub-requirement R_4–1 applies to employee Bob on days 1, 5, 11, and 12 and to employee Gil on days 12, 13, and 14. If Bob or Gil have to work on one of these days, a violation associated with acceptance level 30 occurs. The information given for sub-requirements R_9–1 and R_9–2 says the following. Ann and Dan have ideally two or more weekends off which is expressed by the target value of sub-requirement $r = R_9$–1. From $g_{rd} = -2$ follows that the hard lower bound on the number of weekends off is 0. Having only one weekend off satisfies sub-requirement R_9–2 but violates sub-requirement R_9–1. Such a violation is associated with an acceptance level of 80. Having no weekend off also violates sub-requirement R_9–2 which is associated with an acceptance level of 50.

5.3 Model formulation

The objective is to minimize the total number of sub-requirement violations associated with acceptance level a^*.

$$\text{Min} \quad \sum_{r \in R^{sub}} \sum_{d \in D_{ra^*}} z_{rd} \tag{1}$$

Requirement	Description	Type
R_1	At most one shift per employee and day	hard
R_2	Only assign an employee with the required qualification to a shift	hard
R_3	Each shift is assigned once (at most once)	soft
R_4	Employee's requests for days off should be considered	soft
R_5	Employees should not work on more than five consecutive days	soft
R_6	No isolated days off	soft
R_7	At least $k = 11$ hours layup between consecutive shifts	soft
R_8	Either zero or two shifts on weekends	soft
R_9	Lower bound on number of weekends off	soft
R_{10}	Workload should not exceed target	soft
R_{11}	Workload should not be below target	soft

Table 1 Requirements for illustrative example

Requirement q	Sub-requirement r	Domain d	Acceptance level a	Target value b_{rd}	Parameter g_{rd}
R_3	R_3–1	$s \in S$	1	1	-1
R_4	R_4–1	$i \in \{\text{Bob}\},\ t \in \{1, 5, 11, 12\}$	30	0	1
		$i \in \{\text{Gil}\},\ t \in \{12, 13, 14\}$	30	0	1
R_5	R_5–1	$i \in I,\ t \in T : t > 5$	60	5	1
R_6	R_6–1	$i \in I,\ t \in T : 1 < t < \lvert T \rvert$	60	1	1
R_7	R_7–1	$i \in I,\ t \in T :\ t > 1$	1	1	1
R_8	R_8–1	$i \in I,\ w \in W$	30	2	-1
R_9	R_9–1	$i \in \{\text{Ann,Dan}\}$	80	2	-2
		$i \in \{\text{Bob,Eva,Gil}\}$	60	1	-1
	R_9–2	$i \in \{\text{Ann,Dan}\}$	50	1	-1
R_{10}	R_{10}–1	$i \in \{\text{Ann,Bob,Dan}\}$	70	80	16
		$i \in \{\text{Eva,Gil}\}$	60	40	16
	R_{10}–2	$i \in \{\text{Ann,Bob,Dan}\}$	30	88	8
		$i \in \{\text{Eva,Gil}\}$	20	48	8
R_{11}	R_{11}–1	$i \in \{\text{Ann,Bob,Dan}\}$	70	80	-16
		$i \in \{\text{Eva,Gil}\}$	60	40	-16
	R_{11}–2	$i \in \{\text{Ann,Bob,Dan}\}$	30	72	-8
		$i \in \{\text{Eva,Gil}\}$	20	32	-8

Table 2 Sub-requirements for illustrative example

5.3.1 Hard requirements

Constraints (2) bound the number of sub-requirement violations at acceptance levels $a < a^*$ to ensure that the optimal values from previous iterations are maintained.

$$\sum_{r \in R^{sub}} \sum_{d \in D_{ra}} z_{rd} \leq v_a \qquad (a \in A : a < a^*) \tag{2}$$

Constraints (3) address requirement R_1, i.e., they ensure that each employee $i \in I$ is assigned to at most one shift each day $t \in T$.

$$\sum_{s \in S_{it}} x_{is} \leq 1 \qquad (i \in I,\ t \in T) \tag{3}$$

Requirement R_2 ensures that for every shift only employees with the required qualifications are assigned to. This is achieved by the construction of the sets I_s and S_{it}.

5.3.2 Soft requirements

Sub-requirement $r = R_3$–1 is considered by constraints (4), which prevent more than one employee being assigned to the same shift and penalize if no employee is assigned to one shift. The left-hand side counts the number of employees assigned to shift s. The target value b_{rs} for each shift s ist one and the maximum allowed deviation is $g_{rs} = -1$.

$$\sum_{i \in I_s} x_{is} = b_{rs} + g_{rs} z_{rs} \qquad (r = R_3\text{–}1,\ s \in D_r) \qquad (4)$$

Constraints (5) cover sub-requirement R_4–1, which aims at complying with employee's requests for days off. For each request (employee i on day t), the target value is zero.

$$\sum_{s \in S_{it}} x_{is} \leq b_{rit} + g_{rit} z_{rit} \qquad (r = R_4\text{–}1,\ (i,t) \in D_r) \qquad (5)$$

If the left-hand side is greater than or equal to one, the request of employee i on day t is rejected. As the target value is zero, deviational variable z_{rit} must be one. Sub-requirement $r = R_5$–1 tries to prevent that employee $i \in I$ works on more than $b_{rit} = 5$ consecutive days. As g_{rd} is positive, only exceedances of the target value of 5 are penalized.

$$\sum_{t'=t-b_{rit}}^{t} \sum_{s \in S_{it'}} x_{is} \leq b_{rit} + g_{rit} z_{rit} \qquad (r = R_5\text{–}1,\ (i,t) \in D_r) \qquad (6)$$

Each time, an employee has to work on a day even though he/she worked on the preceding $b_{rit} = 5$ days, the left-hand side exceeds the target value and a violation occurs. We do not consider the previous period. For this reason, sub-requirement $r = R_5$–1 cannot be violated on the days $t \leq b_{rit}$ and the domain set D_r is constructed such that $t > b_{rit}$. Sub-requirement R_6–1 considers that employees prefer two consecutive days off. We implement this sub-requirement with constraints (7) which penalize if a day off is preceded and succeeded by a work-day.

$$\sum_{s \in S_{it-1}} x_{is} - \sum_{s \in S_{it}} x_{is} + \sum_{s \in S_{it+1}} x_{is} \leq b_{rit} + g_{rit} z_{rit} \qquad (r = R_6\text{–}1,\ (i,t) \in D_r) \qquad (7)$$

Constraints (8) cover sub-requirement R_7–1, which aims at providing employees a k hours layup between consecutive shifts. Set S_{it}^k contains all pairs of shifts $(s_1 \in S_{it-1}, s_2 \in S_{it})$ between which the layup is less than k hours. The requirement is violated if both of these shifts are assigned to the same employee.

$$x_{is_1} + x_{is_2} \leq b_{rit} + g_{rit} z_{rit} \qquad (r = R_7\text{–}1,\ (i,t) \in D_r,\ (s_1, s_2) \in S_{it}^k) \qquad (8)$$

Note that the shifts (s_1, s_2) are not included in the domain D_r as a maximum of one violation per day is possible. Constraints (9) address sub-requirement R_8–1, which aims at assigning employees either no weekend shifts or one shift on each day of the weekend.

The binary variable y_{iw} indicate whether employee $i \in I$ has weekend $w = (t_1, t_2) \in W$ off. This variable is reused to model requirement R_9.

$$2y_{iw} + \sum_{t \in T_w} \sum_{s \in S_{it}} x_{is} = b_{riw} + g_{riw} z_{riw} \qquad (r = R_8\text{–}1,\ (i,w) \in D_r) \qquad (9)$$

Constraints (10) impose a minimum number of weekends off per employee. For this purpose we introduce set R_q^{sub} which comprise all the sub-requirements of requirement $q = R_9$.

$$\sum_{w \in W} y_{iw} \geq b_{ri} + g_{ri} z_{ri} \qquad (r \in R_9^{sub},\ i \in D_r) \qquad (10)$$

Constraints (11) cover requirement R_{10}, which aims at not exceeding the target workloads of employees.

$$\sum_{t \in T} \sum_{s \in S_{it}} l_s x_{is} \leq b_{ri} + g_{ri} z_{ri} \qquad (r \in R_{10}^{sub},\ i \in D_r) \qquad (11)$$

The left-hand side indicates the total workload of employee i by summing over all days $t \in T$ and shifts planned on that day. If this sum exceeds the target value b_{ri}, variable z_{ri} is equal to one to note the violation. In this case, the right-hand side is equal to $b_{ri} + g_{ri}$, which is a hard upper bound for the workload. Analogeously, constraints (12) cover requirement R_{11}, which aims at achieving the target workloads of employees.

$$\sum_{t \in T} \sum_{s \in S_{it}} l_s x_{is} \geq b_{ri} + g_{ri} z_{ri} \qquad (r \in R_{11}^{sub},\ i \in D_r) \qquad (12)$$

$$x_{is} \in \{0,1\} \qquad (i \in I_s,\ s \in S) \qquad (13)$$

$$y_{iw} \in \{0,1\} \qquad (i \in I,\ w \in W) \qquad (14)$$

$$z_{rd} \in \{0,1\} \qquad (r \in R^{sub},\ d \in D_r) \qquad (15)$$

5.4 Model size reduction

The size of the formulation in terms of constraints can be reduced with the following techniques. A moderate reduction can be achieved by aggregating sub-requirements of the same requirement which have overlapping domains. More precisely, each requirement with two or more kinks in the mapping function can be described by only one constraint per employee. Thereby, the deviational variables z_{rd} denote the position of the violation in the mapping function. In doing so, we need to define set $R_{q,i}^{sub}$ containing all sub-requirements of requirement q relevant for employee i. And we introduce parameter $\widehat{g}_{ri}$, which captures the distance between one kink and its adjacent kink with lower acceptance level. Formally:

$$\widehat{g}_{ri} = g_{ri} - \max_{r' \in R_{qi}^{sub} : |g_{ri}| > |g_{r'i}|} (g_{r'i}).$$

For example for i =Ann, we aggregate sub-requirements $r = R_9$–1, $r' = R_9$–2 using $R^{sub}_{9,Ann} = \{r, r'\}$, $\widehat{g}_{ri} = g_{ri} - g_{r'i} = -2 - (-1) = -1$ and $\widehat{g}_{r'i} = g_{r'i} = -1$ as shown in constraints (16).

$$\sum_{w \in W} y_{iw} \geq \max_{r \in R^{sub}_{9,i}} (b_{ri}) + \sum_{r \in R^{sub}_{9,i}} \widehat{g}_{ri} z_{ri} \qquad \left(i \in \bigcup_{r \in R^{sub}_{9}} D_r \right) \tag{16}$$

Analogously, we aggregate (11) and (12):

$$\sum_{t \in T} \sum_{s \in S_{it}} l_s x_{is} \leq \min_{r \in R^{sub}_{10,i}} (b_{ri}) + \sum_{r \in R^{sub}_{10,i}} \widehat{g}_{ri} z_{ri} \qquad \left(i \in \bigcup_{r \in R^{sub}_{10}} D_r \right) \tag{17}$$

$$\sum_{t \in T} \sum_{s \in S_{it}} l_s x_{is} \geq \max_{r \in R^{sub}_{11,i}} (b_{ri}) + \sum_{r \in R^{sub}_{11,i}} \widehat{g}_{ri} z_{ri} \qquad \left(i \in \bigcup_{r \in R^{sub}_{11}} D_r \right) \tag{18}$$

The model size can further be reduced by aggregating sub-requirements of different requirements. This is possible when two requirements are structurally similar (identical LHS), have overlapping domains and share the same target values. For example, sub-requirements $r_1 = R_{10}$–1, $r_2 = R_{10}$–2, $r_3 = R_{11}$–1, and $r_4 = R_{12}$–2 are aggregated as shown in constraints (19).

$$\sum_{t \in T} \sum_{s \in S_{it}} l_s x_{is} = \min_{r \in R^{sub}_{10,i}} (b_{ri}) + \sum_{r \in R^{sub}_{10,i} \cup R^{sub}_{11,i}} \widehat{g}_{ri} z_{ri} \qquad \left(i \in \bigcup_{r \in R^{sub}_{10}} D_r \cap \bigcup_{r \in R^{sub}_{11}} D_r \right) \tag{19}$$

We note that for this last aggregation, $\min_{r \in R^{sub}_{10,i}} (b_{ri}) = \max_{r \in R^{sub}_{11,i}} (b_{ri})$ must apply.

In Section 6, we compare the performance of the basic model formulation (BF) of Section 5.3 with the performance of the aggregated model formulation (AF):

$$(BF) \quad \begin{cases} \text{Min} & (1) \\ \text{s.t.} & (2) - (9) \\ & (10) - (12) \\ & (13) - (15) \end{cases} \qquad (AF) \quad \begin{cases} \text{Min} & (1) \\ \text{s.t.} & (2) - (9) \\ & (16), (19) \\ & (13) - (15) \end{cases}$$

5.5 Optimal solution

Figure 4 shows an optimal schedule of the illustrative example. All the requirements violations are listed in ascending order to their corresponding acceptance level.

Both model formulations (BF and AF) lead to the same schedule, and the corresponding CPU times are negligible ($<<$ 1 s).

6 Computational analysis

We conduct an experimental analysis to evaluate the performance of the proposed approach. In Subsection 6.1, we describe the 27 problem instances that we derived from real-world data. In Subsection 6.2, we use these instances to evaluate the performance of the basic formulation (BF) and the aggregated formulation (AF).

	Week 1							Week 2						
Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Ann	B_1	A_2	AS_3	A_4	B_5			A_8	AS_9		AS_{11}	A_{12}	A_{13}	A_{14}
Bob		L_2	B_3	L_4		B_6	B_7	L_8	B_9	L_{10}			B_{13}	B_{14}
Dan	A_1	E_2	E_3	E_4	A_5			E_8	A_9	A_{10}	B_{11}	L_{12}		
Eva			A_3			A_6	A_7				A_{11}	E_{12}		
Gil	E_1				E_5				E_9	E_{10}	E_{11}			

Sub-requirements violations:

a	r	d
60	R_6–1	i =Ann, $t = 10$
60	R_6–1	i =Bob, $t = 5$
60	R_9–1	i =Bob
70	R_{10}–1	i =Ann
80	R_9–1	i =Ann

Fig. 4 Optimal schedule

6.1 Test set

The planning horizon of each instance is four weeks which coincides with the planning horizon of real-world instances. The requirements to be considered in each instance are the same as presented in Table 1. This selection reflects the diversity of requirements that emerge in real-world instances. Our test instances were constructed such that they differ with respect to three complexity parameters:

- The number of available employees NE: We generated small-sized instances with 10 employees, medium-sized instances with 20 employees and large-sized instances with 30 employees. For each employee, we randomly selected a target workload from the set {80,88,...,160}.
- The workload ratio WR: Given the target workloads of employees, the workload ratio determines the number of eight-hour shifts to be performed. The number of shifts is obtained by multiplying the sum of target workloads of all employees by WR and dividing the result by 8 (the length of a shift). We generated instances with a workload ratio of 0.9, 1, and 1.1. For each shift, we randomly determined the start time and the set of employees who are qualified to perform it. It is ensured that the start times of shifts are well distributed across the planning horizon.
- The decomposition factor DR: The decomposition factor determines the number of kinks in the mapping functions of requirements R_{10}, R_{11}. We generated instances with a decomposition factor of 2, 6 and 10. Requirement R_9 is always decomposed into three sub-requirements. All other soft requirements are decomposed into a single sub-requirement.

The domain, target values and acceptance values of sub-requirements were defined randomly within ranges derived from the real-world instances. In total, we generated 27 instances, one instance for each possible combination of the three complexity parameters.

6.2 Numerical results

We applied the proposed approach with the basic formulation (BF) and the aggregated formulation (AF) to the 27 problem instances described in the previous subsection. Both formulations were implemented in AMPL and solved with the Gurobi Optimizer 5.6.3. All computations were performed on a standard PC with a 3.40GHz Intel i7 CPU and 4GB RAM. We prescribed a CPU time limit of 300 seconds per iteration.

The numerical results are presented in Tables 3 and 4. Columns two, three and four of Table 3 contain the values of the complexity parameters of the corresponding

instance. Column five states the number of iterations that were performed. Note that the number of iterations is equal to the number of different acceptance levels. Columns six and seven and eight and nine list the results for the formulations (BF) and (AF), respectively. For both formulations, we state the number of iterations that were solved to optimality (# Opt.) and the required CPU time (CPU) in seconds. An instance is solved to optimality only if all iterations were solved to optimality. These instances are with an asterisk. Furthermore, in column ten we note which formulation leads within the time limit to the better final schedule, i.e., a schedule with less sub-requirement violations with a lower acceptance value. Thereby, a bar indicates that both schedules are equally good. Note that, in contrast to formulation (BF), formulation (AF) has solved instances 5 and 21 to optimality. Nevertheless the resulting schedules are equally good.

Small- and medium-sized instances are solved to optimality in short CPU time by both formulations. In total, formulation (AF) is able to solve 10 instances to optimality, while formulation (BF) solves 8 instances to optimality. For those instances that were solved to optimality by both models, formulation (BF) has an average CPU time requirement of 51.8 sec, while formulation (AF) has an average CPU time requirement of 45.9 sec. Finally, formulation (AF) generates for eight instances better schedules and for only three instances worse schedules than formulation (BF).

Inst.	NE	WR	DR	#It.	BF		AF		Better schedule
					#Opt.	CPU	#Opt.	CPU	obtained by
1	10	0.9	2	26	26 ∗	17.6	26 ∗	8.7	—
2	10	0.9	6	58	58 ∗	41.9	58 ∗	19.0	—
3	10	0.9	10	69	69 ∗	8.8	69 ∗	6.8	—
4	10	1	2	27	26	456.0	26	442.8	—
5	10	1	6	56	54	976.8	56 ∗	357.6	—
6	10	1	10	73	72	488.0	73 ∗	303.1	AF
7	10	1.1	2	27	27 ∗	336.2	25	774.3	—
8	10	1.1	6	59	59 ∗	10.4	59 ∗	11.8	—
9	10	1.1	10	76	76 ∗	44.2	76 ∗	64.8	—
10	20	0.9	2	27	25	866.1	26	758.6	AF
11	20	0.9	6	62	61	542.1	60	791.1	—
12	20	0.9	10	71	71 ∗	35.3	71 ∗	26.2	—
13	20	1	2	28	27	785.6	27	728.0	AF
14	20	1	6	62	58	1,532.1	58	1,608.6	BF
15	20	1	10	79	77	1,140.6	76	1,231.5	—
16	20	1.1	2	28	22	1,957.5	22	1,956.1	AF
17	20	1.1	6	65	63	1,742.1	60	2,629.7	—
18	20	1.1	10	77	62	4,793.3	61	5,021.7	AF
19	30	0.9	2	28	28 ∗	204.8	28 ∗	184.1	—
20	30	0.9	6	64	57	2,726.2	56	2,769.3	—
21	30	0.9	10	79	78	771.8	79 ∗	274.9	—
22	30	1	2	27	24	1,028.3	25	1,078.4	BF
23	30	1	6	66	63	1,250.6	63	1,215.1	AF
24	30	1	10	78	73	1,700.5	73	1,896.2	—
25	30	1.1	2	28	23	1,710.3	18	3,097.4	BF
26	30	1.1	6	64	59	2,498.2	58	2,276.6	AF
27	30	1.1	10	76	68	3,100.8	68	2,944.7	AF

Table 3 Numerical results of the models BF and AF for all instances

Parameter		#Inst.	average #Viol.		average CPU		#Best		#Inst. Opt.	
			BF	AF	BF	AF	BF	AF	BF	AF
NE	10	9	25.1	25.1	264	221	0	1	6	7
	20	9	47.3	46.4	1,488	1,639	1	4	1	1
	30	9	60.8	60.3	1,666	1,749	2	3	1	2
WR	0.9	9	43.9	43.9	579	538	0	1	5	6
	1	9	22.3	21.7	1,040	985	2	3	0	2
	1.1	9	67	66.3	1,799	2,086	1	4	3	2
DR	2	9	50.7	50.2	818	1,003	2	3	3	2
	6	9	42.2	41.7	1,258	1,298	1	2	2	3
	10	9	40.3	40.0	1,343	1,308	0	3	3	5
All instances		27	44.4	44.0	1,139	1,203	3	8	8	10

Table 4 Consolidated results w.r.t. the complexity parameters

In Table 4, columns one and two display the values of the complexity parameters and column three indicates the number of instances with these complexity parameter values. Columns four and five refer to the average number of violations per instance (average #Viol) for the formulations (BF) and (AF), respectively. Columns six and seven present the average required CPU time (average CPU) in seconds. Columns eight and nine (#Best) contain the information how often each formulation obtained a better final schedule within the time limit than the other formulation. Finally, columns ten and eleven contain the number of instances that were solved to optimality (#Inst. Opt.).

On average formulation (BF) has 44.4 violations whereas formulation (AF) has 44 violations. The required CPU time is on average 64 seconds lower for formulation (BF). Formulation (AF) requires more time in later iterations because formulation (AF) generates in some iterations a better schedule, hence it has tighter restrictions for the next iterations and consequently needs more CPU time. Parameters NE and WR mainly drive the required CPU time. The number of employees NE defines the size of an instance and the workload ratio WR the complexity. For a larger decomposition factor ($DR = 10$) or a larger workload ratio ($WR = 1.1$), formulation (AF) is superior and generates better schedules.

7 Conclusions

We have presented a lexicographic goal programming approach for a real-world staff assignment problem. The problem consists of assigning employees to work shifts subject to hard and soft requirements. Unlike existing lexicographic goal programming approaches which rely on a ranking of requirements, our approach is able to account for trade-offs between requirements. To this end, we decompose the soft requirements into sets of sub-requirements. The ranking is then specified for the sub-requirements according to the preferences of the decision maker. To reduce the size of the resulting goal programming formulation, we propose techniques to aggregate constraints. The applicability of the approach is demonstrated for a collection of problem instances that were derived from real-world data. The approach can solve small- and medium-sized instances to optimality in short CPU times. The aggregation techniques improved the performance of the approach for instances with a large number of sub-requirements. The software provider involved in this research benefits from our research in two ways.

First, the solutions generated by our approach enable the provider to evaluate the current performance of its software. Second, the provider gains insights into the structure of optimal solutions which is helpful for improving the performance of its software.

In future research, we will extend the approach to allow for a fair distribution of sub-requirement violations among employees. This extension can be incorporated by imposing soft upper bounds on the number of violations per employee. Another direction for further research is the development of BLP-based heuristics for large-scale instances. The idea is to reduce the search space in each iteration by fixing a part of the decision variables.

References

1. Beaulieu, H., Ferland, J. A., Gendron, B., Michelon, P., A mathematical programming approach for scheduling physicians in the emergency room, Health Care Management Science, 3, 193–200 (2000)
2. Berrada, I., Ferland, J. A., Michelon, P., A multi-objective approach to nurse scheduling with both hard and soft constraints, Socio-Economic Planning Sciences, 30, 183–193 (1996)
3. Chang, C.-T., Mixed binary interval goal programming, Journal of the Operational Research Society, 57, 469–473 (2006)
4. Chang, C.-T., Lin, T.-C., Interval goal programming for S-shaped penalty function, European Journal of Operational Research, 199, 9–20 (2009)
5. Charnes, A., Collomb, B., Optimal economic stabilization policy: Linear goal-interval programming models, Socio-Economic Planning Sciences, 6, 431–435 (1972)
6. Charnes, A., Cooper, W. W., Management models and industrial applications of linear programming, John Wiley & Sons (1961)
7. Charnes, A., Cooper, W. W., Ferguson, R. O., Optimal estimation of executive compensation by linear programming, Management Science, 1, 138–151 (1955)
8. Charnes, A., Cooper, W. W., Harrald, J., Karwan, K. R., Wallace, W. A., A goal interval programming model for resource allocation in a marine environmental protection program, Journal of Environmental Economics and Management, 3, 347–362 (1976)
9. Ignizio, J. P., Optimal maintenance headcount allocation: an application of Chebyshev Goal Programming, International Journal of Production Research, 42, 201–210 (2004)
10. Jones, D. F., Tamiz, M., Expanding the flexibility of goal programming via preference modelling techniques, Omega, 23, 41–48, (1995)
11. Jones, D. F., Tamiz, M., Goal programming in the period 1990-2000, in: Ehrgott, M., Gandibleux, X. (eds.), Multiple criteria optimization - state of the art annotated bibliographic surveys, Kluwer Academic Publishers, Dordrecht, 129–170 (2002)
12. Kvanli, A. H., Financial planning using goal programming, Omega, 8, 207–218 (1980)
13. Romero, C., A general structure of achievement function for a goal programming model, European Journal of Operational Research, 153, 675–686 (2004)
14. Tamiz, M., Jones, D. F., El-Darzi, E., A review of goal programming and its applications, Annals of Operations Research, 58, 39–53 (1995)
15. Van den Bergh, J., Belïen, J., De Bruecker, P., Demeulemeester, E., Personell scheduling: a literature review, European Journal of Operational Research, 226, 367–385 (2013)

MISTA 2015

A list-scheduling approach for the planning of assessment centers

Adrian Zimmermann · Norbert Trautmann

Abstract Many companies run assessment centers in order to select candidates for open job positions. During an assessment center, each candidate performs a set of exercises; thereby, the candidates are evaluated by so-called assessors. Additional constraints such as preparation and evaluation times, participation of actors in the exercises, no-go constraints, and time windows for lunch breaks contribute to the complexity of planning such assessment centers. We propose a list-scheduling approach to solve this planning problem heuristically; to this end, we develop novel procedures for devising an appropriate scheduling list and for generating a feasible schedule. In an experimental performance analysis, we applied this approach to a set of problem instances that represent or are devised from real cases. It turns out that the approach generates optimal or near-optimal schedules within relatively short CPU times.

1 Introduction

Empirical evidence indicates that human capital is a key success factor for a firm's performance (cf., e.g., [3]). The task of developing the human capital is performed by the firm's human resources managers. E.g., they must recruit new employees from a pool of candidates. Such personnel decisions require an evaluation of the candidates' skills, know-how, and personality. To obtain these evaluations, human resources managers often conduct assessment centers (cf., e.g, [6]). During an assessment center, the candidates perform a series of exercises during which they are observed and evaluated by so-called assessors, e.g., usually managers or psychologists. Based on the observations made during the exercises, the assessors derive an overall evaluation for each candidate.

The planning problem discussed in this paper consists of determining (1) the start times of all exercises and lunch breaks of such an assessment center, and (2) a feasible assignment of assessors and other personnel, such as actors, to exercises. The objective

Adrian Zimmermann
University of Bern
E-mail: adrian.zimmermann@pqm.unibe.ch

Norbert Trautmann
University of Bern
E-mail: norbert.trautmann@pqm.unibe.ch

is to minimize the duration of the assessment center. Each candidate should be observed by approximately half the number of all available assessors. So-called no-go restrictions prohibit the assignment of specific assessors to a candidate. Finally, the requirements for candidates, assessors, and other personnel can vary over the course of an exercise.

To the best of our knowledge, this planning problem has not been treated in the literature. Related planning problems are the multi-mode resource-constrained project scheduling problem (MRCPSP) and the timetabling problem. Timetabling deals with constructing timetables for schools or universities (cf., e.g., [9]). In timetabling, it is assumed that all classes have the same duration; this is not the case for the exercises of the assessment center. In this work, we interpret the assessment center planning problem as the extension of an MRCPSP. The exercises performed by each candidate correspond to the project activities, and the assessment center participants can be interpreted as renewable resources, i.e., resources with a constant capacity in each time period of the planning horizon. Each feasible assignment of assessors to an exercise corresponds to an alternative execution mode of the project activity. In the literature, only a few variants of the MRCPSP with specific mode-assignment constraints have been studied (cf. [4], [5], [7], and [8]); none of these variants corresponds to the assessor-assignment constraints of the planning problem discussed here.

In this paper, we propose a new list-scheduling approach for planning assessment centers heuristically. Our heuristic is based on a list-scheduling approach (cf., e.g., [1]): first, we order the activities in a list and apply randomized sampling; then, we schedule the activities sequentially using a novel schedule generation scheme (SGS). We devise novel criteria for ordering the activities in the list, and for assigning assessors to the activities. We apply our approach to four real-world instances and to 30 test instances that we constructed based on real-world data. Our computational results indicate that our heuristic approach generates optimal or near-optimal schedules within short CPU time.

The remainder of this paper is structured as follows. In Section 2, we describe the assessment center planning problem and present an illustrative example. In Section 3, we present our list-generating criteria and the SGS. In Section 4, we discuss our computational results. In Section 5, we give some concluding remarks.

2 Planning problem

In Section 2.1, we state the planning problem discussed in this paper. In Section 2.2, we provide an illustrative example.

2.1 Overview

The assessment center planning problem discussed in this paper has been reported to us by a Swiss-based service provider in the human resources sector. This service provider organizes assessment centers for firms.

An assessment center takes place during one day; there is no prescribed maximum length of the assessment center. There are three different groups of participants: (1) the candidates, who need to perform a series of exercises, (2) the assessors, who observe and evaluate the candidates during those exercises, and (3) the actors, who in exercises designed as role-plays represent, e.g., an unhappy customer with whom the candidate

must interact with. To improve the comparability between the overall evaluations, each candidate must perform the same exercises. Each candidate has to perform each exercise exactly once. Depending on the exercise type, one or two assessors must be present.

When assigning assessors to candidates the following restrictions must be taken into account. To ensure an objective overall evaluation, each candidate should be observed by at least half the number of assessors rounded down. Do to fairness considerations, each candidate should be observed by approximately the same number of assessors. Hence, no candidate should be observed by more than half the number of assessors rounded up plus one. Once an assessor has observed a candidate, there is no limit on how many more times this assessor can observe that same candidate. We refer to these restrictions as the 50%-assignment rule. Additionally, there might exist so-called no-go relationships that prohibit the assignment of an assessor to a candidate. Such relationships arise if the candidate and the assessor know each other personally. We assume that an assignment of the assessors to the candidates exists which is feasible w.r.t. all these constraints.

The requirements for candidates, assessors, and actors can vary over the course of an exercise. Exercises often include a preparation time during which only the candidate is required. During the actual execution of the exercise, the candidate is then joined by the assessors and, if required, the actor. After the execution, the assessors and actors in general require some time to record their observations; this time is called evaluation time and can be different for assessors and actors. Figure 1 illustrates these time-varying requirements. The grey-shaded bars represent the time during which the participants are occupied over the course of an exercise.

Besides exercises, each candidate must have a lunch break within a prescribed time window.

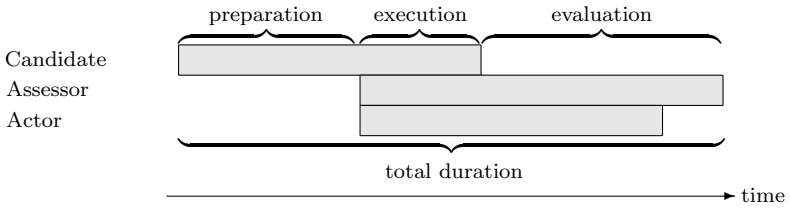


Fig. 1 Varying requirements for candidates, assessors, and actors during an exercise

2.2 Illustrative example

In this section we illustrate the planning problem with an example that is based on real-world data. The number of participants and exercises, and the no-go relationships are shown in Table 1. The lower limit on the number of different assessors that have to be assigned to each candidate is 2, and the upper limit is 3.

The data of the exercises and the lunch break is listed in Table 2. The duration and the preparation, execution, and evaluation times are stated in 5-minute time periods.

Candidates	Assessors	Actors	Exercises	No-go relationships
{C1,C2,C3}	{A1,A2,A3,A4}	{R1}	{E1,E2,E3,E4}	(C2,A4)

Table 1 Illustrative example: main data

Only the candidates have a lunch break and cannot be involved in any other activity during the lunch break. The earliest possible and the latest possible start times of the lunch break are the time periods 30 and 78, respectively.

Exercise	E1	E2	E3	E4	Lunch break
Total duration	18	29	16	6	6
Preparation time	8	19	0	0	-
Execution time	8	8	12	6	-
Evaluation time	2	2	4	0	-
Required # of assessors	2	2	2	1	-
Required # of actors	1	-	-	-	-

Table 2 Illustrative example: exercise and lunch break data

Since there are four exercises which have to be performed by each of the three candidates once, there is a total of 12 activities. Table 3 shows the indices of these activities. The lunch breaks for the candidates are denoted with LB.

Exercise	E1			E2			E3			E4			Lunch break		
Candidate	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
Activity	1	2	3	4	5	6	7	8	9	10	11	12	LB	LB	LB

Table 3 Illustrative example: activities

3 List-scheduling approach

In this section, we present our list-scheduling approach. We model each candidate and each assessor as a renewable resource with a capacity of one, and we model the set of actors as a renewable resource with a capacity that equals the number of actors. We divide the planning horizon into 5-minute time periods.

An overview of the procedure is depicted as a flowchart in Figure 2. First, the activities are ordered in a list by applying appropriate sorting criteria (cf. Section 3.1). Then, the order of the activities is modified by applying a random sampling method (cf. [2]), i.e., with some low probability, the order of two activities that are in adjacent positions is switched. Based on the order of the activities in the list, the activities and lunch breaks are then scheduled with the help of a schedule generation scheme (cf. Section 3.2). In Section 3.3, we describe how the assessors are assigned to the activities. In Section 3.4, we depict two schedules for the illustrative example obtained by our list-scheduling approach.

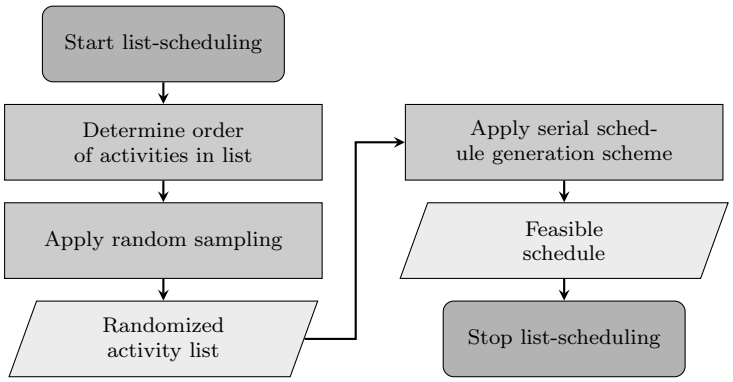


Fig. 2 Flowchart: list-scheduling procedure

3.1 List generation

Priority rules proposed in the literature are not appropriate for determining the order of the activities of an assessment center, since (1) there are no precedence constraints between the exercises, and (2) there is no difference between the activities that refer the same exercise. In what follows, we derive novel criteria for ordering the activities in the list.

The list-generation procedure is depicted as a flowchart in Figure 3. First, the number of positions in the list L is calculated by multiplying the number of exercises E with the number of candidates C. Next, the candidates $1, \ldots, C$ are assigned E times to the positions $1, \ldots, L$. Finally, the exercises are assigned to the positions $1, \ldots, L$ such that each exercise is included once for each candidate.

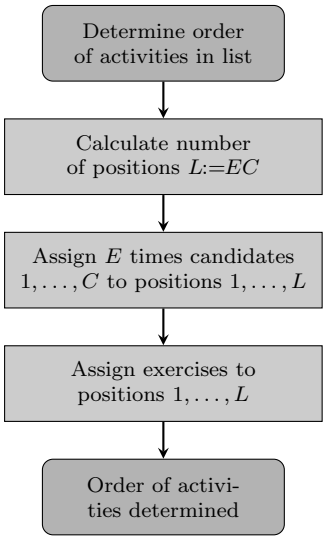


Fig. 3 Flowchart: list-generation procedure

The list consists of three parts, for which the following criteria apply (cf. Figure 4). If these criteria are not sufficient to distinguish between exercises, we use the activity index as a tie-breaker.

- To the last C positions of the list, an exercise with the shortest duration is assigned. By scheduling this activity last, some waiting time of the participants might be reduced.
- To the positions in the first part of the list, an exercise with the shortest preparation time is assigned. The number of positions where this criteria applies corresponds to the number of activities that can be performed simultaneously w.r.t. the number of assessors required. By scheduling the corresponding activities first, some of the waiting time of the assessors can be reduced.
- To the positions in the middle part of the list, the exercises are assigned such that the distance between two positions of the same exercise is maximized. As secondary criterion, either the minimum or the maximum preparation time is used. With each of the two secondary criteria, a list is generated. By scheduling activities referring to different exercises such that they run (partially) in parallel, some candidates can already begin with the preparation of an exercise, while the remaining candidates are occupied with the execution of another exercise.

The two alternative lists for the illustrative example are depicted in columns four and six of Table 4. The dashed lines indicate the three different parts of the list.

Position	1	2	...													L
Candidate	1	2	...	C	1	2	...	C	1	2	...	C	1	2	...	C
Criteria	Shortest preparation time				Largest distance between same exercise								Shortest duration			

Fig. 4 Criteria for assigning exercises to positions

Position	Candidate	Secondary criteria			
		maximum preparation time		minimum preparation time	
		exercise	activity	exercise	activity
1	C1	E3	7	E3	7
2	C2	E3	8	E3	8
3	C3	E2	6	E1	3
4	C1	E1	1	E2	4
5	C2	E2	5	E1	2
6	C3	E1	3	E2	6
7	C1	E2	4	E1	1
8	C2	E1	2	E2	5
9	C3	E3	9	E3	9
10	C1	E4	10	E4	10
11	C2	E4	11	E4	11
12	C3	E4	12	E4	12

Table 4 The two alternative lists for the illustrative example

3.2 Serial schedule generation scheme

The flowchart of the serial schedule generation scheme is depicted in Figure 5. Based on their order in the list, the activities are selected sequentially and scheduled as early

as possible. To this end, the earliest resource-feasible start time t^S is determined. It may occur that when the selected activity is scheduled at t^S, it is no longer possible to schedule the lunch break within its time window. In that case, the corresponding lunch break is scheduled first, and t^S is calculated anew for the selected activity. Once all activities in the list have been scheduled, all remaining lunch breaks are scheduled.

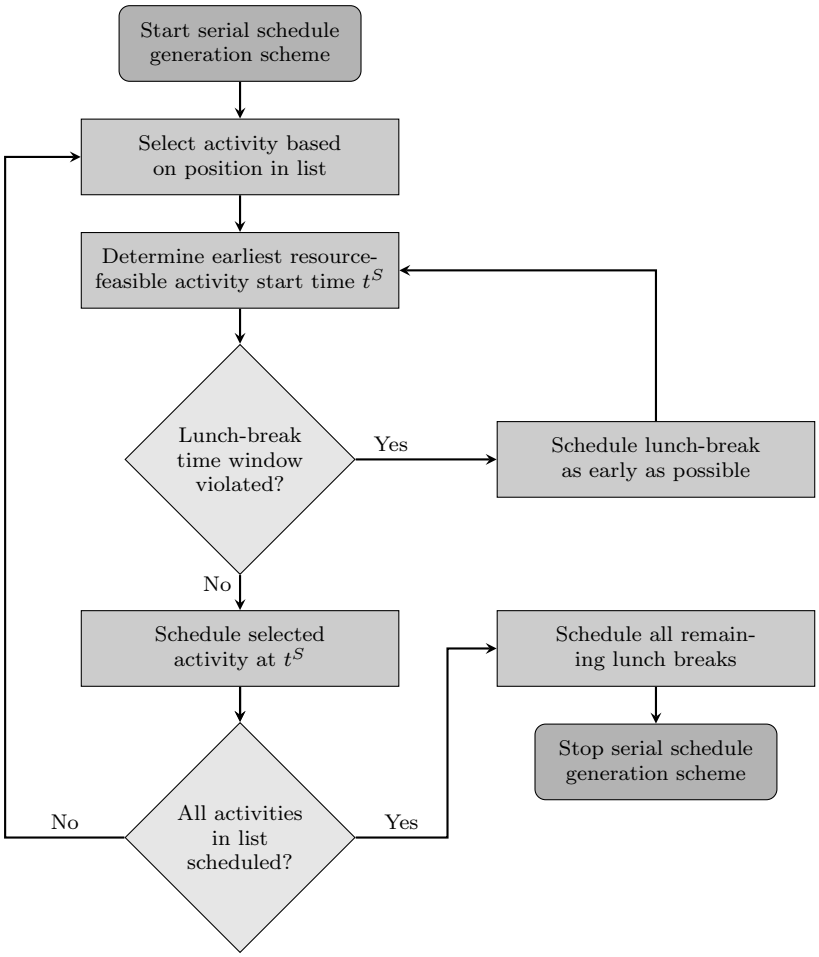


Fig. 5 Flowchart: schedule generation scheme

3.2.1 Earliest resource-feasible start time

The flowchart of the procedure that determines the earliest resource-feasible start time t^S for the selected activity is depicted in Figure 6. The procedure starts by selecting the first time period of the planning horizon. From the selected time period onwards, the availability of the required candidate and, in the case of a role-play exercise, the

actors is determined. When determining the availabilities, only those time periods are considered during which the respective participants have to be present.

Once the candidate (and, in the case of a role-play exercise, an actor) is available, a sub-procedure for the assessor assignment is initiated (see Section 3.3). If the assessor assignment is successful, t^S is set to the currently selected time period and the procedure stops. Otherwise, the next time period of the planning horizon is selected, and the availabilities of the participants are examined anew.

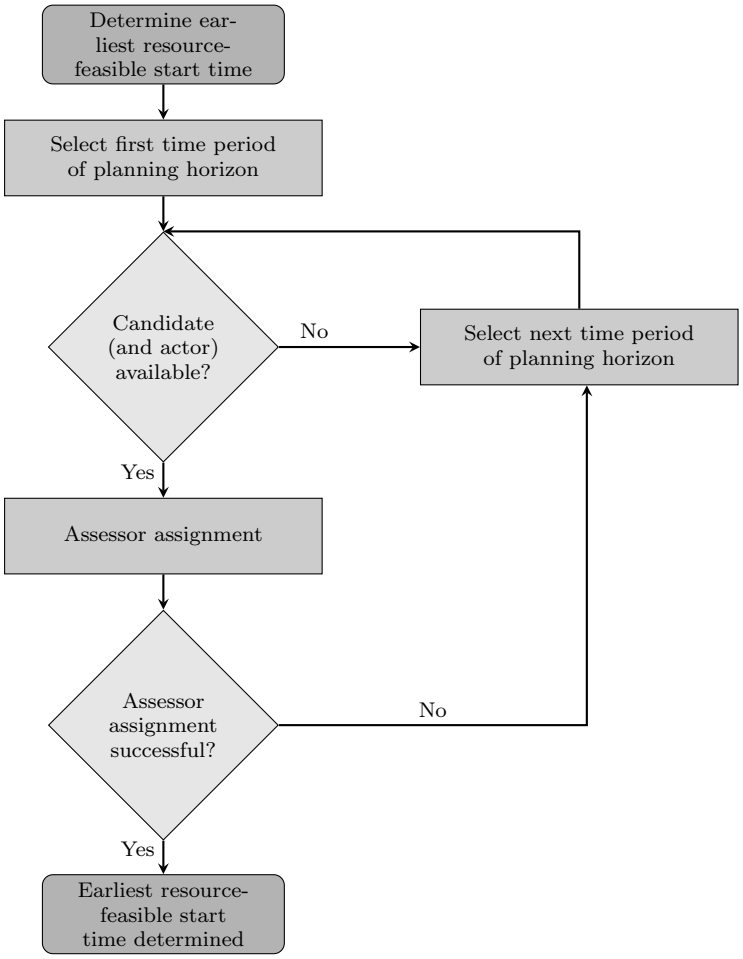


Fig. 6 Flowchart: determine earliest resource-feasible start time

3.2.2 Lunch breaks

Before scheduling an activity at t^S, the following procedure determines whether the lunch-break time window is violated. Assuming that the activity is scheduled at t^S,

the candidate's resulting availability is determined throughout the lunch-break time window. If it is impossible to schedule the lunch break within this time window, the lunch break is scheduled before the activity.

The above-described situation might never arise and, thus, some lunch breaks might never be considered during the scheduling of the activities in the list. After all activities have been scheduled, all remaining lunch breaks are scheduled at their earliest resource-feasible start times.

Scheduling the lunch break after the completion of the last activity might increase the duration of the schedule unnecessarily. Starting with the candidate that has the smallest index value, for each candidate we determine whether the lunch break is performed after the last activity. If the schedule duration is reduced, we switch the order of that last activity and the lunch break. The reduction of a schedule's duration by such a switch is illustrated in Figure 7. The light grey bars correspond to activities performed by the same candidate. The dark grey bar represents the time during which the assessor is occupied with another candidate's activity.

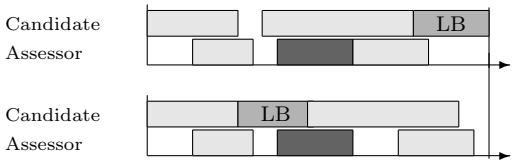


Fig. 7 Lunch break switch

3.3 Assessor assignment

The flowchart of the procedure that assigns the assessors to an activity is depicted in Figure 8. First, the assessors are ordered randomly, i.e., the assessor assignment may vary in different runs of the procedure. Then, the assessors are selected sequentially based on that random order. An assessor is temporarily assigned if (1) the assignment does not violate the lower and upper limit of the 50%-assignment rule, (2) there is no no-go relationship with the required candidate, and (3) the assessor is available.

The procedure stops if the required number of assessors has been assigned and the temporarily assignments are fixed, or if all assessor have been examined. If the required number of feasible assessors could not be assigned, any temporarily assignments to the selected activity are reversed.

Whether an assessor assignment violates the 50% assignment rule depends on which assessors have observed the required candidate thus far. The following two cases and corresponding feasibility criteria are considered.

a) The number of different assessors that observe the required candidate equals the upper limit. Only assessors that have already observed the required candidate at least once are feasible.
b) The number of different assessors that observe the required candidate lies below the lower limit. Only assessors that have not yet observed the required candidate are feasible.

In any other case, no additional feasibility criteria apply. Note that the criteria in a) or b) might become active after the first assessor has been assigned.

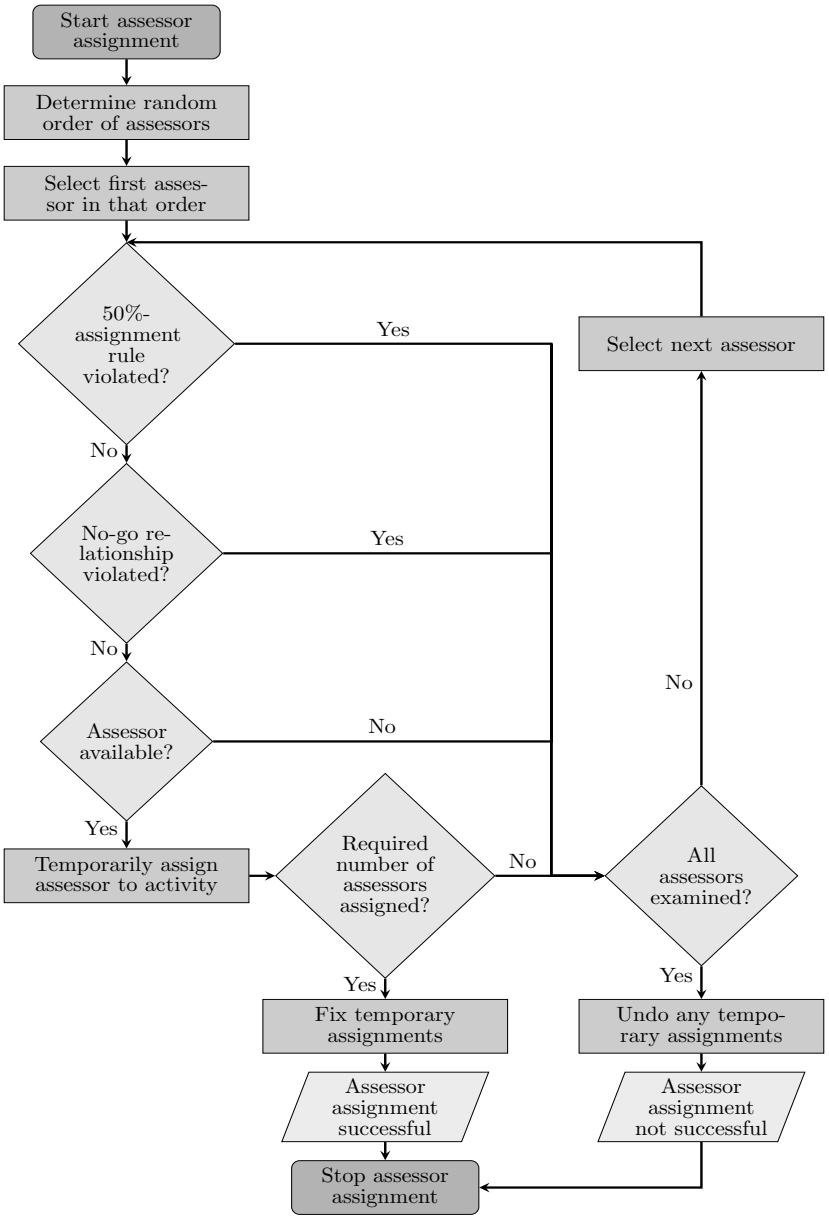


Fig. 8 Flowchart: assign required number of assessors to activity

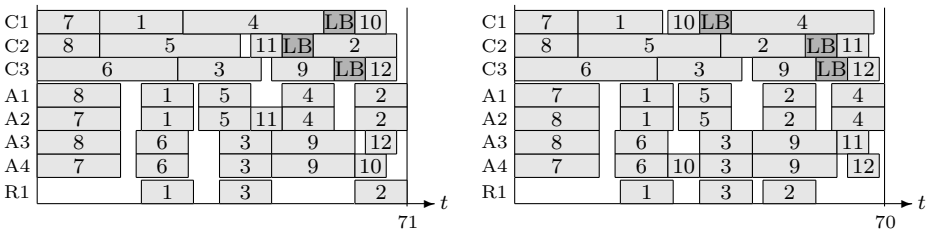


Fig. 9 Illustrative example: schedules obtained by our SGS without random sampling (left) and with random sampling (right)

3.4 Results for the illustrative example

For the illustrative example, we obtained an optimal schedule with a duration of 70 time units by applying the mixed-integer linear program (MILP) presented in [10]. Two alternative schedules for the illustrative example obtained by our list-scheduling approach are depicted in Figure 9. The duration of the best schedule generated with the list based on the maximum-preparation-time criteria and random assessor assignment is 71 time units. With the random sampling approach, an optimal schedule is generated.

4 Computational results

We have implemented the list-scheduling approach presented in Section 3 in Visual Basic for Applications (VBA) in Microsoft Excel. We compare our approach against the MILP presented in [10]. All calculations were performed on a desktop PC with an Intel Core i5 CPU with 2.7GHz and 4GB RAM.

In Section 4.1, we describe the test instances that we used in our computational study. In Section 4.2, we discuss our computational results.

4.1 Test set

Four real-world instances of our test set were provided by the Swiss-based service provider. All these instances include five different exercises and one lunch break for each candidate. The number of candidates (C), assessors (A), actors (R), exercises (E), and activities (I) of the instances are listed in the first columns of Table 5.

Based on the data of the real-world instances, we have generated 30 additional test instances. We have designed the largest test instance (test instance 30) such that it corresponds to real-world instance 4, except that test instance 30 contains one additional no-go relationship. We have systematically varied the number of candidates from four to six and the number of assessors from five to nine. For each combination of the different number of candidates and assessors, we have generated an instance with four and an instance with five exercises including one or two role-plays and two or three actors, respectively. Each test instance contains one no-go relationship.

4.2 Experimental design and results

For the solution of the MILP, we have used the Gurobi Optimizer 5.6 as solver. We have limited the CPU time of the solver to 3'600 seconds for the real-world instances, and to 1'200 seconds for the test instances. For each of the two activity lists, we have generated 200 schedules with the random sampling method. Hence, per instance, our procedure has generated 400 schedules. After some preliminary tests, we set the probability for switching two activities adjacent in the list to 15%.

The results of our analysis are shown in Table 5. The duration of the best schedules obtained is displayed in the columns titled with [dur]. The column titled with [gap] shows the MIP gap of the solution obtained by the MILP. The entry $*$ indicates that the instance was solved to optimality within the prescribed time limit. The column [CPU] shows after how many seconds the MILP obtained the best solution. The last column [CPU] shows the total running time of our algorithm in seconds. The columns [max. prep.] and [min. prep.] show the best results obtained with the lists generated by the maximum-preparation-time criteria and the minimum-preparation-time criteria, respectively. The best results obtained per instance are highlighted in bold.

First, we ran our heuristic without the random sampling procedure. We still perform 200 runs for each list, due to the possibility of randomly assigning assessors to the activities. The results are summarized in the column [no random sampling]. On average, the list generated with the maximum-preparation-time criteria performs better than the list generated with the minimum-preparation-time criteria. Seven test instances and real-world instance 3 are solved better with the latter criteria. The column [random sampling] shows that the random sampling procedure can improve a solution considerably. E.g., with the minimum-preparation-time criteria, the duration obtained for test instance 21 is 114 with and 122 without random sampling.

Compared to the MILP, the solution to the largest real-world instance obtained by our procedure is 5 time units shorter. The solutions to the remaining real-world instances are slightly longer, but they are obtained much faster. E.g., for real-world instance 3, the MILP obtains a solution that is one time unit shorter than the heuristic solution. The time required to obtain that solution is more than 50 minutes whereas the heuristic only requires 20 seconds.

5 Conclusions

In this work, we considered a real-world assessment center scheduling problem. We developed a list-scheduling approach for solving this problem heuristically. We applied our approach to solve real-world instances as well as several problem instances that were constructed based on real-world data. We showed that our approach consistently generates good schedules within a short amount of computational time.

Sometimes assessment centers include group exercises that are performed by multiple candidates simultaneously in the presence of several assessors. The list-scheduling approach presented in this paper should be extended by procedures for forming these groups and scheduling the activities accordingly. Additionally, we will integrate the presented approach into an MIP-based heuristic.

References

1. Adam, T. L., Chandy, K. M., Dickson, J. R., A comparison of list schedules for parallel processing systems, Communications of the ACM, 17, 685–689 (1974)
2. Hartmann, S. Project scheduling with multiple modes: a genetic algorithm, Annals of Operations Research, 102, 111–135 (2001)
3. Huselid, M. A., The impact of human resource management practices on turnover, productivity, and corporate financial performance, Academy of Management Journal 38, 635–672 (1995)
4. Li, H., Womer, K., Modeling the supply chain configuration problem with resource constraints, International Journal of Project Management, 26, 646–654 (2008)
5. Salewski, F., Schirmer, A., Drexl, A., Project scheduling under resource and mode identity constraints: model, complexity, methods, and application, European Journal of Operational Research, 102, 88–110 (1997)
6. Spychalski A. C., Quinones M. A., Gaugler B. B., Pohley K. (1997): A survey of assessment center practices in organizations in the United States, Personnel Psychology, 19, 195–203 (1997)
7. Tareghian, H. R., Taheri, S. H., A solution procedure for the discrete time, cost and quality tradeoff problem using electromagnetic scatter search, Applied Mathematics and Computation, 190, 1136–1145 (2007)
8. Tiwari, V., Patterson, J. H., Mabert, V. A., Scheduling projects with heterogeneous resources to meet time and quality objectives, European Journal of Operational Research, 193, 780–790 (2009)
9. de Werra, D., An introduction to timetabling. European Journal of Operational Research, 19, 151–162 (1985)
10. Zimmermann, A., Trautmann, N., Scheduling of assessment centers: an application of resource-constrained project scheduling, in: Fliedner, T., Kolisch, R., Naber, A. (eds.), Proceedings of the 14th International Conference on Project Management and Scheduling, 263–266 (2014)

Instances						MILP			Heuristic				
									no random sampling		random sampling		random sampling
									max. prep.	min. prep.	max. prep.	min. prep.	combination
real-world	C	A	R	E	I	dur.	gap [%]	CPU	dur.	dur.	dur.	dur.	CPU
1	7	10	2	5	42	**87**	8.0	2617	90	93	91	92	12
2	11	11	3	5	66	119	8.3	1788	115	116	115	**114**	36
3	9	11	3	5	54	**97**	6.3	3156	100	99	98	99	20
4	6	9	3	5	36	**85**	3.6	620	86	95	86	88	8
test set	C	A	R	E	I	dur.	gap [%]	CPU	dur.	dur.	dur.	dur.	CPU
1	4	5	2	4	20	**75**	4.0	35	79	77	79	77	3
2	4	6	2	4	20	**69**	2.9	61	70	75	**69**	75	3
3	4	7	2	4	20	**69**	2.9	1	73	73	**69**	73	3
4	4	8	2	4	20	**67**	*	2	**67**	**67**	**67**	**67**	3
5	4	9	2	4	20	**67**	*	1	**67**	**67**	**67**	**67**	3
6	5	5	2	4	25	**92**	2.2	15	101	98	**92**	**92**	4
7	5	6	2	4	25	**73**	8.0	139	79	75	75	75	4
8	5	7	2	4	25	**71**	5.6	29	73	73	73	73	4
9	5	8	2	4	25	**69**	2.9	14	75	75	**69**	75	4
10	5	9	2	4	25	**69**	2.9	1	73	73	**69**	**69**	4
11	6	5	2	4	30	110	1.8	777	114	111	111	**108**	6
12	6	6	2	4	30	84	7.1	586	85	85	84	**83**	6
13	6	7	2	4	30	**79**	8.9	415	**79**	**79**	**79**	**79**	6
14	6	8	2	4	30	**71**	5.6	55	75	75	75	75	6
15	6	9	2	4	30	**71**	5.6	16	73	73	73	73	6
16	4	5	3	5	24	97	9.3	90	102	98	98	**94**	4
17	4	6	3	5	24	**82**	2.4	7	**82**	89	**82**	89	4
18	4	7	3	5	24	**82**	2.4	8	**82**	87	**82**	87	4
19	4	8	3	5	24	**80**	*	14	**80**	**80**	**80**	**80**	4
20	4	9	3	5	24	**80**	*	2	**80**	**80**	**80**	**80**	4
21	5	5	3	5	30	113	2.7	128	119	122	**112**	114	6
22	5	6	3	5	30	**90**	11.1	494	92	92	94	92	6
23	5	7	3	5	30	**85**	5.9	592	89	88	88	88	6
24	5	8	3	5	30	**82**	2.4	21	**82**	89	**82**	89	6
25	5	9	3	5	30	**82**	2.4	56	**82**	87	**82**	87	6
26	6	5	3	5	36	137	3.7	522	138	140	134	**132**	8
27	6	6	3	5	36	104	8.7	637	103	**101**	105	**101**	8
28	6	7	3	5	36	98	9.2	258	102	97	**96**	**96**	8
29	6	8	3	5	36	**88**	9.1	700	90	91	89	91	8
30	6	9	3	5	36	**85**	5.9	620	86	95	86	88	8

Table 5 Computational results

MISTA 2015

Home Health Care Scheduling: A Case Study

Zhi Yuan · Armin Fügenschuh

Abstract This article provides a case study on the problem of scheduling nurses for home health care on a weekly basis. The list of health care tasks are available before the start of the week. Each client may require multiple visits per week, and they can specify the preferred day and the time window on each preferred day that they expect a visit. Besides, certain visits require a minimum number of days' difference in between. The nurse may also specify the preferred working day and the maximum working hours. The optimal schedule to be found should minimize the personnel cost as well as the total working time, without compromising the service quality. This problem is a combination of the staff rostering problem that consists in assigning health care tasks to a competent nurse on the appropriate day without exceeding her maximum working hours, and the vehicle routing problem with time windows, where an optimal route for each day's scheduled visits should be found respecting the time windows. We formulate this problem as an integer linear programming model based on the multi-commodity network flow formulation, and develop also problem specific greedy construction and local search approaches for it. The scalability of different approaches are studied, and a real-world instance is used for validating our proposed approach. An estimate of at least 10% cost reduction potential is observed comparing with the current manual plan.

1 Introduction

The health care service system in Germany and many other countries is facing increasing costs due to the aging population. In this work in cooperation with a local health care service provider in medium-size town (150,000 inhabitants) in Germany, we focus on the specific field of home health care, i.e., visiting and providing medical services to clients at home. These medical services range from cleaning, personal hygiene to some medical treatments, including blood pressure measuring, medication prescription, injections and so on.

The home health care scheduling can be regarded as a combination of staff rostering problem [10] and a vehicle routing problem (VRP) with time windows [8]. On the one side, one has to assign weekly visits to nurses with sufficient competence, following the preferred

Zhi Yuan and Armin Fügenschuh
Professorship of Applied Mathematics, Department of Mechanical Engineering, Helmut Schmidt University / University of the Federal Armed Forces, Hamburg, Germany
E-mail: {yuanz,fuegenschuh}@hsu-hh.de

days specified by both the nurses and clients, without exceeding the nurse maximum workload. On the other side, the workload or the working time of each nurse depends on how fast one can arrive from one client's home to another client's home within each client's specified time slots on each of their preferred days. Besides, different from most of the existing home health care scheduling problems in the literature, our scheduling task is on a weekly basis. This is because certain medical treatments require a minimum day difference in between. For example, the insulin injections of certain clients need to be given twice per week, with three days difference in between. This inter-visit day difference is also considered in our optimization process.

In this preliminary case study, the optimization objective is to cover all the weekly client visits with minimal operating cost, especially the personnel size, and also to minimize the nurses' total workload by an effective routing procedure. This should be achieved without the loss of the service quality, for example, sufficient treatment duration should be guaranteed, and the client-specified day and time slots should be obeyed as hard constraint.

This weekly based home health care scheduling problem is formulated as an integer linear programming (ILP) model as a multi-commodity flow problem. Besides, a problem specific greedy construction method is developed and a local search procedure is applied to further improve the constructed solution. The parameters of the algorithm are automatically adapted during the algorithm run, to obtain an instance-specific best parameter setting. Furthermore, the solution found by the heuristic approach is also input as an initial upper bound for a commercial ILP solver to speed up the branch-and-bound process. Real-world instances with up to 90 patients and 460 weekly treatments have been used to validate our approach.

The rest of the article is organized as follows. Section 2 provides a brief literature overview on the home health care scheduling problem. This problem will be described in more details in Section 3, and will be formulated as an ILP model in Section 4. The primal heuristics are presented in Section 5. Section 6 is dedicated to the experimental setup and the computational results for the real-world instances from our industrial partner. Finally, Section 7 provides some concluding remarks and discussions for potential future directions.

2 Literature Review

In general, the home health care scheduling problem belongs to a broader class of problems called workforce scheduling and routing problem, see for example [5] for a literature survey. Such problems can be regarded as a combination of staff rostering problem [10] and a vehicle routing problem (VRP) with time windows [8]. Particularly from the staff rostering aspect, it has many things in common with the nurse rostering and scheduling problem [6, 4] in terms of skill category, shift type, and time related constraints. However, the home health care scheduling problem has the further requirement of a routing task from patient to patient. There exists also works that focus only on the routing part of the home health care scheduling problem, for example, Kergosien et al. [14] modelled it into a multiple traveling salesman problem, and solved the resulting integer linear programming model by a branch-and-cut approach.

Begur et al. [2] presented the use of a spatial decision support system to schedule and route home health care nurses in Birmingham, Alabama, USA. The system integrates geographic information system with scheduling heuristics and databases, and it is reported an over 20,000 US Dollars saving annually for travel expenses and scheduling preparation, and

it helps improve the balance of work among nurses. A saving-type route-building heuristic is proposed and described, and a route improvement is done manually through a visual interactive system.

Cheng and Rich [7] formulated the home health care scheduling problem as a multiple depot vehicle routing problem with time windows. Both full-time and part-time nurses are considered, and each nurse starts and ends their daily service from their home. The lunch break problem is considered by adding an additional lunch node into the scheduling graph. Two mixed integer programming models were presented, and were reported to be able to solve by CPLEX problems of size up to 10 patients. A two-phase heuristic is also developed, using a randomized greedy algorithm to construct a possibly infeasible solution in the first phase, and then improve it by a problem-specific local search in the second phase.

Eveborn et al. [11] introduced a decision support software, developed to aid the staff planner creating daily schedules at a home health care organization in Sweden. Various practical constraints such as staff competence, time windows for visits, or breaks for meals were considered. A mixed-integer programming formulation is given, based on a set partitioning model. As solution method, they make use of repeated matching approach, which maps the staffs-to-home-visits problem into a matching problem, and then solves the resulting matching problem with an exact method or a repeated assignment heuristic. They estimate a total cost saving of 20%, and a reduction of 7% of the total working time.

Rasmussen et al. [22] further presented the home care crew scheduling problem in Denmark. The scheduling is also on a daily basis with temporal dependencies and some other service oriented constraints. One example of such temporal dependencies given by the authors is this: A first home carer switches on a washing machine, and a second home carer should come and empty the washing machine after two to four hours. In such case, the routing part of the problem amounts to the VRP with coupled time windows [13]. The other service oriented constraints include, for example, leaving as few visits uncovered as possible. Each visit is associated with a priority, and if some visits may have to be rescheduled or cancelled, it should not cancel the most important visits of the day. The problem is formulated as a set partitioning problem and a branch-and-price algorithm is developed for its solution.

Koeleman et al. [16] consider a home health care scheduling problem in a stochastic setting. They assume the patients arrive according to a Poisson distribution, and the probability distribution of to which class the patients belong to, and their health care duration per week, is assumed to be known. Then the decision has to be made whether they should accept, reject, or put the patient in a waiting list. This problem is modelled as a Markov decision process, and the (near-)optimal policy can be found by a trunk reservation heuristic. However, no timetabling and routing of the nurses to patients have been considered.

Bertels and Fahle [3] also stated that the home health care scheduling problem is a combination of a staff rostering and a vehicle routing problem. They developed also a hybrid algorithm that consists of linear programming, constraint programming and metaheuristics for solving the problem.

Most of the existing literature as listed above schedules home health care on a daily basis. There exists only few works that schedules in a multi-day horizon, including Nickel et al. [21] and Di Gaspero et al. [9]. Both works formulated the problem as constraint programming (CP) models, and then developed large neighborhood search metaheuristics upon the CP models. The main difference from the application perspective between the both works is that Nickel et al. [21] required the same patients to be visited by the same nurse, while this requirement is relaxed in [9].

To the best of our knowledge, none of the presented models are completely applicable to our problem, mainly for some or all of the following reasons. Our schedule is on a weekly basis, and the inter-visit day difference should be integrated into the optimization process [23]. In the literature, e.g. [2], a two-visit-per-week patient will be fixed to a Monday and Thursday or a Tuesday and Friday before the optimization process starts. Although such fixed assignments simplifies the problem, it also reduced scheduling flexibility and efficiency. In this work, we model inter-visit day difference as a constraint, and let the optimization process to determine the treatment day. Besides, not only the daily maximum working time but also the total weekly maximum working time can be imposed for each nurse. Furthermore, each nurse as well as the patient should be able to specify their individual preference on working days in advance.

3 Problem Description

For notational simplicity we will refer to the employees of the service as *nurses*, the clients as *patients*, the service activities that the clients require as *treatments*. Our task is to develop computer software to assist the health care provider to generate a nurse schedule on a weekly basis.

The optimization task is to assign each patient or treatment to a nurse with competent qualification on an appropriate day at a time within a patient-specified time window.

Patients and treatments. A patient may require a number of nurse visitations, or treatments, in a week. A treatment is a medical activity to be performed by a visiting nurse at a proper time on an appropriate day. These treatments include washing, cleaning, bandage changing, medication prescription, and giving injections. The duration of each treatment is fixed and given in advance, but the start time of each visit can be varied within a patient-specified time window.

Day preference and time windows. Each patient can specify which days are preferred for his weekly treatments, e.g., only Monday and Tuesday, or no treatments on Thursday, etc. Furthermore, on each of these preferred days, a time window within which treatment can start can be specified by the patient, e.g., the cleaning can start on Monday from 9:00 to 11:00 or Thursday from 8:00 to 9:00. Note that the time windows imposed here are hard constraints, which means that if the nurse is scheduled to arrive earlier than the given lower bound of the time window, he or she would have to wait.

Inter-visit day difference. For some patients who need several visits per week, there may be some pairs of visits between which at least one to three days' difference must be retained. Examples of such pairs of treatments include some injections that must be given twice a week with at least three days break in between.

Nurses. There are mainly two types of nurses distinguished by our application partner, namely, the professional nurse and the assistant nurse, differing in qualification and also salary cost. Becoming a professional nurse in Germany requires three years' education, and one must pass the national exam to obtain the medical certificate. These nurses are able to perform all types of treatments. The assistant nurse usually receives a short-term on-the-job training. They can perform simple treatments such as cleaning or changes of bandages, etc. However, they are not allowed to perform treatments such as prescriptions or injections. The assistant nurses' salary is also lower than the professional nurses' salary, and it is estimated to be 70% of what the professional nurses receive.

Working date and time. Nurses' everyday work starts at 7 a.m. in the headquarter of the organization with a car, and the car must return to the headquarter by the end of the

day's work. According to the organization's legal regulation, the daily working hour of each nurse cannot exceed 6 hours, while the total weekly working hours for each nurse cannot be over 30 hours. Each nurse can also choose on which day (normally working days from Monday to Friday) and at most how many hours she is willing to work on a day or within a week. Working time includes the treatment time that a nurse spends at a patient, the driving time between patients and from or to the headquarter, as well as the waiting time, if she arrives at a patient too early.

Goals and objectives. In this work, our objective is to reduce the operating cost and the nurses' workload without compromising the service quality, e.g., the treatment duration, patient-specified date and time and certain required treatment qualification need to be kept. Our primary optimization goal is to reduce the personnel cost, i.e., to use as few staff as possible to carry out all the tasks. The secondary optimization goal is to reduce the total working time of all nurses.

4 Mathematical Models

We formulated the home health care scheduling problem as an integer linear programming (ILP) model based on the multi-commodity flow model with the Miller-Tucker-Zemlin constraints to handle the time consistency [19].

4.1 A graphical example

A graphical example of the home health care scheduling problem is shown in Figure 1. Each depot node is depicted as a triangle, $d^{i,j}$, and represents a nurse i on a day j. Each circle represents a treatment node. Here, nurse 1 carries out treatments t_1, t_2, and t_5 on day 1, and carries out treatment t_4 on day 2. Nurse 2 has treatments t_6 and t_3 on day 1, and has a free day on day 2. We also follow the terms used in multi-commodity flow, meaning that the arcs from a depot node to a treatment node, e.g., from $d^{1,1}$ to t_1, are called *pull-out arcs*, the arcs between two treatment nodes, e.g., t_1 to t_2, are called *deadhead arcs*, and the arcs from a treatment node to depot node, e.g., t_5 to $d^{1,1}$, are called *pull-in arcs*.

4.2 Sets

We denote the set of all nodes with N. There are two types of nodes in the example shown in Figure 1, namely, depot nodes and treatment nodes.

- N_{dp}: the set of depot nodes where a nurse starts and ends his or her day at;
- N_{trm}: the set of treatment nodes that should be visited by one of the nurses.

Similarly, we denote the set of all arcs with A, and distinguish three types of arcs existing in our graphical example, namely, pull-out arcs, deadhead arcs, and pull-in arcs.

- $A_{pullout}$: a pull-out arc starts from a depot node and ends at a treatment node;
- $A_{deadhead}$: a deadhead arc starts and ends both at a treatment node, representing a trip from one patient to another patient;
- A_{pullin}: a pull-in arc starts from a treatment node and ends at a depot node;

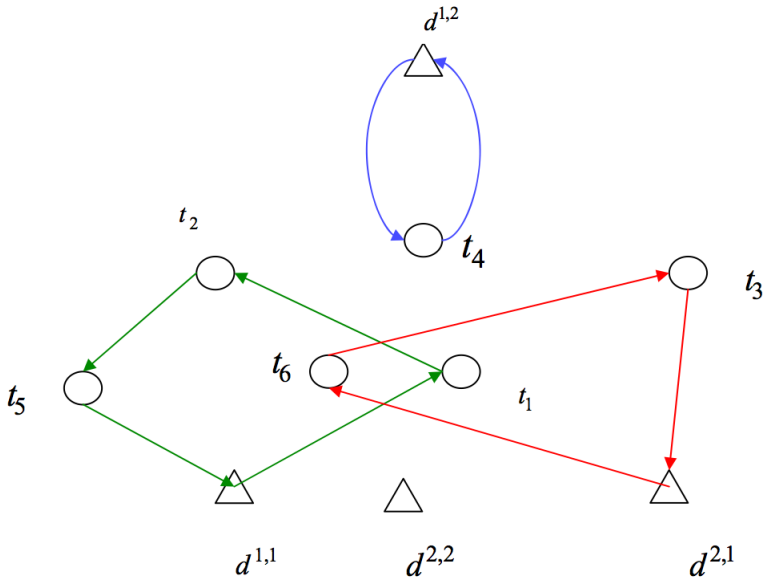


Fig. 1 A graphic example of the home health care problem. Triangles represent depot nodes, where node $d^{i,j}$ represents a nurse i on a day j. Note that some nurse may be totally free on some day. Circles represent treatment nodes with its index. Each tour must start and end at a depot node.

Apart from the sets that are visible in the figure, there are some other sets outside the graph, such as

- K: the set of nurses;
- D: the set of working days, i.e., $D := \{1,\ldots,5\}$, representing Monday to Friday. Note that we consider each nurse per day as a depot in our model;
- $R \subset N_{trm} \times N_{trm}$: the set of pairs of treatments that are related, in the sense that if two treatments i and j are related, $(i,j) \in R$, then treatments i and j must have some days' difference between each other.

4.3 Parameters

The parameters used in the ILP model are listed below:

- c^k_{start}: the starting cost of each nurse $k \in K$.
- $u^{k,d}_{i,j}$: the upper bound for the flow capacity on each arc $(i,j) \in A$ for a nurse $k \in K$ on a day $d \in D$. The value of u is binary, since each treatment node must be visited once and only once, and each depot can deploy only one unit of flow. Besides, we also use this parameter to remove some infeasible arcs in preprocessing, e.g.,
  1. if a treatment $i \in N_{trm}$ cannot be taken by a nurse $k \in K$, then we set all the arcs that are incident to node i to be infeasible by assigning their upper bounds to 0, i.e., $u^{k,d}_{i,j} := 0, u^{k,d}_{j,i} := 0, \forall j \in N, d \in D$.
  2. If a treatment $i \in N_{trm}$ cannot be taken on a day $d \in D$, similarly we set the upper bounds of all arcs incident to node i on day d to 0.
  3. If a nurse $k \in K$ is not possible to work on day $d \in D$, then all arcs on this layer are set to infeasible, $u^{k,d}_{i,j} := 0, \forall (i,j) \in A$.

- $\underline{t}_i$ and $\bar{t}_i$: the lower bound and the upper bound of the time window for a treatment i.
- $\delta_{i,j}$: the necessary duration from the start of treatment i until the start of treatment j, if j is taken by the same nurse on the same day subsequently after i. It consists of two parts, the service time for treatment i, and the trip duration from i to j, i.e., $\delta_{i,j} = \delta_i^{service} + \delta_{i,j}^{deadhead}$.
- $T_{day}^{k,d}$: the daily maximum working time of a nurse k on day d.
- T_{week}^{k}: the weekly maximum working time of nurse k.
- $\sigma_{i,j}$: the necessary day difference between two related treatments $(i,j) \in R$. The value of σ is typically restricted by $1 \leq \sigma \leq 3$.

4.4 Variables, Objective, and Constraints

The decision variables are listed as follows:

- $x_{i,j}^{k,d} \in \{0,1\}$: the flow variable representing whether an arc $(i,j) \in A$ is traveled by a nurse $k \in K$ on day $d \in D$;
- $t_i \in \mathbb{Z}^+$: the starting time of a treatment $i \in N_{trm}$;
- $s^k \in \{0,1\}$: binary variable to indicate whether a nurse $k \in K$ has been deployed.
- $\tau^{k,d} \in \mathbb{Z}_0^+$: the working time of a nurse k on a day d minus the starting time of 420, i.e., 7 a.m. If a nurse on a day does not start any tour, then its value equals to 0.

The objective function consists of two hierarchical parts. Firstly, in this work, the primary objective for the organization is to reduce the personnel cost (without compromising the service level, e.g., service time and nurse qualification). Imprecisely speaking, the goal is to use as few nurses as possible to take care of all the clients. Note that nurses with different qualifications may require different starting cost, then it is the total starting cost to be minimized. Secondly, at a subsidiary level, we wish to shorten the nurse's daily working hours. So the objective is formulated as follows:

$$\min \quad \sum_{k \in K} c_{start}^{k} \cdot s^k + \sum_{k \in K, d \in D} \tau^{k,d}, \tag{1}$$

subject to the following constraints:

- bundle constraint, every treatment will be visited exactly once,

$$\sum_{k \in K, d \in D, i \in N} x_{i,j}^{k,d} = 1, \qquad \forall j \in N_{trm}, \tag{2}$$

- flow capacity constraint, the flow cannot exceed the upper bound on each arc,

$$x_{i,j}^{k,d} \leq u_{i,j}^{k,d}, \qquad \forall (i,j) \in A, k \in K, d \in D, \tag{3}$$

- flow conservation constraint, the inflow of each treatment node should equals its outflow,

$$\sum_{j \in N} x_{j,i}^{k,d} = \sum_{j \in N} x_{i,j}^{k,d}, \qquad \forall i \in N_{trm}, k \in K, d \in D, \tag{4}$$

- time window constraint, which the starting time of each treatment should obey,

$$\underline{t}_i \leq t_i \leq \bar{t}_i, \qquad \forall i \in N_{trm}, \tag{5}$$

– time compatibility constraint, i.e., the starting time difference between two consecutive nodes should not be smaller than its necessary amount, for the three types of arcs, namely,
  1. deadhead arcs:

$$t_i + \delta_{i,j} + M \cdot (x_{i,j}^{k,d} - 1) \leq t_j, \qquad \forall (i,j) \in A_{deadhead}, k \in K, d \in D, \tag{6}$$

  with sufficiently large value for M;
  2. pull-out arcs:

$$420 + \delta_{i,j} + M \cdot (x_{i,j}^{k,d} - 1) \leq t_j, \qquad \forall (i,j) \in A_{pullout}, k \in K, d \in D, \tag{7}$$

  where every nurse is supposed to start their daily work at 7 am (420 minutes after 0:00); and
  3. pull-in arcs:

$$t_i + \delta_{i,j} + M \cdot (x_{i,j}^{k,d} - 1) - 420 \leq \tau^{k,d}, \qquad \forall (i,j) \in A_{pullin}, k \in K, d \in D, \tag{8}$$

  where the daily working time $\tau^{k,d}$ of a nurse k on a day d is measured by the difference between the ending time of her work at the depot and the starting time of her work at 7 a.m.;
– the maximum daily working time should not be exceeded,

$$\tau^{k,d} \leq T_{day}^{k,d}, \qquad \forall k \in K, d \in D, \tag{9}$$

– the maximum weekly working time should not be exceeded,

$$\tau^{k} \leq T_{week}^{k}, \qquad \forall k \in K, \tag{10}$$

– the nurse deployment indicator s^k is determined by checking each pull-out arc,

$$s^k \geq \sum_{(i,j) \in A_{pullout}, d \in D} x_{i,j}^{k,d}, \qquad \forall k \in K, \tag{11}$$

– and the necessary day difference between a pair of related treatments is modeled in two steps:
  1. the former treatment i should not start too late so that the latter treatment j can be visited during the week, i.e., i should start latest on the day of $(5 - \sigma_{i,j})$:

$$\sum_{h \in N, k \in K, d \in \{1,\ldots,5-\sigma_{i,j}\}} x_{h,i}^{k,d} = 1, \qquad \forall (i,j) \in R, \tag{12}$$

  2. then the latter treatment j should be started no earlier than the day $d' \geq d + \sigma_{i,j}$,

$$\sum_{h \in N, k \in K} x_{h,i}^{k,d} + \sum_{h' \in N, k' \in K, d' \in \{1,\ldots,d+\sigma_{i,j}-1\}} x_{h',i}^{k',d'} \leq 1, \qquad \forall (i,j) \in R, d \in \{1,\ldots,5-\sigma_{i,j}\}. \tag{13}$$

To sum up, the home health care scheduling problem can be formulated as follows:

$$\begin{aligned} &\text{minimize} && (1), \\ &\text{subject to} && (2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13), \\ &&& x \in \{0,1\}^{|A| \times |K| \times |D|}, t \in \mathbb{Z}_+^{|N_{trm}|}, s \in \{0,1\}^{|K|}, \tau \in \mathbb{Z}_0^{+\,|K| \times |D|}. \end{aligned}$$

5 Primal Heuristics

Our primal heuristics include a tailored greedy construction, different local search operators, and a PGreedy mechanism for automatic adaptation of the greedy parameters.

5.1 Greedy Construction Heuristic

The greedy construction heuristic builds a complete solution from scratch, by iteratively choosing an immediate best solution component, until a complete solution is generated. There are two greedy decisions to be made during the construction: choosing a nurse to start, and choosing a next treatment node to append.

There are two types of nurses. An assistant nurse can handle only a subset of "simple" treatments and is less expensive than professional nurses. If there are only very few (less than 20) simple treatments left, we will always start with the professional nurse. Otherwise, given the proportion of simple treatments as P_s, we select an assistant nurse by the probability $P_{assist} := P_s \cdot \frac{c_{prof}}{c_{assist}}$, where c_{prof} and c_{assist} refer to the cost of a professional and an assistant nurse, respectively. Within the same nurse type, we start with the nurse with the most maximum weekly working time.

After the nurse is chosen, all his or her allowed working days are randomized. Then for each day, we first filter the treatments that are feasible for that day, and then build the schedule by iteratively appending the next best treatment node. For selecting the next treatment, the following factors are considered:

Link time. $\Delta t_j := \min\{\delta_{i,j}, \underline{t}_j - t^{current}\}$, i.e., the earliest possible starting time of the next node j minus the current time;

Time window size. $t_j^{day} := \bar{t}_j - \max\{\underline{t}_j, t^{current} + \delta_{i,j}\}$, i.e., how flexible the treatment can start during the day, since the treatments that are less flexible may be more preferred during the construction, while the more flexible treatment may be treated at a later time;

Time window in other days. t_j^{other}, which is the sum of the time window size of any other days. The same rationale above applies here: if a treatment can be only taken on this day, it should have a higher priority than the ones that are more flexible.

Then the greedy best next node j^* is selected as follows:

$$j^* = \arg\min_{i \in N_{feasible}} \quad \Delta t_j + \alpha \cdot t_j^{day} + \beta \cdot t_j^{other}, \tag{14}$$

where the two greedy weighting parameters α and β are determined dynamically by a PGreedy approach described below.

5.2 Local Search

We have applied two types of local search operators: a *node insertion* and a *nurse type exchange* operator.

Node insertion. Since our primary objective is to reduce the number of nurses, this is also the goal for the node insertion. We first sort all the nurses in the constructed schedule by the number of treatments handled. Then we iteratively try to insert a treatment node from the least-loaded nurse to all the nurses that are more-loaded. The best-improvement

strategy is applied, i.e., if there are multiple positions with multiple nurses where a treatment can be inserted, then the one with the least additional working hours is chosen. Ties are broken randomly. Note that after a node has been inserted, the corresponding time windows of the treatment nodes have to be updated by a constraint propagation. After all the nodes from the least-loaded nurse have been tried, then we resort all the nurses except the least-loaded nurse, and we start from the second least-loaded nurse, and repeat inserting all its nodes to more-loaded nurses as described above. Once a node is inserted to the schedule of another nurse, we check immediately whether the least-loaded nurse can insert its nodes to the nurse with a decremented node. This process ends, when all the nurses but the most-loaded one have been tried to insert their nodes. Note that this node insertion local search does not necessarily reduce the overall objective value, but its goal is mainly to try to reduce the number of nurses. It happens in some cases that the objective value may increase after the node insertion operation, since the total working time increases. However, it is very effective in reducing the number of nurses.

Nurse type exchange. After the node insertion operation, a nurse type exchange is performed. It starts by checking all the treatments of each professional nurse, if all the treatments taken by a professional nurse can be taken by an assistant nurse, then an assistant nurse is deployed instead.

5.3 Online Parameter Adaptation

The best greedy criterion is usually unknown in advance, besides, each instance may have a different best greedy criterion. The weighting parameters in the greedy construction, α and β can be adapted while running the algorithm. This is done by following the procedure described as Parameterized Greedy (PGreedy) [12]. The PGreedy algorithm can find the best greedy criterion while running on a particular instance, by using a black-box optimization algorithm to search the greedy parameter space. One example of such black-box search algorithm proposed in [12], and adopted in our work, is the improving hit-and run (IHR) procedure [24].

The heuristic procedure works as follows. The improving hit-and-run is applied to adapt the best-so-far greedy parameter setting to start each iteration. In each iteration, it runs a greedy construction (Section 5.1) with the adapted parameter setting, followed by a local search (Section 5.2). If a new best-so-far solution is found, we also update the best-so-far greedy parameter setting to the new one as in the IHR procedure. It leaves to define the range of the parameters. In our preliminary experiments, α is set to $[0, 0.2]$, and β is set to $[0, 0.1]$.

6 Experiments and Results

The primal heuristics are implemented in Java. The code is executed on a personal computer with AMD AthlonXP CPU at 1.5 GHz and 256 MB DDR RAM, running Windows XP as operating system. The mathematical model is formulated using the Zimpl language [15], and the Zimpl generated ILP model is then solved by ILOG CPLEX 10. Zimpl and CPLEX ran on a computing server with 32 GB RAM and 8 $\times$ 2.4 GHz AMD Opteron 880 CPU, running SUSE 10.2 Linux OS.

Table 1 The instances used in our case study, extracted from a current real-world home care schedule. The entire instance has 99 patients, 460 treatments, and 9 nurses. Smaller instances were extracted, including from 22 to 285 treatments. Below it lists the size of the problems including the number of patients, treatments, nurses and related pairs, as well as the size of the ILP models built by Zimpl, including the number of variables, constraints and non-zero coeffecients.

Instance	Problem size				ILP model size		
	#patient	#treatment	#nurse	#related-pair	#variable	#constr	#non-zero
`nurse22`	5	22	2	0	5.3K	8.4K	4.3K
`nurse75`	15	75	3	4	87K	99K	124K
`nurse110`	25	110	3	3	185K	218K	202K
`nurse153`	33	153	4	5	474K	542K	583K
`nurse210`	46	210	4	10	891K	962K	1.4M
`nurse285`	60	285	6	9	2.1M	2.1M	3.0M
`nurse460`	99	460	9	17	8.5M	8.9M	69.8M

6.1 The Instances

One data set is available from our application partner, a local home health care service provider in the whole town area. This instance is a real-world weekly nurse schedule currently in use. It contains 99 patients, 460 treatments, and 9 nurses, and is named `nurse460`. The number of pairs with inter-visit day difference is already greatly reduced in our dataset preprocessing phase, and 17 pairs are still left. Each treatment has a patient-specific time window, which is given in advance. The size of the time window varies greatly, ranging from 30 minutes to 360 minutes. The resulting ILP model from Zimpl consists of over 8 millions variables, 9 millions constraints, and 70 millions non-zeros, as listed in the last row of Table 1. CPLEX is not able to even finish one run of the simplex algorithm for the LP relaxation model within our computation time limit, which is set to 6 hours.

In order to better study the problem scalability, we further extract smaller instances from the real-world instance `nurse460`. These instances were extracted by randomly selecting a number of patients, and then including all the treatments and related pairs of the selected patients. The size of the extracted instances ranges from 5 patients with 22 treatments to 60 patients with 285 treatments. The number of nurses has an important influence on the computational difficulty of the ILP model, since each individual nurse determines a specific commodity layer in our multi-commodity network, hence the size of the network model is proportional to the number of nurses. For more details on this issue, we refer to [23]. Therefore, we tried to minimize the number of nurses for the extracted instances by taking the number of nurses of the best found heuristic solution. The size of the ILP model built by Zimpl, including the number of variables, constraints, and non-zero coefficients for each instance is also presented in Table 1.

6.2 Computational Results of the Primal Heuristics

For each of the extracted subinstances, 10 trials of the heuristic are run, and each trial was allowed 30 seconds CPU time. We reported the best solution of the 10 trials in Table 2, and reported its overall computation time as $30 \times 10 = 300$ seconds.

It is worth mentioning that for the real-world instance `nurse460` a good scheduling solution can be found by our heuristic algorithm within 5 minutes. This schedule requires only eight nurses, which improves the original manual nurse schedule currently in use by

Table 2 The computational results of the primal heuristics. Reported is the best solution out of 10 runs of 30 CPU seconds each. The objective value, total number of nurses and number of nurses of the two different types (professional or assistant), total working time and the average working time per nurse are listed.

Instance	Comp. Time	Obj. Value	Nurses (prof. / asst.)	Total #nurse	Total Work Time	Average Work Time
nurse22	300	1702200	1 / 1	2	2200	1100
nurse75	300	2702071	2 / 1	3	2071	690
nurse110	300	2704577	2 / 1	3	4577	1526
nurse153	300	3705336	3 / 1	4	5336	1334
nurse210	300	3706175	3 / 1	4	6175	1544
nurse285	300	5707509	5 / 1	6	7509	1252
nurse460	300	7713293	7 / 1	8	13293	1662

Table 3 The computational results of the commercial ILP solver. Two LP relaxation strategies were used: simplex and interior point method. The computation time in seconds, the best integer solution (upper bound), best node (lower bound) and the gap is reported.

Instance	Simplex				Interior Point			
	Comp. Time	Best integer	Best node	Gap	Comp. Time	Best integer	Best node	Gap
nurse22	33	1702200	1702200	0%	27	1702200	1702200	0%
nurse75	21600	-	1003438	-	21600	2702103	2000098	26%
nurse110	21600	-	1000359	-	21600	-	1000359	-
nurse153	21600	-	1000000	-	21600	-	1333692	-
nurse210	21600	-	700633	-	21600	-	701800	-
nurse285	21600	-	0	-	21600	-	1000000	-
nurse460	21600	-	0	-	21600	-	0	-

reducing one nurse. It would be also interesting to compare the secondary optimization goal, the total working time, with the manual schedule, but unfortunately, this information is not available from our application partner.

6.3 Computational Results of the ILP Solver

The commercial ILP solver CPLEX is also used to solve the model of the instances presented above. Two different strategies were applied to solve the subsidiary LP relaxation in the branch-and-bound procedure: the default simplex method and the interior point method.

As shown in Table 3, the solver CPLEX has difficulty in solving the ILP model. Only the smallest instance `nurse22` can be solved to optimality. The interior point method appears to be a better alternative for solving the LP relaxation problem by providing better lower bounds compared to the default simplex method. Therefore, the interior point method is used instead of simplex for solving LP relaxation. For instances that are larger than 75 treatments, no feasible integer solution was found. In such case, the branch-and-bound procedure cannot prune any subtree, and it results in an exhaustive search. Therefore, the heuristic approach is essential for solving this problem. Not only can the heuristic provide feasible solutions, it may also be used as an initial upper bound to help the branch-and-bound procedure to prune provable inferior subtrees.

Table 4 The computational results of initializing the ILP solver CPLEX by heuristic solution. The second and third column presents the best heuristic solution and its gap with respect to the lower bound calculated in Table 3; the fourth to seventh column shows the computation time (in seconds), best integer (objective value), best node (lower bound), and the root gap of the ILP solver with initialization of the heuristic solution; the eighth and ninth column shows the number of nurses, and the total working minutes (and the reduced minutes with respect to best heuristic solution).

Instance	Primal heuristic		ILP solver with heuristic initinalization				Final Solution	
	Obj. Value	Gap	Comp. Time	Best integer	Best node	Gap	#Nurse	Work Time (diff.)
nurse22	1702200	0%	30	1702200	1702200	0%	2	2200 (0)
nurse75	2702071	26%	21600	2702001	2000144	26.0%	3	2001 (3.4%)
nurse110	2704577	63%	21600	2704525	1000359	63%	3	4525 (1.1%)
nurse153	3705336	64%	21600	3705336	1500359	60%	4	5336 (0)
nurse210	3706175	81%	21600	3706175	701800	81%	4	6175 (0)
nurse285	5707509	82%	21600	5707509	1000000	82%	6	7509 (0)
nurse460	7713293	100%	21600	7713293	0	100%	8	13293 (0)

6.4 Initialize ILP Solver by Heuristic Solution

As indicated in the previous section, although the ILP solver CPLEX is equipped with various heuristic approaches for finding feasible solution, it is unable to find feasible solution within our computation time limit. One possible speed-up is to provide the best heuristic solution as an initial feasible solution for the ILP solver. The advantage of doing so is twofold: on the one side it may help the branch-and-bound procedure to reach more nodes by discarding some provable inferior subtrees; and on the other hand, the ILP solver may help further improve the heuristic solution.

The results are shown in Table 4. Comparing to the lower bound listed Table 3, the lower bound in `nurse75` and `nurse153` is improved. Especially the lower bound improvement of `nurse153` leads to over 4% improvement of the root gap. From the upper bound perspective, although the number of nurses calculated by the heuristic is not further improved, we observe that the total working time in `nurse75` and `nurse110` has been improved by 3.4% and 1.1%, respectively.

7 Conclusions and Future Works

The current work presented a preliminary case study of the home health care scheduling problem for a local health care provider in Germany. At the current stage, it is important to evaluate the scalability of different solution approaches, as well as the potential cost saving for the organization. In this work, we formulated the weekly based home health care scheduling problem as an integer linear programming model, and applied a state-of-the art ILP solver to it. Unfortunately, the current model is only able to be solved optimally up to a tiny instance size. The heuristics on the other hand perform very well, and are able to obtain good solutions within five minutes. Initializing the branch-and-bound process with a heuristic solution seems to speed up the solving process, but the gap for a real-world size instance is still vast. The results also indicate at least 10% cost saving potential for using a computerized optimization process.

As this is still an ongoing work, there are many directions that the current work can be extended. The future directions can be categorized into two aspects, methodology and

application. From the methodological aspect, although this work shows that local search approaches are more preferable for the real-world size problem than the exact approaches, it is still interesting to obtain a provable optimality or a lower bound to assess the local search approaches. A potential direction is to reformulate the current mathematical model into a set partitioning problem and solve it by a branch-and-price approach [1], since the branch-and-price approaches have proven to be successful for solving the VRP with time windows [8]. As for the heuristic approach, it will be interesting to further explore the more effective local search techniques. Our current node insertion operator is mainly aimed at reducing personnel size. It will be interesting to start a second phase of the local search aiming at also reducing the total working time, and rebalancing the workload among different nurses. To this end, local search operators such as node exchange and swapping can be also effective. In fact, it will also be interesting to iterate these local search operators in a variable neighborhood descent fashion [20], so that the local optimum found is the local optimum with respect to all the local search operators. Besides, instead of restarting the heuristic in each iteration from scratch, it will be more effective to perturb a small part from the best-so-far solution, and then perform local search afterwards without a full construction, as done in the iterated local search [18].

From the application aspect, currently we consider mainly minimizing operating cost as the optimization objectives. In fact, there are still other objectives that our application partner cares about other than costs, such as follows.

Workload balancing. Each nurse should have more or less the same amount of workload, such that each nurse has a similar number of patients or treatments to handle. The current schedules generated by the node insertion local search may cause certain nurses with very heavy workload while some other nurses have only very few treatments to handle. This can be done by adding a balancing objectives into the mathematical model, and by also applying node insertion to insert treatments from heavily-loaded nurses to less-loaded nurses.

Dynamic reallocation. The scheduling tool should be robust in the presence of changes. If any changes happen, the reallocation of a new schedule should be made in a short time with as few modifications to the original schedule as possible. There are various possible changes, for instances, new clients joining in, clients changing their preferred visitation dates or time windows, or nurse staff's availability. Some of these changes are known one week before, but in some urgent cases, which are not rare, patients may call in a short time to change the visit appointment to another time, nurses may call in sick shortly before the schedule starts, or a damage to their car suddenly occurs. In such cases, especially the last minute cases, a new schedule should be generated in a short time, with as few changes to the original schedule as possible. In such case, a local search procedure is important.

Consistent nurse. One important feature to improve the service quality is to always assign the same nurse to the same patient's treatments. From the nurse's perspective, it is easier to catch up with the patient's situation, while from the client's perspective, they also desire more consistency. Keeping the nurse consistency may reduce scheduling flexibility, and may result in higher operating cost or longer working time. Technically, each node insertion will need to consider inserting a set of treatments of the same patient instead of inserting one treatment node. To this end, Kovacs et al. [17] also provides a nice survey of vehicle routing problems with consistency considerations.

Acknowledgements This work is supported by BMBF Verbundprojekt E-Motion.

References

1. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.
2. S. V. Begur, D. M. Miller, and J. R. Weaver. An integrated spatial DSS for scheduling and routing home-health-care nurses. *Interfaces*, 27(4):35–48, 1997.
3. S. Bertels and T. Fahle. A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers & Operations Research*, 33(10):2866–2890, 2006.
4. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.
5. J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, pages 1–29, 2014.
6. B. Cheang, H. Li, A. Lim, and B. Rodrigues. Nurse rostering problems – a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003.
7. E. Cheng and J. L. Rich. A home health care routing and scheduling problem. Technical report, Technical Report TR98-04, Department of CAAM, Rice University, 1998.
8. J. F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. VRP with Time Windows. In P. Toth and D. Vigo, editors, *The vehicle routing problem*, pages 157–193. Society for Industrial and Applied Mathematics, 2001.
9. L. Di Gaspero and T. Urli. A CP/LNS approach for multi-day homecare scheduling problems. In M. J. Blesa et al., editors, *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 1–15. Springer International Publishing, 2014.
10. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.
11. P. Eveborn, P. Flisberg, and M. Rönnqvist. LAPS CARE – an operational system for staff planning of home care. *European Journal of Operational Research*, 171(3):962–976, 2006.
12. A. Fügenschuh. Parametrized greedy heuristics in theory and practice. In *Proceedings of Hybrid Metaheuristics*, volume 3636 of *Lecture Notes in Computer Science*, pages 21–31. Springer, 2005.
13. A. Fügenschuh. The vehicle routing problem with coupled time windows. *Central European Journal of Operations Research*, 14(2):157–176, 2006.
14. Y. Kergosien, C. Lenté, and J.C. Billaut. Home health care problem: An extended multiple traveling salesman problem. In *Proceedings of 4th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, 2009. 8 pages.
15. T. Koch. *Rapid Mathematical Prototyping*. PhD thesis, Technische Universität Berlin, 2004.
16. P. M. Koeleman, S. Bhulai, and M. Van Meersbergen. Optimal patient and personnel scheduling policies for care-at-home service facilities. *European Journal of Operational Research*, 219(3):557–563, 2012.
17. A. A. Kovacs, B. L. Golden, R. F. Hartl, and S. N. Parragh. Vehicle routing problems in which consistency considerations are important: A survey. *Networks*, 64(3):192–213, 2014.
18. H. Lourenço, O. Martin, and T. Stützle. Iterated local search. *Handbook of metaheuristics*, pages 320–353, 2003.
19. C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.
20. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.
21. S. Nickel, M. Schröder, and J. Steeg. Mid-term and short-term planning support for home health care services. *European Journal of Operational Research*, 219(3):574–587, 2012.
22. M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219(3):598–610, 2012.
23. Z. Yuan. Solving Real-World Vehicle Routing Problems Using MILP and PGreedy Heuristics, Diplomarbeit, Technische Universität Darmstadt, 2007.
24. Z. B. Zabinsky, R. L. Smith, J. F. McDonald, H. E. Romeijn, and D. E. Kaufman. Improving hit-and-run for global optimization. *Journal of Global Optimization*, 3(2):171–192, 1993.

MISTA 2015

A biased random-key genetic algorithm for scheduling divisible loads

Julliany S. Brandão · Thiago F. Noronha · Mauricio G. C. Resende · Celso C. Ribeiro

Abstract A divisible load is an amount $W \geq 0$ of computational work that can be arbitrarily divided into chunks and distributed among a set P of worker processors to be processed in parallel. The Divisible Load Scheduling Problem consists in (a) selecting a subset of active workers, (b) defining the order in which the chunks will be transmitted to each of them, and (c) deciding the amount of load α_i that will be transmitted to each active worker, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. We propose a biased random-key genetic algorithm for solving the divisible load scheduling problem. Computational results show that the genetic algorithm outperforms the best heuristic in the literature.

Keywords Divisible load scheduling, Random-key genetic algorithms, parallel processing, scientific computing

Julliany S. Brandão
Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil, and Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Av. Maracanã 229, Rio de Janeiro, RJ 20271-110, Brazil
E-mail: jbrandao@ic.uff.br

Thiago F. Noronha
Universidade Federal de Minas Gerais, Avenida Antônio Carlos 6627, Belo Horizonte, MG 24105, Brazil
E-mail: tfn@dcc.ufmg.br

Mauricio G. C. Resende
Mathematical Optimization and Planning, Amazon.com
333 Boren Avenue North, Seattle, WA 98109, USA (work of this author was done when he was employed by AT&T Labs Research)
E-mail: resendem@amazon.com

Celso C. Ribeiro
Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
E-mail: celso@ic.uff.br

1 Introduction

A *divisible load* is an amount $W \geq 0$ of computational work that can be arbitrarily divided and distributed among different processors to be processed in parallel. The processors are arranged in a star topology and the load is stored in a central *master* processor. The latter splits the load into chunks of arbitrary sizes and transmits each of them to other processors, called *workers*. The master does not process the load itself.

The master can only send load to one worker at a time. Any worker can only start processing after it has completely received its respective chunk of the load. The workers are heterogeneous in terms of processing power, communication speed, and setup time for start communicating with the master. Not all available processors should necessarily be used for processing the load.

The Divisible Loading Scheduling Problem (DLSP) was introduced in [12], motivated by an application in intelligent sensor networks. Applications of DLSP arise from a number of scientific problems, such as parallel database searching [11], parallel image processing [25], parallel video encoding [26,35], processing of large distributed files [37], and task scheduling in cloud computing [28], among others.

In this work, we deal with the same DLSP variant treated in [1, 10]. Let $P = \{1, ..., n\}$ be the set of worker processors indices. Each worker $i \in P$ has (i) a setup time $g_i \geq 0$ to start the communication with the master, (ii) a communication time $G_i \geq 0$ needed to receive each unit of the load from the master and (iii) a processing time $w_i \geq 0$ needed to process each unit of the load. Therefore, it takes $g_i + \alpha_i \cdot G_i$ units of time for the master to transmit a load chunk of size $\alpha_i \geq 0$ to the worker $i \in P$. Furthermore, it takes an additional $w_i \cdot \alpha_i$ units of time for this worker to process the chunk of load assigned to it.

The scheduling problem consists of (a) selecting a subset $A \subseteq P$ of active workers, (b) defining the order in which the chunks will be transmitted to each active worker and (c) deciding the amount of load α_i that will be transmitted to each worker $i \in A$, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. This problem was proved to be NP-hard in [42].

Since the load can be split arbitrarily, all workers stop at the same time in the optimal solution [5]. In addition, if the order in which the chunks are transmitted to the workers is fixed, then the best solution can be computed in $O(n)$ time, using the AlgRap algorithm developed in [1]. In other words, given a permutation of the workers in P, AlgRap computes the set of active workers and the amount of load that has to be sent to each of them to minimize the makespan. Therefore, DLSP can be reduced to the problem of finding the best permutation of the processors, i.e., the one which induces the minimum makespan. In order to find this permutation, we propose a biased random-key genetic algorithm [18,19,22], which has been successfully used for solving many permutation based combinatorial optimization problems [17,19–21,23,30].

The paper is organized as follows. Related work is reviewed in the next section. The proposed heuristic is described in Section 3. Computational experiments are reported and discussed in Section 4. Concluding remarks are drawn in the last section.

2 Related work

There are many variants of DLSP in the literature. Divisible load scheduling may be performed in a *single round* or in *multiple rounds*. In the single round case [1, 3–5, 7, 9, 10, 13, 15, 16, 24, 27, 32, 36, 38, 39], each active worker receives and processes a single chunk of the load. In the multi-round case [1, 4, 6, 8, 14–16, 31, 33, 39–41], each active worker receives and processes multiple chunks of load. After the master finishes the transmission of the first round of load to all active workers, it immediately starts the transmission of the next batch of load chunks following the same order, until all the load is distributed among the workers.

The workers may be *homogeneous* or *heterogeneous*. If the workers are homogeneous, the values of g_i, G_i, and w_i are the same for all processors $i \in P$ [4,8,9,24,40,41]. Contrarily, in the heterogeneous case the values of g_i, G_i, and w_i may be different for each worker [1, 3, 5, 6, 10, 13–16, 27, 31–33, 36, 38–41].

The system can be *dedicated* or *non-dedicated*. When the system is dedicated, it is assumed that all resources (processors, memory, network, etc.) are used to process a single computational load [1, 5–7, 9, 10, 13, 15, 16, 31, 36, 38–41]. Non-dedicated systems may be used to simultaneously process different computational loads [6, 7, 32].

In this work, we consider the single round DLSP with dedicated and heterogeneous workers [1, 4, 5, 10, 15, 38, 42], that was proved to be *NP*-hard in [42]. Blazewicz and Drozdowski [10] showed that once a permutation of the workers is given, a solution with minimum makespan can be obtained in $O(n \log n)$ time, where $n = |P|$ is the number of workers. Later, Abib and Ribeiro [1] proposed the faster AlgRap algorithm that finds this solution in $O(n)$ time.

Non-linear programming formulations for the single round DLSP with dedicated and heterogeneous workers was proposed in [13]. The first mixed integer linear programming formulation was proposed in [4] and was later improved and extended in [1]. Computational experiments reported in [1] have shown that the CPLEX branch-and-cut algorithm based on this formulation was able to find optimal solutions for 490 out of 720 instances with up to 160 processors. However, for the largest unsolved instances, the computational gaps were very high, which motivated the development of heuristics for solving DLSP.

To the best of our knowledge, the best heuristic in the literature for the DLSP variant studied in this paper is the HeuRet algorithm of Abib and Ribeiro [1]. At each iteration, their algorithm (i) estimates the *performance* e_i of each worker in $i \in P$ and (ii) builds a solution by taking the processors in P in a non-increasing order of the e_i values. The algorithm sets $e_i = G_i$, for all $i \in P$, in the first iteration and makes use of the exact AlgRap algorithm to compute an initial solution s_0 with makespan T_0. Next, at each forthcoming iteration k, the estimated performance e_i of each worker $i \in P$ is updated from the values of g_i, w_i, G_i, and T_{k-1} and a new solution with makespan T_k is built by algorithm AlgRap considering the new order defined on the workers. The procedure stops when $T_{k-1} < T_k$, i.e. when the new solution degenerates the makespan of the previous one.

The heuristic proposed in the next section improves upon HeuRet because, instead of defining a greedy local search on the performance estimation of the workers, it performs a global search in the space of worker permutations in order to find the permutation that induces the optimal or a near-optimal solution.

3 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced in [2] for combinatorial optimization problems whose solutions can be represented by a permutation vector. Solutions are represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution vector and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover [19]. In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a non-elite solution. The selection is said biased because one parent is always an elite individual and has a higher probability of passing its genes to the new generation.

The BRKGA-DLS biased random-key genetic algorithm for DLSP evolves a population of chromosomes that consists of vectors of real numbers (keys). Each solution is represented by a $|P|$-vector, in which each component is a real number in the range $[0, 1]$ associated with a worker processor in P. Each solution represented by a chromosome is decoded by a heuristic that receives the vector of keys and builds a feasible solution for DLSP using algorithm AlgRap [1]. Decoding consists of two steps: first, the processors are sorted in a non-decreasing order of their random keys; next, the resulting order is used as the input for AlgRap. The makespan of the solution provided by AlgRap is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in [34] to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability 0.60 and from the least fit parent with probability 0.40. Instead of the mutation operator, the concept of mutants is used: a fixed number of mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population.

The keys associated to each worker are randomly generated in the initial population. At each generation, the population is partitioned into two sets: TOP and $REST$. Consequently, the size of the population is $|TOP| + |REST|$. Subset TOP contains the best solutions in the population. Subset $REST$ is formed by two disjoint subsets: MID and BOT, with subset BOT being formed by the worst elements on the current population. As illustrated in Figure 1, the chromosomes in TOP are simply copied to the population of the next generation. The elements in BOT are replaced by newly created mutants. The remaining elements of the new population are obtained by crossover with one parent randomly chosen from TOP and the other from $REST$. This distinguishes a biased random-key genetic algorithm from the random-key genetic algorithm of Bean [2], where both parents are selected at random from the entire population. Since a parent solution can be chosen for crossover more than once in a given generation, elite solutions have a higher probability of passing their random keys to the next generation. $|MID| = |REST| - |BOT|$ offspring solutions are created. In our implementation, the population size was set to $|TOP| + |MID| + |BOT| = 5 \times |P|$, with the sizes of sets TOP, MID, and BOT set to $0.15 \times 5 \times |P|$, $0.7 \times 5 \times |P|$, and $0.15 \times 5 \times |P|$, respectively, as suggested by Noronha et al. [30].

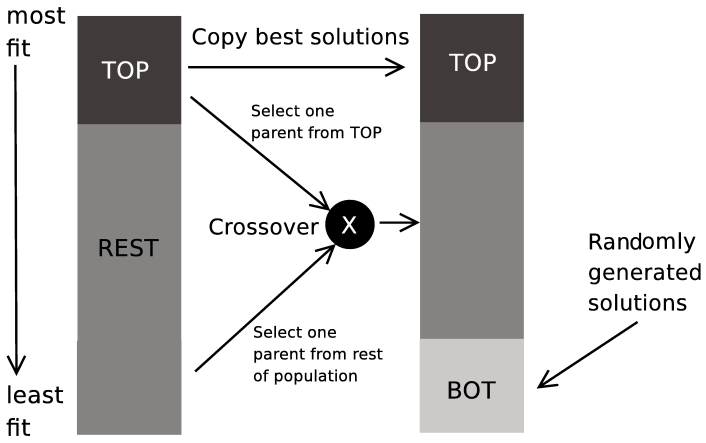


Fig. 1 Population evolution between consecutive generations of a BRKGA.

4 Computational experiments

We report computational experiments to assess the performance of the biased random-key genetic algorithm BRKGA-DLS. This algorithm was implemented in C++. The experiments were performed on a Quad-Core AMD Opteron(tm) Processor 2350, with 16 GB of RAM memory.

The set of instances used in the experiments was proposed in [1]. There are 120 grid configurations with $n = 10, 20, 40, 80, 160$ worker processors and eight combinations of the parameter values g_i, G_i and w_i, $i = 1, \ldots, n$, each of them ranging either in the interval [1, 100] or in the interval [1000, 100000]. Three instances were generated for each combination of n, g_i, G_i, and w_i. Six different values of the load $W = 100, 200, 400, 800, 1600, 3200$ are considered for each grid configuration, corresponding to 18 instances for each combination of parameters g_i, G_i and w_i, and amounting to a total of 720 test instances. Each heuristic was run ten times for each instance, with different seeds for the random number generator of [29].

The first experiment consisted in evaluating if BRKGA-DLS efficiently identifies the relationships between keys and good solutions and converges faster to near-optimal solutions. We compare its performance with that of a multi-start procedure that uses the same decoding heuristic as BRKGA-DLS. Each iteration of the multi-start procedure, called MS-DLS, applies the same decoding heuristic of BRKGA-DLS, but using randomly generated values for the keys. In this experiment, BRKGA-DLS was run for 1000 generations and MS-DLS for $1000 \times q$ iterations, where $q = 5 \times |P|$ is the population size of BRKGA-DLS. The average solution values found by BRKGA-DLS were 4.95% better than those provided by MS-DLS over the 720 instances. Also, the worst (resp. best) solution values found by BRKGA-DLS were 5.98% (resp. 3.78) smaller than the respective worst (resp. best) solution values obtained with MS-DLS. These results indicate that BRKGA-DLS identifies the relationships between keys and good solutions, making the evolutionary process converge to better solutions faster than MS-DLS.

In the second experiment, we compared BRKGA-DLS with HeuRet and MS-DLS. We first evaluated how many optimal solutions have been obtained by each heuristic

over the 720 test instances. The default CPLEX branch-and-cut algorithm based on the formulation in [1] was also run for the 720 instances. Version 12.6 of CPLEX was used and the maximum CPU time was set to 24 hours. CPLEX was able to prove the optimality for 497 out of the 720 test instances. The first line of Table 1 shows that BRKGA-DLS found optimal solutions for 413 instances (i.e. 83.1%) out of the 497 instances for which the optimal solutions are known, while HeuRet found 320 optimal solutions and MS-DLS found only 177 of them. The second line of Table 1 gives the number of instances for which each heuristic found the best known solution value. The third line of this table shows the number of runs for which BRKGA-DLS and MS-DLS found the best known solution values (we recall that HeuRet is a deterministic algorithm, while the others are randomized). Finally, the last line of this table gives, for each of the three heuristics, a score that represents the sum over all instances of the number of methods that found strictly better solutions than the specific heuristic being considered. The lower a score is, the best the corresponding heuristic is. It can be seen that BRKGA outperformed both HeuRet and MS-DLS heuristics with respect to the number of optimal and best solutions found, as well as with respect to the score value. In particular, BRKGA-DLS obtained better scores than HeuRet for all but one instance and found the best known solution values for 645 out of the 720 test instances, while HeuRet found the best solution values for only 313 instances.

Table 1 Summary of the numerical results obtained with BRKGA-DLS, HeuRet and MS-DLS for 720 test instances.

	MS-DLS	HeuRet	BRKGA-DLS
Optimal values (over 497 instances)	177	320	413
Best values (over 720 instances)	189	313	645
Best values (over 7200 runs)	2166	-	6191
Score value	803	112	1

The third and last experiment provides a more detailed comparison between HeuRet and BRKGA-DLS, based on 20 larger and more realistic instances with $|P| = 320$ and $W = 10,000$. The values of G_i and g_i have been randomly generated in the ranges $[1, 100]$and $[100, 100, 000]$, respectively. However, differently from [1], the values of w_i have been randomly generated in the interval $[200, 500]$. These values are more realistic, since the processing rate of a real computer is always larger than its communication rate. In this experiment, BRKGA-DLS was made to stop after $|P|$ generations without improvement in the best solution found. The results are reported in Table 2. The first column shows the instance name. The second and third columns display, respectively, the makespan and the computation time (in seconds) obtained by HeuRet. The next two columns provide the average makespan over ten runs of BRKGA, the corresponding coefficient of variation, defined as the ratio of the standard deviation to the average. The average computation time in seconds of BRKGA over ten runs is given in the sixth column. The last column shows the percent relative reduction between the average solution found by BRKGA-DLS with respect to that found by MS-DLS. It can be seen that the average makespan obtained by BRKGA-DLS is always smaller than that given by HeuRet. In addition, the coefficient of variation of BRKGA-DLS is very small, indicating that it is a robust heuristic. The percent relative reduction of BRKGA-DLS with respect to MS-DLS amounted to 3.19% for instance dls.320.10 and to 2.38% on average.

As in real-life applications the load size may be very large, and the total communication and processing times may take many hours, reductions in the elapsed time of this magnitude may be very significant. Although the running times of BRKGA-DLS are larger than those of HeuRet, their average values never exceeded the time taken by HeuRet by more than 30 seconds. Since practical applications of parallel processing take long elapsed times, the trade-off between the reduction in the elapsed time and this small additional running time needed by BRKGA accounts very favorably to BRKGA-DLS. In addition, we notice that the BRKGA-DLS genetic algorithm itself may be efficiently parallelized with a high speed-up and distributed over the set P of processors, making its computation time irrelevant when compared with that of the parallel application.

Table 2 BRKGA vs. HeuRet on the largest instances with 320 processors.

	HeuRet		BRKGA			
Instance	makespan	time (s)	makespan	CV (%)	time (s)	reduction (%)
dls.320.01	312813.64	0.01	306613.22	0.04	27.24	1.98
dls.320.02	321764.07	0.01	313847.15	0.07	16.72	2.46
dls.320.03	402264.85	0.01	392059.46	0.11	23.72	2.54
dls.320.04	348474.15	0.01	341436.89	0.02	28.16	2.02
dls.320.05	342086.46	0.01	334946.37	0.03	21.67	2.09
dls.320.06	311824.17	0.01	305601.28	0.02	21.93	2.00
dls.320.07	325732.30	0.01	316467.42	0.02	23.19	2.84
dls.320.08	323171.95	0.01	315065.11	0.03	26.23	2.51
dls.320.09	312326.77	0.01	305948.81	0.02	25.03	2.04
dls.320.10	296984.47	0.01	287521.34	0.12	24.04	3.19
dls.320.11	290559.21	0.01	284822.15	0.04	20.56	1.97
dls.320.12	343076.56	0.01	333085.72	0.05	19.53	2.91
dls.320.13	287276.21	0.01	281525.27	0.04	22.77	2.00
dls.320.14	311054.47	0.01	303796.42	0.06	26.81	2.33
dls.320.15	362369.67	0.01	352642.18	0.04	20.41	2.68
dls.320.16	287083.60	0.01	281082.29	0.09	25.38	2.09
dls.320.17	339666.43	0.01	329893.61	0.04	23.53	2.88
dls.320.18	368795.14	0.01	361281.06	0.07	22.08	2.04
dls.320.19	347671.73	0.01	338075.70	0.03	27.59	2.76
dls.320.20	372427.24	0.01	364013.40	0.06	18.28	2.26
Averages	330371.15	0.01	322486.24	0.05	23.24	2.38

5 Conclusions

We considered the single round divisible load scheduling problem with dedicated and heterogeneous workers. A new biased random-key genetic algorithm has been proposed for the problem.

The BRKGA-DLS heuristic improves upon the best heuristic in the literature, since it performs a global search in the space of worker permutations in order to find the best order in which the active worker processors will receive load from the master.

Computational experiments on 720 test instances with up to 160 processors have shown that BRKGA-DLS found optimal solutions for 413 instances (out of the 497 instances where the optimal solution is known), while the HeuRet heuristic found optimal solutions for only 320 of them. Moreover, BRKGA-DLS obtained better scores than

HeuRet for all but one instance and found solutions as good as the best known solution for 645 out of the 720 test instances. To summarize, BRKGA-DLS outperformed the previously existing HeuRet heuristic with respect to all measures considered in Table 1.

For the new set of larger and more realistic instances with 320 processors, BRKGA-DLS found solution values 2.38% better than HeuRet on average. In addition, the processing times of BRKGA-DLS are relatively small and never exceeded 30 seconds. Therefore, parallel processing applications dealing with large amounts of data and taking long elapsed times can benefit from BRKGA-DLS, since the additional running time needed by BRKGA-DLS may result in a significant reduction in the makespan.

References

1. Abib, E.R., Ribeiro, C.C.: New heuristics and integer programming formulations for scheduling divisible load tasks. In: Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, pp. 54–61. Nashville (2009)
2. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing **2**, 154–160 (1994)
3. Beaumont, O., Bonichon, N., Eyraud-Dubois, L.: Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: International Symposium on Parallel and Distributed Processing, pp. 1–7. IEEE, Miami (2008)
4. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems **16**, 207–218 (2005)
5. Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In: 12th Heterogeneous Computing Workshop, pp. 98–111. IEEE Computer Society Press, Nice (2003)
6. Berlinska, J., Drozdowski, M.: Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing **36**, 199–211 (2010)
7. Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. Journal of Computational and Applied Mathematics **232**, 638–654 (2009)
8. Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems **31**, 555–567 (1995)
9. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press (1996)
10. Blazewicz, J., Drozdowski, M.: Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics **76**, 21–41 (1997)
11. Błażewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling–concept and verification. Parallel Computing **25**, 87–98 (1999)
12. Cheng, Y.C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems **24**, 700–712 (1988)
13. Drozdowski, M.: Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. 321. Politechnika Poznanska (1997)
14. Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous systems with limited memory. Parallel Processing and Applied Mathematics **3911**, 847–854 (2006)
15. Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing **6**, 19–29 (2003)
16. Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research **172**, 545–559 (2006)
17. Duarte, A., Mart, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. International Transactions in Operational Research **21**, 541–558 (2014)
18. Ericsson, M., Resende, M.G.C., Pardalos, P.M.: A genetic algorithm for the weight setting problem in OSPF routing. Journal of Combinatorial Optimization **6**, 299–333 (2002)
19. Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics **17**, 487–525 (2011)

20. Gonçalves, J.F., Resende, M.G.: A biased random key genetic algorithm for 2D and 3D bin packing problems. International Journal of Production Economics **145**, 500–510 (2013)
21. Gonçalves, J.F., Resende, M.G.: An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research **21**, 215–246 (2014)
22. Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering **47**, 247–273 (2004)
23. Gonçalves, J.F., Resende, M.G.C., Costa, M.D.: A biased random-key genetic algorithm for the minimization of open stacks problem. International Transactions in Operational Research (2015). DOI 10.1111/itor.12109
24. Kim, H.J.: A novel optimal load distribution algorithm for divisible loads. Cluster Computing- The Journal of Networks Software Tools and Applications **6**, 41–46 (2003)
25. Lee, C.k., Hamdi, M.: Parallel image processing applications on a network of workstations. Parallel Computing **21**, 137–160 (1995)
26. Li, P., Veeravalli, B., Kassim, A.A.: Design and implementation of parallel video encoding strategies using divisible load analysis. IEEE Transactions on Circuits and Systems for Video Technology **15**, 1098–1112 (2005)
27. Li, X., Bharadwaj, V., Ko, C.: Divisible load scheduling on single-level tree networks with buffer constraints. IEEE Transactions on Aerospace and Electronic Systems **36**, 1298–1308 (2000)
28. Lin, W., Liang, C., Wang, J.Z., Buyya, R.: Bandwidth-aware divisible task scheduling for cloud computing. Software: Practice and Experience **44**, 163–174 (2014)
29. Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation **8**, 3–30 (1998)
30. Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased random-key genetic algorithm for routing and wavelength assignment. Journal of Global Optimization **50**, 503–518 (2011)
31. Shokripour, A., Othman, M., Ibrahim, H.: A method for scheduling last installment in a heterogeneous multi-installment system. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 714–718. Chengdu (2010)
32. Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: A new method for job scheduling in a non-dedicated heterogeneous system. Procedia Computer Science **3**, 271–275 (2011)
33. Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: New method for scheduling heterogeneous multi-installment systems. Future Generation Computer Systems **28**, 1205–1216 (2012)
34. Spears, W., deJong, K.: On the virtues of parameterized uniform crossover. In: R. Belew, L. Booker (eds.) Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236. Morgan Kaufman, San Mateo (1991)
35. Turgay, A., Yakup, P.: Optimal scheduling algorithms for communication constrained parallel processing. Lecture Notes in Computer Science **2400**, 197–206 (2002)
36. Wang, M., Wang, X., Meng, K., Wang, Y.: New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. International Journal of Pattern Recognition and Artificial Intelligence **27** (2013)
37. Wang, R., Krishnamurthy, A., Martin, R., Anderson, T., Culler, D.: Modeling communication pipeline latency. ACM Sigmetrics Performance Evaluation Review **26**, 22–32 (1998)
38. Wang, X., Wang, Y., Meng, K.: Optimization algorithm for divisible load scheduling on heterogeneous star networks. Journal of Software **9**, 1757–1766 (2014)
39. Wolniewicz, P.: Divisible job scheduling in systems with limited memory. Ph.D. thesis, Poznan University of Technology, Poznań (2003)
40. Yang, Y., Casanova, H.: RUMR: Robust scheduling for divisible workloads. In: Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing, pp. 114–125. Seattle (2003)
41. Yang, Y., Casanova, H.: UMR: A multi-round algorithm for scheduling divisible workloads. In: Proceedings of the 17th International Parallel and Distributed Processing Symposium, pp. 24–32. Nice (2003)
42. Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A., et al.: On the complexity of multi-round divisible load scheduling. Tech. Rep. 6096, INRIA Rhône-Alpes (2007)

MISTA 2015

Optimisation of a Stagger Chart for Aviation Fleet Planning

Richard Weedon • Samad Ahmadi • Mike Critchley

Abstract Within the Commercial Aviation Industry, the maintenance planning process takes into account the number of spare engines available to meet a minimum spares level (usually contractual). This is to minimize the risk of disruption to aircraft, if engines are required to be replaced due to unplanned events. To ensure the efficient use of the spare engines pool while maximizing the engine time on wing between planned removals requires a forecast that is set for the predicted life of an operator's specific aircraft type fleet. The engines are required to be refurbished at certain intervals which can be projected on to a forecast plan. The process of producing this plan by implementing the engine removals is known as Stagger. The aim of this research is to produce good quality Stagger Plans using evolutionary algorithms based upon data from an actual forecast. This paper presents our early attempts on modelling this problem and then solving it with Genetic Algorithms. Results show that the Stagger Plan produced by the GA reduced the number of weeks that the spare engines level had fallen below the minimum spare engine value when this was compared to the original forecast.

1 Introduction

Often, Operators of aircraft have various contracts with external companies such as the engine manufacturers to maintain their aircraft engine fleets. These fleets can consist of hundreds of engines. The maintenance of these engines is highly regulated to ensure maximum safety is adhered to and compliance with major air worthiness authorities. The modern commercial jet engine consists of thousands of parts which generally have specific working lives as determined by a time limits manual set by the manufacturer. These lives are also affected by how the engine is operated and in what environments. The lives are measured in hours and cycles, where a cycle in this paper is considered as a takeoff and landing. Typically these parts are affiliated with modules of the engine. The lives of certain key parts of the engine are typically tracked by using an engine health management system (EHM) and documented at shop floor visits. This information can be used in addition with the dates the engine entered into service with the operator, to predict over the life time of the fleet, a forecast of planned engine refurbishments.

To keep the aircraft operational in order to maximize revenue requires additional spare engines to account for unplanned events and also for engines that are required to come off wing due to part life time expiry or another reason as highlighted by the EHM system, for refurbishment. Typically an engine refurbishment can take between 3 – 4 months turnaround time from removal to being installed back on wing. The duration of shop visits in the Stagger Plan in this paper are between 15 and 16 weeks. If a part delivery of a fleet of aircraft to the operator occurs at a similar time then this can cause engines to require refurbishment at the same time which if not monitored can cluster and leave the spares pool empty. This could result in aircraft being grounded due to no engine availability, resulting in loss of revenue and

cancelled flights. If the engine removals are staggered, then this can reduce the risk of the number of spare engines falling below the accepted minimum value and also reduce the load on the overhaul workshops. However removing an engine off an aircraft too early than the planned removal date, results in parts being replaced which have an amount of useable life left. This is usually referenced as lost days. To monitor the operator's fleet, Forecasting and Stagger Planning tools are used but typically involve a large amount of manual processing. This paper is part of an ongoing research on this problem by authors and aims to introduce the Stagger concept and propose a basic model of the problem. A solution to the Stagger problem is developed using Evolutionary Algorithms.

2 Other studies

The maintenance of aircraft engines is a complex and expensive process that involves exhaustive fleet planning but also ensuring parts are ordered in advance (due to the complexity and associated lead times of some parts) and ready for engines that are removed. Gatland *et. al*. [1] discusses the problems that arise from the maintenance capacity planning of engines. They produce a simulation that models an engine maintenance facility that investigates effects of facility loading on turnaround time, throughput and capacity. Their paper provides further information on factors that dictate engine removals and the duration of shop visits (turnaround times). To reduce costs in the overhaul of engines accurate costing of the removal is an important part of the forecast. Engine maintenance decisions are often evaluated by using a metric commonly known as the life cycle cost (LCC).

Painter and Beachkofski [2] explain the costing implications and subsequent engine maintenance decision making by developing a simulator and data mining model to produce a more accurate LCC metric. An engine generally consists of modules. These modules are often swapped between engines to increase turnaround time at the overhaul workshop. Matching modules with similar life remaining can increase the time on wing (TOW) and reduce costs. In [3] a module swapping optimization simulator was developed for use with the air force but could equally be used in civilian engines.

A similar problem to Stagger is shown in [4] where a multi agent simulator was developed for cost reduction of engine removal scheduling. The OPS tool used an original removal plan forecast and readjusts the schedule by prioritizing engines that required overhaul due to Weibull scoring, or unforeseen circumstances. The tool does not readjust the forecast for optimization in the same way that this Stagger problem aims to provide however, it does explain the constraints such as lost days, and minimum spares. The paper also provides some mathematical representation of the turnaround times and the ratio of spare engines to fleet sizes which have not been explained in detail in this Stagger research. Additionally [4] provides information on how fleet planners/manager maintain a schedule of engine removals. Another similar problem to Stagger is also seen in [5] but for navy ship repair scheduling. Here, the paper mathematically models the constraints and fitness function to minimize the number of overlapping activities to maximize availability. The schedule is for 200 weeks with 1 maintenance cycle per ship. Instead of spare engines as a constraint, in this case 2/3 of the fleet has to be operational at any time. An evolutionary algorithm is used similarly in [5] to the Stagger solution presented in this paper.

There are software solutions available that claim to manage maintenance and cost planning such as EFPAC [6]. This software does claim to have a removal plan optimizer but does not go into detail about how this is performed. Also Clockwork Solutions [7], have a product called Insight LCM, the LSC Group [8], provide modelling and simulation solutions with optimized resource planning. SAS also provide various optimized solutions such as SAS Asset Performance Analytics [9].

Finding the optimal solution for the forecast that enables the engine to be removed with minimal lost days but maintains a minimum spare engine level can be related to timetabling problems where events can be represented as the engine removals and periods are represented

as the week numbers in the forecast subject to certain constraints which are explained later in the mathematical model shown in figure 3. There is a large amount of literature on timetabling problems and methods of solving them. Typically the approaches to solving these problems have been categorized into Sequential methods, cluster methods, constraint based methods, generalized search, hybrid evolutionary algorithms, metaheuristics, multi-criteria approaches, case based reasoning techniques, hyper-heuristics and multi-criteria approaches (for survey of approaches see [11,12]).

The study conducted in this paper, is to solve a Stagger problem that incorporates the use of an evolutionary algorithm to produce a Stagger Plan.

3 Forecasting and Stagger Planning Design

3.1 Problem Definition

The main focus of this study is to identify the best sequencing of maintenance activities based on constraints of the size of the engine spares pool and minimum contractual spares level while minimizing total lost days on wing for the fleet. A typical dataset has been produced for this problem that consisted of actual data that would normally be obtained from various information systems such as EHM data and engine shop visit forecasting data. The dataset consists of an engine number that determines the engine unique id, a start and end date of the forecast which typically shows the perceived life of the aircraft in the operator's fleet (the aircraft maybe sold to another operator at the end of the forecast or mothballed). The term mothballed is where an aircraft is kept for storage or awaiting scrap. The dataset also includes a list of all the engine removal dates that are planned for refurbishments for each engine. These refurbishments are classified as check & repair, first refurbishment, second refurbishment and mature refurbishment. There are other types of shop visits which are not relevant to this particular problem. Additionally aircraft and engine retirement dates if earlier than the end of the forecast are also included in the dataset and their induction to the Operator's fleet. The Minimum spares level for the duration of the forecast is set to 2 in this particular study.

	03/08/2015	10/08/2015	17/08/2015	24/08/2015	31/08/2015	07/09/2015	14/09/2015	21/09/2015	28/09/2015	05/10/2015	12/10/2015	19/10/2015	26/10/2015	02/11/2015	09/11/2015	16/11/2015	23/11/2015	30/11/2015
ESN	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
1046	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1070	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1071	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1072	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1073	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1074	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1110	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1113	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1114	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
AIRCRAFT_COUNT	48	48	48	48	48	50	50	50	50	50	50	50	50	50	50	50	50	50
ENGINE_DEMAND	98	98	98	98	98	102	102	102	102	102	102	102	102	102	102	102	102	102
CHECK_AND_REPAIR_COUNT	2	2	2	2	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ENGINE_COUNT	107	107	107	107	107	111	111	111	111	111	111	111	111	111	111	111	111	111
AVAILABLE_ENGINE_COUNT	104	103	103	103	106	109	109	109	109	109	108	108	108	108	108	108	108	108
SPARES_LEVEL	8	7	7	7	10	9	9	9	9	9	8	8	8	8	8	8	8	8

Fig. 1: A Non-Optimal Stagger Plan

3.2 Modelling the Stagger Problem

An example of a Stagger Plan that has not been optimized can be seen in figure 1, this shows the engines on the far left with the weekly dates for the duration of the forecast at the top. The 0 represents an engine that is currently available either as a spare or currently flying. The pink colored 1 represents engines that are currently having a shop visit. A white – 1 identifies an engine that has not been inducted into the fleet and therefore cannot be included into the calculations. Check & Repairs are not included as a main shop visit and are subsequently not recognized as a shop visit in the removal data used for the Stagger Plan of an individual engine. However, a weekly total of check & repairs has been included and is part of a calculation that affects how many engines are available as spares.

The chart in figure 1 is similar to what the final designed optimized Stagger Plan will show with an additional calculated value – lost days. The lost days and spare engines can be seen graphically for a Stagger forecast in figures 4, 5 and 6 in section 5. The calculations required to produce the minimum spares level and lost days are as follows:

Spares Available = Engine Count Available – Engine Demand. (1)

Engine Count Available = Engine Count – C&R Count - Engines in shop that week (2)

Engine Demand = Aircraft Count × no. of engines on an aircraft. (3)

Engine Count = No. of engines in shop (removal) for a particular week. (4)

Lost Days = Original engine removal week – adjusted engine removal week × 7. (5)

3.3 A Genetic algorithm for the Stagger Problem

In this paper an evolutionary algorithm is developed to produce an optimal solution for a Stagger Plan. A chromosome was represented by a 3 dimensional array. The first dimension contained all of the week numbers in the forecast, the second dimension was the engine serial numbers (ESN's) and the third dimension consisted of 9 values: ESN, start of removal week number, end of removal week number, Number of weeks removal moved forward, week ESN inducted into fleet, week ESN removed from fleet, Total Fleet spares Value, Lost Days and finally a Marker that is a Boolean data type. All of the dates have been converted to week numbers over a range from 0 to the last week of the forecast – 1.

The fitness function is required to determine the quality of each member of the population. In this study the fitness function is actually an inequality which was developed around the number of spare engines available each week and the number of lost days. The number of spare engines available should never fall below the minimum value and the function should be weighted accordingly. The lost days should be kept as minimal as possible but should not override the spare engines available. When the child is being scored the coding checks every week of the forecast and if the spare engines level drops below the minimum level then the inequality shown in equation (6) is used.

$$\frac{-Av.LostDays + 1}{50} \quad For\ Spare\ engines < Minimum\ spare\ engine\ level \tag{6}$$

Alternatively if the spare engines level is above the minimum for the whole forecast then the inequality in equation (7) is used:

$$\frac{500}{Av.LostDays + 1} \quad For\ Spare\ engines \geq Minimum\ spare\ engine\ level \tag{7}$$

These inequalities were designed so that the lost days should achieve a higher performance as the lost day's approaches 0 if the minimum spare engines level is maintained for the whole forecast. A positive fitness score would represent that the minimum spares level has been met for the whole forecast. A negative fitness function would indicate that the minimum spares figure has fallen below the threshold. How close the negative fitness score is to zero indicates that the lost days are minimal.

The crossover operation consisted of copying one part of one parents array by randomly picking an ESN and copying all of the ESN's to the left (including the randomly picked ESN) and then copying the remaining ESN's to the right from another parents array to create a child. To enable the selection of the chromosomes to represent the parents then 3 methods of selection were developed to determine if any specific method produced an improvement to the results. These selection methods are roulette wheel selection, elitism selection and a random selection. The roulette wheel selection used an array to store the proportioned fitness of all the individuals by using the formula:

$$\text{Stagger Fitness} / \text{Total Population Fitness} \times 360 \quad (8)$$

The elitism method uses an array that orders those individuals by their fitness and the highest scoring chromosomes are then selected as parents. Likewise for the random method a random number generator was used to pick parents randomly in an array. The mutation operator stage of the process was developed by calling a random number generator with the range of numbers from 1 to 2 as shown in the pseudo code in figure 2.

```
A = Engine original removal date
B = Engine Induction date
Max stagger forward (M) = 26
For each week (x) in schedule
For each original engine scheduled (O) removal in week (x)
        If A < M+1 then
          New removal(N) = Randomise(A-B)
          Elseif A >= M+1 then
                If B >= M then
                  N = A-Randomise(26)
                Else if B < M
                  N = A – Randomise(B)
Repeat for each engine
Repeat for each week
```

Fig. 2

If a 1 was generated then the mutation operation was implemented. At mutation, another random number was generated from 0 to the number of engines in the forecast – 1, to pick an engine at random. The code was developed to determine if an engine is in shop in a particular week and whether the engine was inducted into the fleet within 26 weeks of the forecast start week. Also if the removal date is less than the 26 weeks from the induction week then this needs to be accounted for. The resultant figure from the above logic is sent as a range for another random number to be generated. This random number is subtracted from the original removal week to produce a new removal week. Finally the weakest (fitness) chromosome was erased from the population after each chromosome is spawned.

The software used to develop the optimized Stagger Plan was Microsoft Excel VBA 2010. The structure of the coding involved the use of 2 and 3 dimensional arrays. As mentioned earlier, an array was used to store the data that was imported from the worksheets which is referenced in the Setup worksheet with the cell ranges that the data lies within. Another array was created that produced a generic Stagger Plan with the original engine removals and the week number of the forecast. A third dimension was used to switch between removal start and end dates, engine induction dates, adjusted removal date, engine retired from fleet date, lost days and fleet spare engines. This array was used as a template and was copied into each

chromosome at the initialize population stage of the fleet prior to mutation. At initialization of the population stage no crossover operation was used due to no parents. To create a chromosome a class module was used to create a collection called *Staggers.*

A Stagger is a member (property) of the Staggers collection which was used to store the chromosome. Each Stagger had a property associated with it. These include a variant type array which was a version of Stagger Plan, a generation property to store the generation number that it was spawned in and a name property. After each Stagger was created another array was used to work out its fitness and store the result. This array *varXScore* was used to compare all the other active chromosomes in the population to identify which one is erased. A temporary array was used to sort the highest score in descending order, from the *varXScore* array when this process was initiated.

Once the population was initialized then subsequent generations were created using a loop until the required number of generations was reached as set on the Setup worksheet. At the Mutation stage, if it was selected by the random generator the mutation was set by using the original removal date and randomly moving the removal date forward by the criteria discussed earlier.

When the last generation was created the Stagger Plan from the chromosome with the best fitness was exported into a worksheet called Final. The scores of all of the chromosomes can be seen in the Score worksheet.

4 Results

4.1 Initial Tests

The test runs were produced initially with the population set to 4 and the number of generations set at 3. The number of parents per generation for this study was always set to 2. These settings were used initially to test the duration of the run and whether any major performance issues would be encountered. The table below shows the fittest chromosome from each run for both the roulette wheel selection and elitism selection methods used for the selection of parents:

Run 1						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-3	14.28	-0.31	6	3	00:00:36	Elitism
-3	15.04	-0.32	7	4	00:00:33	Elitism
-3	15.85	-0.34	6	3	00:00:34	Elitism
-2	15.90	-0.34	7	4	00:00:37	Roulette
-3	14.30	-0.31	5	2	00:00:36	Roulette
-3	14.35	-0.31	5	2	00:00:34	Roulette

Due to the very small population and number of generations created, there was an improvement to the Stagger Plan. A positive fitness score would represent that the minimum spares level has been met for the whole forecast. A negative fitness would indicate that the minimum spares figure has fallen below the threshold. How close the negative fitness score is to zero indicates that the lost days are minimal. In fact the average lost days for all removals in the Stagger is around 14 days in the above table. All the parents that are created at the initialization stage contain an average lost day's figure > 250 which is suggesting there is a problem with the calculation of the creation of the parents as the maximum Stagger should only be 180 days.

However the figure settles with the next generations. Due to time constraints of this study this problem was not investigated further. Changing the population to 20 with 10 generations did increase the processing time from around 35 seconds to 105 seconds as shown in the table below.

Run 2						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-1	13.89	-0.30	28	9	00:01:44	Roulette
-2	13.75	-0.29	23	4	00:01:39	Elitism
-3	14.28	-0.31	24	5	00:01:28	Roulette
-2	13.75	-0.29	23	4	00:01:40	Elitism

As can be seen in the above tables increasing the population size and the number of generations has not improved the results too much at this point. Increasing the generations to 25 with a population of 20 increased the processing time to approximately 225 seconds. The best fitness achieved for these runs only improved by +0.01 and the average lost days decreased to 13.04.

Run 3						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-2	13.49	-0.29	45	26	00:03:47	Elitism
-3	13.04	-0.28	41	22	00:03:52	Roulette

It can be seen through the above tables that there is no real difference between the elitism and roulette wheel selection methods at this stage. Increasing the generations to 75 again showed an improvement with the fitness with respect to the lost days being reduced to 11.6 days for the whole Stagger Plan. The processing time was 13 minutes. However the minimum spares value is not improving for this run because the best chromosome still had – 2. The fitness function is improving the lost days consistently with the increase in the number of generations.

4.2 Improving the Fitness Function

An amendment was made to the inequality of equation (6) to give:

$$\frac{-Av.\,LostDays + 1}{50} + 2 \times (No.\,of\ Spare\ engines) \tag{9}$$

This was designed to provide more weight to the minimum spares level to ensure it would add more bias to the spare engines rather than the lost days. Using a population of 20 and 10 generations provided the following results as shown in run 4.

Run 4						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-2	13.52	-0.29	22	3	00:01:42	Roulette
-3	13.54	-0.29	22	3	00:01:44	Elitism

Run 5								
Min Spares	**Av. Lost days**	**Fitness score**	**Chromosome**	**Gen.**	**Duration (min)**	**Crossover**	**p**	**g**
-1	11.97	-2.26	50	41	00:07:12	roulette	10	50
0	12.78	-0.28	54	45	00:07:07	elitism	10	50
-1	16.56	-2.35	7	4	00:00:34	roulette	4	3
-2	15.31	-4.33	6	3	00:00:33	elitism	4	3
0	15.66	-0.33	21	2	00:01:38	roulette	20	10
-2	14.61	-4.31	29	10	00:01:40	elitism	20	10
-1	15.89	-2.34	38	19	00:03:42	roulette	20	25
0	13.09	-0.28	45	26	00:03:44	elitism	20	25
-3	14.80	-0.32	7	4	00:00:38	roulette	4	3
-1	13.46	-0.29	5	2	00:00:33	elitism	4	3
-2	14.61	-0.31	22	3	00:01:39	roulette	20	10
-2	13.59	-0.29	30	11	00:01:40	elitism	20	10
-3	14.09	-0.30	30	11	00:03:42	roulette	20	25
-2	11.17	-0.24	45	26	00:03:45	elitism	20	25
-2	14.11	-0.30	57	48	00:07:05	roulette	10	50
-1	11.25	-0.24	51	42	00:07:09	elitism	10	50

In run 5, the p represents the population size and g represents the number of generations. The colored section represents the fitness inequality in equation (8) and the uncolored section represents the fitness inequality in equation (7). Each of the results displayed in the above table are those chromosomes who have the best fitness function of that particular test. Interestingly, the elitism selection for the fitness inequality in equation (7) outperforms the roulette wheel selection for fitness and lost days and the weekly remaining spare engines. For the revised fitness function, the operators are similar. Comparing the spare engines remaining between the 2 fitness inequalities clearly show that the fitness inequality in equation (8) produces a better spare engine remaining figure and appears to consistently improve the spare engine figure as the generations are spawned.

A further elitism selection run produced a minimum spare level of 1 which was using a population size of 10 and 40 generations. The fitness was 1.64 and the average lost days for each week of the forecast was 16.85 days. The lost days can be seen in a plot as shown in figure 3 and the minimum spare engines can be seen in figure 4 below.

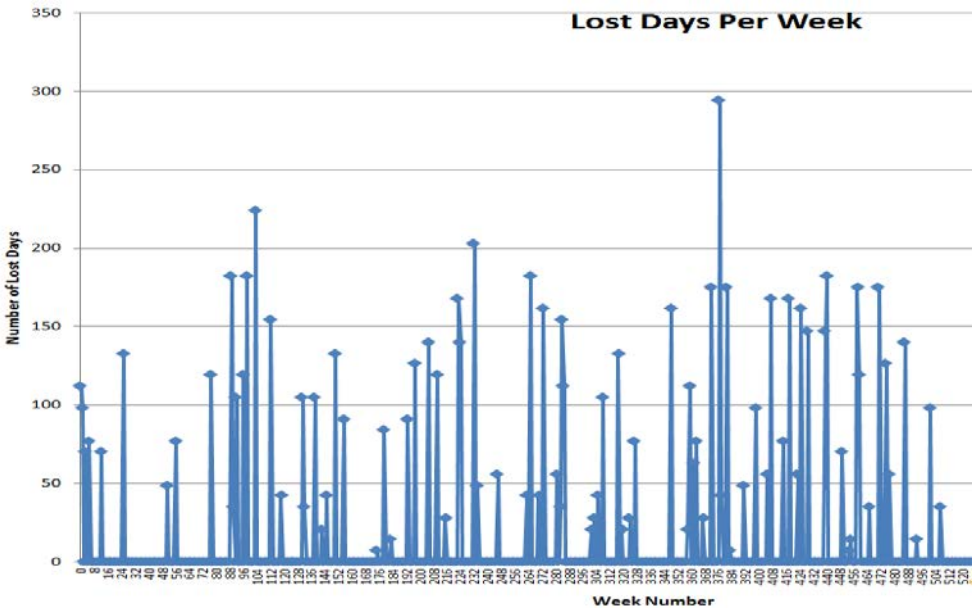


Fig.3

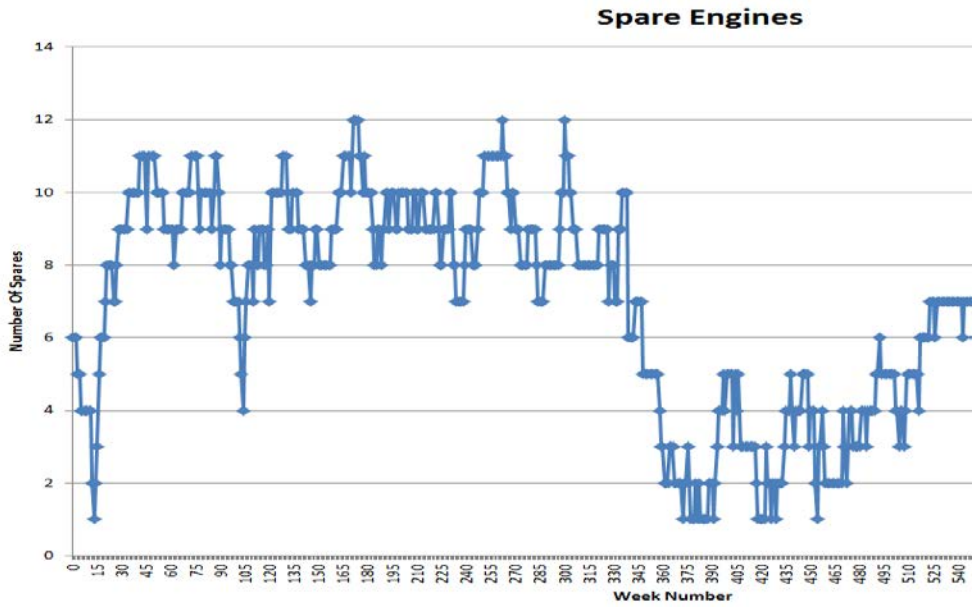


Fig.4

The Stagger Plan produced above did fall below the minimum spares level of 2 however, the total lost days across all weeks in the forecast was 8372 days. Although this figure seems high but the lost days per engine (114 in this fleet) are 73.4. Also if the average daily cycles flown per engine is 2.2 then the total number of cycles lost is 18418.4. The lost cycles per engine are then 161.6. Typically an engine is taken off wing at approximately 50 cycles prior to the latest possible removal date to provide a safety margin.

The results have shown that the evolutionary algorithm overall improved the optimization of Stagger. The results from tables 1 to 4 show an improvement in the fitness score for lost days. However, the minimum spares level was completely random from –1 to -3 for the majority of the test runs above. Improving the fitness function to that of equation 8 did not improve the fitness scores until a code change to the mutation operator was addressed. After this amendment

the fitness scores did improve. The elitism selection and the roulette wheel selection did not seem to be too different for both fitness inequalities. Run 3 shows the roulette wheel selection produced a better fitness score and lost day values where run 2 produced better elitism selection results. The final run produced better results that were an improvement from the standard Stagger Plan that used the forecasted removal dates with – 4 spare engines as the lowest value of spares as shown in figure 5 below:

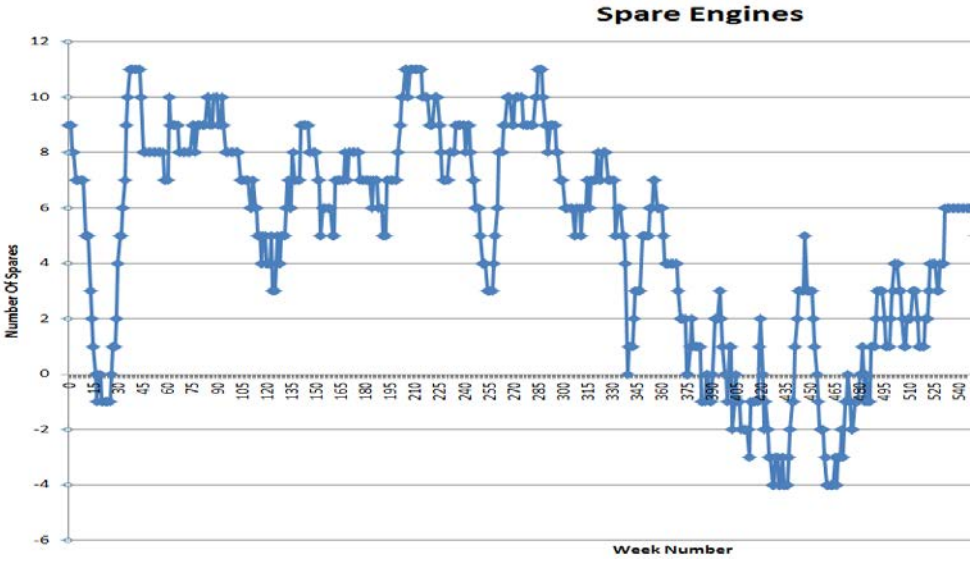


Fig.5

The majority of the runs above were timed to determine that the processing is consistent for each run of similar settings and also to evaluate how much time a large population and number of generations would take to complete, providing the system could handle the memory and processor resources required. The times were fairly consistent with the runs. The computer that performed the test runs was running windows 7, 32-bit operating system with 4GB ram and an AMD Phenom II quad core processor – 3.3GHz. Typically performing a run used approximately 25% on the CPU and increased the RAM usage by 0.06GB consistently, providing no other applications were used in addition to Microsoft Excel. These values include all the other system processes that are running in the background.

None of the above runs produced an optimal solution that satisfied the condition that no spare engines would fall below the minimum value. However, if the population was increased to 100 and left for 10000 generations, then this may produce a better solution. This figure would create a total of 10,100 members. To create a population with 10 members over 50 generations (60 in total) involved a processing time of approximately 7 minutes which if used as a guide for 10,100 members would take 19.64 hours to generate. The best result was obtained with a population of 50 to a 100 with 40,000,000 iterations (generations). Unfortunately using that many generations with the existing set up would not be feasible.

5 Conclusion

In this paper the Stagger problem has been introduced with a basic solution developed that uses an evolutionary algorithm. The Stagger problem could be expanded to cover Life Separation. This is where some engines are taken off wing as they are delivered to the fleet and replaced with a spare engine that has half of the life of the new engine. The engine that has been taken off is typically stored as new for a length of time to allow a break between future removals and subsequently reduce demand in the future as all of the engines need to be refurbished. However, there are a limited number of spare engines for a fleet to use as can be seen in figures 7 and 8 and where a high volume of shop visits occur these spare engines may be required. This extra feature of Stagger could be investigated in future work.

Future work is also planned to present a Mathematical model and also more accurate and sophisticated algorithms to find optimal solutions.

References

1. Gatland, R.; Yang, E.; Buxton, K., “Solving engine maintenance capacity problems with simulation”, Simulation Conference, 1997, IEEE, pages 892-899, 7-10 Dec. 1997
2. Painter, M.K.; Erraguntla, M.; Hogg, G.L.; Beachkofski, B., "Using Simulation, Data Mining, and Knowledge Discovery Techniques for Optimized Aircraft Engine Fleet Management," Proceedings of the Winter Simulation Conference, 2006. IEEE, pages 1253,1260, 3-6 Dec. 2006
3. Yutsung W; Jaw, L.; Rendek, P.; Moses, E.; Robinson, M.; Driver, S.; Senior, K., "Demonstration of A Reliability Centered Maintenance (RCM) Tool to Extend Engine's Time-On-Wing (TOW)," Aerospace Conference, 2007, IEEE, Pages 1,5, 3-10 March 2007
4. Stranjak, A.; Dutta, P.S.; Ebden, M.;Rogers, A.;Vytelingum, P.
5. “A multi-agent simulation system for prediction and scheduling of aero engine overhaul”, 2006, Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (Industrial Track), ACM DL, Industrial Track, 8-12 May 2006
6. Deris, S.; Omatu S.; Ohta H.; Kutar S.; Samat P. A.; “Ship maintenance scheduling by genetic algorithm and constraint-based reasoning”, European Journal of Operational Research, 1999, Volume 112, Issue 3, 1 February 1999, Pages 489-502
7. Aerdata, 2015, EFPAC Engine Maintenance Cost Planning [Online] Available from http://www.aerdata.com/efpac-engine-management-system.html, [Accessed 14/05/15]
8. LSC Group, 2015, Modelling & Simulation [online] Available from: http://www.lsc.co.uk/our_services/ikm__business_analysis/modelling__simulation/LSC Group, [Accessed 03/06/2015]
9. Clockwork Solutions, 2015, Insight LCM, [online] Available from: http://clockwork-solutions.com/products/insight-lcm/, [Accessed 03/06/2015]
10. SAS, 2015, SAS Asset Performance Analytics, [online] Available from: http://www.sas.com/en_us/software/supply-chain/asset-performance-analytics.html, [Accessed 03/06/2015]
11. Babaei, H, Karimpour, J. and Hadidi, A., A survey of approaches for university course timetabling problem, Computers & Industrial Engineering, Available online 21 November 2014, ISSN 0360-8352, http://dx.doi.org/10.1016/j.cie.2014.11.010.
12. Burke, E.K. and Petrovic, P., Recent research directions in automated timetabling, European Journal of Operational Research, Volume 140, Issue 2, 16 July 2002, Pages 266-280, ISSN 0377-2217, http://dx.doi.org/10.1016/S0377-2217(02)00069-3.

Abstracts

MISTA 2015

Investigating Scheduling models for Power Plant Preventive Maintenance using Genetic Algorithm

Neha Prajapat • Windo Hutabarat • Ashutosh Tiwari • Marco Pattacini

1 Introduction

Optimal scheduling of preventive maintenance tasks is imperative for power generation companies as maintenance activities have a substantial impact on equipment reliability and production costs. Power generation companies are responsible for providing a dependable service at a competitive price to customers; thus high value power plant equipment must be subjected to preventive maintenance at predetermined intervals to prevent unexpected failures. All maintenance activities require the plant to be taken off-line which can cost companies in the region of millions in lost revenue. In particular combined cycle power plants have a large number of expensive machines which have different maintenance guidelines and strict maintenance intervals. Alignment of these activities through optimization can provide many significant advantages for generation companies. The preventive maintenance scheduling problem has been widely studied within the power industry with different applications areas. The Generator Maintenance Scheduling problem (GMS) has received the most academic interest as this addresses a large scale complex system [1] which often governs other planning and operations activities [2]. The preventive maintenance scheduling problem has been formulated in numerous different ways including: integer programming [3; 4], dynamic programming [5] and simulation methods [6].

Optimization criteria for preventive maintenance scheduling are typically cost and reliability based [7]. Cost based criteria are the most popular approach often accounting for maintenance and production costs, start-up costs, indirect costs and fuel and electricity prices for profit calculation. Reliability has been used as a primary objective function particularly for the GMS problem [4]. A number of different optimization methods have been used to solve the preventive maintenance scheduling problem including: heuristic methods [8], decomposition methods [9] and evolutionary computation approaches among others. Evolutionary computation approaches have been an area of growing interest with algorithms such as Genetic Algorithms [10], Simulated Annealing [11] and Particle Swarm Optimization [12] being increasingly applied to solve the problem.

In the following sections two models are proposed to solve the preventive maintenance scheduling optimization problem for the main components of a combined cycle power plant. This paper presents the multi objective optimization problem of scheduling multiple maintenance activity types for major combined cycle power plant equipment such as the gas turbine, steam turbine and steam generator.

Neha Prajapat; Marco Pattacini
Alstom Power, Rugby, CV21 2NH; Cranfield University, MK43 0AL, United Kingdom
E-mail: neha.prajapat@power.alstom.com; marco.pattacini@power.alstom.com

Windo Hutabarat, Prof Ashutosh Tiwari
Cranfield University, Bedford, MK43 0AL, United Kingdom
E-mail: w.hutabarat@cranfield.ac.uk; a.tiwari@cranfield.ac.uk

Nomenclature
i- machine number
j- period number
k- maintenance activity index
r_i- minimum inspection frequency
d_i- fixed duration of maintenance of machine i
$x_{i,j}$- maintenance assignment variable
$C_{i,k}$- period number variable for maintenance k for machine i
B- number of outage days
S_i- constraint violation indicator for machine i
T- total constraint violation
U_i- minimum number of maintenance activities for machine i
W_1, W_2- weighting factors for equivalent operating hours EOH

2 Problem Description

Combined cycle power plants have a large number of high value equipment such as the gas turbine, steam turbine, and heat recovery steam generator. Maintenance intervals for the machinery are based on equivalent operating hours (EOH) accrued by the power plant. The EOH limits or inspection intervals are governed by the number of operating hours of the plant (OH- operating hours), the number of cyclic events (CE- cyclic events such as start-up and shut down), the operating mode of the gas turbine and by the operating regime of the entire plant. These factors influence the degradation of equipment due to the temperature gradient and hence govern the inspection intervals. The EOH limit for each piece of equipment is calculated using formula (1). This is used as a minimum inspection frequency.

$$EOH = W_1 OH + W_2 CE\,, \qquad EOH = r_i \tag{1}$$

In (1) $W_1$and $W_2$are weighting factors based on the operating regime of the power plant and the operating mode of the gas turbine. This paper considers the case where the power plant will be run on one particular operating mode and one operating regime for the plant lifetime. The EOH limits are a minimum inspection frequency for the equipment.

The problem formulation is based on 0/1 integer programming [3]. Machines are maintained at integer multiples of a fixed time period [13] and each maintenance activity has a duration d_i which is a fixed number of outage days. The value of k is incremented by 1 for each machine every time another maintenance activity is scheduled. Quarter of a year was chosen as an appropriate time period; there are 80 periods in the plant lifetime. The minimization of the number of outage days is chosen as the objective function. The optimization problem is formulated as follows:

Minimise $$B = \sum_{j=1}^{80} \max_i (x_{i,j} d_i) \tag{2}$$

Subject to interval constraint $$C_{i,k+1} - C_{i,k} \leq r_i \quad \text{where} \quad \forall x_{i,j} = 1, \quad j = C_{i,k} \tag{3}$$

The constraint violation of the maintenance interval length is formulated as an objective function for both models; formulation of both models is as follows.

2.1 Model 1

This model has been designed to be easy to manipulate for engineers and the solutions produced will have the same number of maintenance activities as current solutions. The primary objective function is given in (2). The second objective function is the minimization of maintenance interval constraint violations as given in equation (4). A variable S_i is used to indicate if there is any constraint violation for each machine. This model also fixes the number of maintenance activities to 30 as a constraint in equation (6). This constraint is based on the existing solution.

Minimise $$T = \sum_{i=1}^{6} S_i \tag{4}$$

For each machine i $\forall k$ $$S_i = \begin{cases} 0 \; if \; C_{i,k+1} - C_{i,k} \leq r_i \\ 1 \qquad otherwise \end{cases} \quad (5)$$

Subject to $$\forall i, j \qquad \sum x_{i,j} = 30 \quad (6)$$

2.1 Model 2

In this model the total number of maintenance activities is not fixed; this provides more flexibility allowing a much larger search space to be explored. The first objective function for this model is given in equation (2). The second objective (7) is to minimize the sum of constraint violations; this sums the degree of constraint violation for all violations.

Mimimise $$T = \sum_{i,k} C_{i,k+1} - C_{i,k} - r_i \quad \forall \, i,k: C_{i,k+1} - C_{i,k} \leq r_i \quad (7)$$

After test runs it was noticed that this constraint led to very few maintenance activities being scheduled so a further penalty of 500 was applied to each machine schedule which did not meet the minimum number of maintenance activities. A new variable is introduced for the minimum number of activities for each machine U_i. This variable has fixed values for each machine based on current solution.

$$\forall i \quad if \; k \leq U_i \quad T_i = 500 \quad (8)$$

The second model is designed to explore solutions which may not be intuitive to engineers and to provide flexibility with respect to the constraint violations.

3 Results and Discussion

The optimization platform chosen is GanetXL [14] and the multi objective optimization algorithm used is NSGAII. GanetXL is an add-in for MS Excel to enable optimization on spreadsheet based models. The model was formulated with multiple objectives in Excel and solved using the NSGAII algorithm.

A manual 'trial and error' optimization exercise was carried out on the first model. This heuristic method resulted in a solution with no constraint violations and 320 outage days providing a saving in the region of 12% relative to the current solution. This solution was for comparison and was not used as a starting point by the optimizer in subsequent runs.

Following the manual optimization, both models were run with different setting combinations for the mutation and crossover operators. Setting combinations used are shown in Table 1; the probabilities of each operator are indicated in brackets.

		Mutation	
		Simple (0.05)	By-gene (0.05)
Crossover	No crossover (0)	1	
	Single point (0.95)	2	3
	Multi point (0.95)		4

Table 1- A table to show the various run settings with the NSGAII optimisation algorithm

Model 1 was run with a population size of 200 for 50,000 generations for all combinations in Table 1. Out of all 200,000 solutions generated only 5 solutions were found which did not violate the interval constraints. It was concluded that the model was not able to find solutions which didn't violate constraints in a reasonable number of evaluations, hence no further investigation was performed using this model. Model 2 was run with a population size of 200 for each of the settings combinations in Table 1, the model was run until either convergence or 50,000 evaluations for each case.

Results are shown in Figure 1 for all 4 scenarios against the current solution.

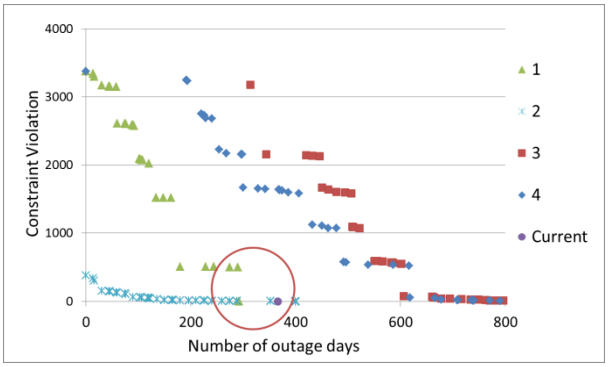


Figure 1- A graph to show constraint violation against outage days for various setting combinations for model 2

The region of interest is highlighted by the circled area. Figure 2 focusses in on this region.

The solutions of interest in figure 2 lie on and near the x axis as these solutions have little or no constraint violations. It can be seen that the optimization settings 3 and 4 produce no solutions in figure 2; hence the by-gene mutation operator is not able to find solutions in the region of interest. For optimization setting 2 a range of solutions were found with differing constraint violations close to the x axis; however the only solution which did not violate any constraints had 400 outage days. This is a significantly worse result compared to the current solution. The best result from setting 2 had 288 outage days and a constraint violation of 2. This result could be considered interesting for engineers to analyze. For optimization setting 1, only 1 solution was found in the region of interest. The solution found had no constraint violations and 292 outage days; this is a reduction of 20% compared to the current solution and a reduction of around 9% compared to the initial optimized result. This is a highly impressive result considering the large number of input variables and the large search space. Counter intuitively this schedule increases the overall number of maintenance activities assigned to machine 1. Diagrammatic schedules are shown in Figure 3 for the initial optimized solution and the optimal result produced using setting 1.

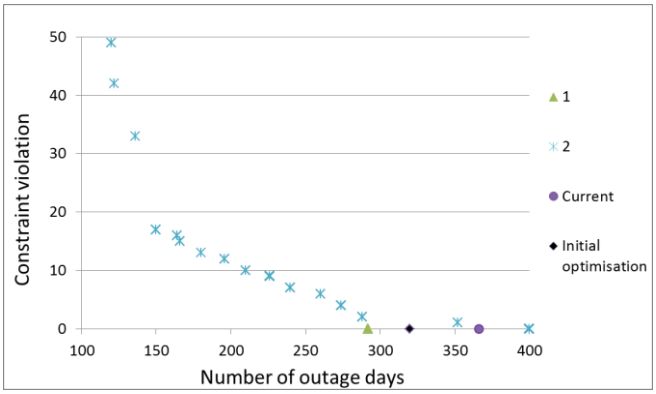


Figure 2 - A graph to show constraint violation against outage days for various setting combinations for model 2

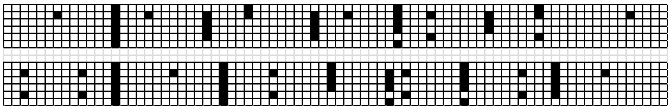


Figure 3- Diagrammatic schedules for the initial optimized solution (above) and the optimal solution found using setting 1 (below).

A number of interesting observations and conclusions can be made based on the results produced. Model 1 took considerably less time to run than the second model due to the increased size of the search space in model 2. The constraint violation was implemented differently for both models. Model 2 performed much better as the magnitude of the overall constraint violation was calculated by the model and the algorithm was able to use this information to guide the search. The advantage of model 2 is that the results show a range of solutions with exact constraint violation information allowing the decision maker to consider various solutions; this information is not calculated in model 1. The by-gene mutation operator was not able to produce solutions in the region of the optimal solutions suggesting it is a poor choice for this optimization model. Remarkably the simple mutation operator without crossover performed extremely well within a limited number of generations. This result was unexpected and the algorithm was able to produce solutions which aligned the maintenance activities very early in the optimization run; this was not evident in previous optimization runs. The result increased the total number of maintenance activities for machine 1 in the first row which is counter intuitive for engineers. However this counter intuitive method has led to an overall reduction in outage days and provides a highly interesting solution for engineers to assess. Some modification will be required as some activities are scheduled too close together.

Further research and repeat optimization runs are required to confirm the advantage of the simple mutation operator and to explore the effect of the crossover operator on the results. Refinement of the optimized solution using expert opinion and cost analysis is also required.

4 References

[1] Perez-Canto, S. "Using 0/1 mixed integer linear programming to solve a reliability-centered problem of power plant preventive maintenance scheduling", *Optimization and Engineering*, vol. 12, no. 3, pp. 333-347, (2011).

[2] Dahal, K. P., Aldridge, C. J. and McDonald, J. R. "Generator maintenance scheduling using a genetic algorithm with a fuzzy evaluation function", *Fuzzy Sets and Systems*, vol. 102, no. 1, pp. 21-29, (1999).

[3] Dopazo, J. F. and Merrill, H. M. "Optimal Generator Maintenance Scheduling Using Integer Programming.", *IEEE Trans Power Appar Syst*, vol. PAS-94, no. 5, pp. 1537-1545, (1975).

[4] Perez-Canto, S. and Rubio-Romero, J. C. "A model for the preventive maintenance scheduling of power plants including wind farms", *Reliability Engineering and System Safety*, vol. 119, pp. 67-75, (2013).

[5] Yamayee, Z. A. and Sidenblad, K. "Computationally Efficient Optimal Maintenance Scheduling Method.", *IEEE transactions on power apparatus and systems*, vol. PAS-102, no. 2, pp. 330-338, (1983).

[6] Tsai, Y. -., Wang, K. -. and Teng, H. -. "Optimizing preventive maintenance for mechanical components using genetic algorithms", *Reliability Engineering and System Safety*, vol. 74, no. 1, pp. 89-97, (2001).

[7] Reihani, E., Sarikhani, A., Davodi, M. and Davodi, M. "Reliability based generator maintenance scheduling using hybrid evolutionary approach", *International Journal of Electrical Power and Energy Systems*, vol. 42, no. 1, pp. 434-439, (2012).

[8] Moghaddam, K. S. and Usher, J. S. "Preventive maintenance and replacement scheduling for repairable and maintainable systems using dynamic programming", *Computers and Industrial Engineering*, vol. 60, no. 4, pp. 654-665, (2011).

[9] Marwali, M. K. C. and Shahidehpour, S. M. "A deterministic approach to generation and transmission maintenance scheduling with network constraints", *Electric Power Systems Research*, vol. 47, no. 2, pp. 101-113, (1998).

[10] Baskar, S., Subbaraj, P., Rao, M. V. C. and Tamilselvi, S. "Genetic algorithms solution to generator maintenance scheduling with modified genetic operators", *IEE Proceedings: Generation, Transmission and Distribution*, vol. 150, no. 1, pp. 56-60, (2003).

[11] Kim, H., Nara, K. and Gen, M. "A method for maintenance scheduling using GA combined with SA", *Computers and Industrial Engineering*, vol. 27, no. 1-4, pp. 477-480, (1994).

[12] Pereira, C. M. N. A., Lapa, C. M. F., Mol, A. C. A. and Da Luz, A. F. "A Particle Swarm Optimization (PSO) approach for non-periodic preventive maintenance scheduling programming", *Progress in Nuclear Energy*, vol. 52, no. 8, pp. 710-714, (2010).

[13] Zhao, Y., Volovoi, V., Waters, M. and Mavris, D. "A sequential approach for gas turbine power plant preventive maintenance scheduling", *Proceedings of the ASME Power Conference, 2005*, Vol. PART A, pp. 353, (2005).

[14] Savic, D. A., Bicik, J. and Morley, M. S. "A DSS generator for multiobjective optimisation of spreadsheet-based models", *Environmental Modelling and Software*, vol. 26, no. 5, pp. 551-561, (2011).

MISTA 2015

Routing Strategy for Prioritized Limited Multi-server Processor-Sharing System that includes Servers with Various Capacities

Yoshiaki Shikata • Nobutane Hanayama

1 Introduction

Under the processor-sharing (PS) discipline, if there are n (> 0) requests in a single-server system, each request receives 1/n of the server capacity. No arriving request has to wait for service because it is served promptly, even if the service rate becomes slow. Thus, the sojourn time of each request that receives service in the server is n times the service time. A PS discipline with a priority structure has been proposed, wherein a larger service ratio is allocated to requests with higher-priority [1]. In order to prevent an increase in the sojourn time of each request in such a prioritized single-server PS paradigm, and to realize a realistic sharing model, a method for limiting the number of requests that receive service has been proposed [2]. In such a prioritized limited single-server PS system, a higher-priority request is allocated a larger service ratio compared with a lower-priority request. Moreover, the sum of the number of the requests that receive service is restricted to a fixed value. Arriving requests that cannot receive service are attached to the service waiting queue (waiting system), or are rejected (loss system).

On the other hand, communication services, such as web server farms, database systems, and grid computing clusters, routinely employ multi-server systems to provide a range of services to their customers. An important issue in such multi-server systems is to determine the server to which an arriving request should be routed in order to optimize a given performance criterion. Therefore, in this paper, we first propose a novel prioritized limited multi-server PS system where each server can have various capacities, and N ($\geq$2) priority classes are allowed in each PS server. Routing strategies of such prioritized limited multi-server PS system that take into account the capacity of each server are also proposed, and the performance evaluation procedure of these strategies is discussed. Then, practical performance measures of these strategies, such as loss probability, mean waiting time in the service waiting queue, and mean sojourn time, are evaluated via simulation. Based on the evaluation results, we discern the

Yoshiaki Shikata
Shobi University
E-mail: shikata@ictv.ne.jp

Nobutane Hanayama
Shobi University
E-mail: nob-hanayama@jcom.home.ne.jp

most suitable routing strategies of the prioritized limited PS system that includes multi-servers with various capacities, and requests of N priority classes.

Under the PS rule, when a request either arrives or departs from the PS server, the remaining sojourn time of other requests currently receiving service is extended or reduced, respectively. This extension or reduction of sojourn time is calculated using the number of requests of each class and the priority ratio. Employing a simulation program to execute these events and calculations allows us to analyze the prioritized limited multi-server PS rule, which is realistic in a time-sharing system (TSS) with a sufficiently small time slot.

Load-balancing strategies for multi-class multi-server PS systems with a Poisson input stream and heterogeneous service rates have been investigated [3]-[7]. However, there have been few studies on prioritized limited PS systems [8]. Accordingly, routing strategies for prioritized limited multi-server PS systems with various capacities are scarce, and practical performance measures of such strategies have yet to be investigated.

2. Prioritized limited multi-server PS system that includes servers with various capacities

2.1 System concept

In the prioritized limited multi-server PS system, an arriving request enters the dispatcher, which routes the request to each PS server according to a predetermined strategy. Suppose that there are N classes, and an arriving request, which is routed to server h, encounters n_{hj} class-j requests (including the arriving request). According to the proposed prioritized limited multi-server PS rule, if $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq SFC$, an arriving class-k request individually and simultaneously receives $m_k / \sum_{j=1}^{N} m_j * n_{hj}$ of the capacity of server h. When a server that meets condition $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq SFC$ does not exist, the arriving request is queued in the corresponding class waiting room prepared in the dispatcher, or rejected. Here, m_j (≥ 1) denotes the priority ratio of the class-j request, SFC ($\leq \infty$) is the service facility capacity, and C_h is the capacity ratio of server h to the reference server. The service time of a request in the server h is given by the service time of that request in the server with capacity ratio one (the reference server) divided by C_h. In the waiting system, when the service for a request ends in one of the servers, another request is obtained from the service waiting queue and is routed to this server.

2.2 Routing strategies

The following four routing strategies are considered, and their performances are compared.

(1) RST-based strategy

In this strategy, at the arrival of a request, the sum of the remaining service time (RST, see section 2.3) of each class request currently receiving service for each server is evaluated. An arriving request is routed to the server that satisfies the condition $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq$ SFC, and has the smallest sum of the RST.

(2) NSC-based strategy

In this strategy, at the arrival of a class-k request, the value $C_h * m_k / \left(\sum_{j=1}^{N} m_j * n_{hj}\right)$ of server h, which is called the normalized service capacity (NSC) that can be allocated to this request, is evaluated. An arriving request is routed to the server that satisfies the condition $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq SFC$, and has the largest value $C_h * m_k / \left(\sum_{j=1}^{N} m_j * n_{hj}\right)$.

(3) NNR-based strategy

In this strategy, at the arrival of a request, the value $(\sum_{j=1}^{N} n_{hj}) / C_h$ of server h, which is called the normalized number of requests (NNR) that receive service, is evaluated. An arriving

request is routed to the server that satisfies the condition $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq SFC$, and has the smallest value $(\sum_{j=1}^{N} n_{hj}) / C_h$.

For these three strategies, when plural servers with the same evaluation for the RST sum, NSC, or NNR exist, a server to which an arriving request is routed is chosen from these servers with the same probability.

(4) RAND strategy

In this strategy, the arrival request is dispatched to each server that satisfies the condition $(\sum_{j=1}^{N} m_j * n_{hj}) / C_h \leq SFC$ randomly with the probability $C_h / \sum_{l=1}^{S_n} C_l$ for server h. Here, S_n represents the server number in the system.

2.3 Extension or reduction of remaining sojourn (or service) time

At the arrival of a request to server h, the remaining sojourn time of this request is determined based on the service time of this request and the server capacity given to it. The service time is inversely proportional to the capacity ratio of server h (see section 2.1). For example, when a class-k request arrives at server h, if n_{hj} class-j requests (including the arriving request) are served in this server, $m_k / \sum_{j=1}^{N} m_j * n_{hj}$ of this server capacity is given to the request from this time, until the arrival (or departure) of the next request. The sojourn time of an arriving class-k request S_{ak} is then given by

$$S_{ak} = (S_{rk} / C_h) * \sum_{j=1}^{N} m_j * n_{hj} / m_k \ , \quad (1)$$

where S_{rk} represents the service time of an arriving class-k request in the reference server.

Moreover, at the arrival of a request, the server capacity given to each request currently receiving service decreases owing to the increase in the number of requests that share the server capacity. The ratio of the sojourn time of each request before and after the arrival of a request is equal to the inverse ratio of the server capacity given to each request before and after the request's arrival. For example, when a class-k request arrives, $m_i / \{\sum_{j=1}^{k-1} m_j * n_{hj} + m_k * (n_{hk} - 1) + \sum_{j=k+1}^{N} m_j * n_{hj}\}$ of the server capacity is given to the class-i request that receives service by this time, but from this time until the arrival (or departure) of the next request, $m_i / (\sum_{j=1}^{N} m_j * n_{hj})$ is given to a class-i request. Therefore, the remaining sojourn time S_{ni} of each class-i request after this class-k request arrives is then extended as follows:

$$S_{ni} = S_{oi} * \left(\sum_{j=1}^{N} m_j * n_{hj}\right) / \left\{ \sum_{j=1}^{k-1} m_j * n_{hj} + m_k * (n_{hk} - 1) + \sum_{j=k+1}^{N} m_j * n_{hj} \right\} \quad (2)$$

where S_{oi} is the remaining sojourn time of a class-i request immediately before this class-k request arrives.

Similarly, at the end of the sojourn time of a request, the server capacity given to requests currently receiving service increases owing to the decrease in the number of requests that share the server capacity. The ratio of the sojourn time of each request before and after the departure of a request is also equal to the inverse ratio of the server capacity given to each request before and after the request's departure. For example, for the end of the sojourn time of a class-k request, the remaining sojourn time S_{ni} of each class-i request after this class-k request departs is reduced as follows:

$$S_{ni} = S_{oi} * \left\{\sum_{j=1}^{k-1} m_j * n_{hj} + m_k * (n_{hk} - 1) + \sum_{j=k+1}^{N} m_j * n_{hj}\right\} / (\sum_{j=1}^{N} m_j * n_{hj}) \quad (3)$$

where S_{oi} is also the remaining sojourn time of a class-i request immediately before this class-k request departs, and n_{hk} does not include a departing request.

By executing these events and calculations in a simulation program, the performance of the prioritized limited multi-server PS system can be analyzed. In the simulation program, the variable time increment method, where the simulation time is omitted until the next event that causes a change in the system state occurs, such as the arrival or departure of a request mentioned above, is used in order to shorten the simulation execution time.

At the arrival of a class-k request in server h, RST for this request is calculated as S_{rk} / C_h. Then, RST reduction for this request at the outbreak of each event mentioned above can be evaluated by the duration of the omitted time from the outbreak of the previous event by $m_k / \left(\sum_{j=1}^{N} m_j * n_{hj}\right)$.

3 Some evaluation results

In the evaluation, the class-1 ($m_1 = 4$), class-2 ($m_2 = 3$), and class-3 ($m_3 = 2$) requests are assumed to be served in each server. The arrival rate, or mean service time of each priority class request, is assumed to have the same value, and to be 0.7 or 1.0. Three servers are prepared, and each server has capacity ratio 1, 0.8, and 0.6, respectively. SFC is assumed to be 20. The two-stage Erlang inter-arrival time distribution and exponential service time distribution are considered. Evaluation results are obtained as the average of ten simulation results. Approximately 140,000 requests were produced for each class in each run.

The loss probability of the class-1 request in the case of the NNR-based and RST-based strategies (0.019) is slightly smaller than that value in the case of the other strategies. On the other hand, the loss probability of the class-3 request in the case of the NSC-based strategy (0.0033) is smaller than that value in the case of the other strategies. The loss probability of the class-2 request in the NNR-based, RST-based, and NSC-based strategies shows almost the same value.

The sojourn time of the class-1 and class-2 requests in the case of the NSC-based strategy (2.27 and 2.73, respectively) is smaller than that value in the case of the other strategies. The sojourn time of the class-3 request in the case of the NNR-based strategy (3.3) is smaller than that value in the case of the other strategies. The loss probability and sojourn time in the case of the RAND strategy is larger than that value in the case of the other strategies.

In the case of the NSC-based strategy, the largest number of class-1 requests is dispatched to the server with the highest capacity. The largest number of class-3 requests is dispatched to the server with the lowest capacity. The class-2 requests are dispatched to each server in approximately the same ratio. On the other hand, in the case of the RST-based strategy, each class request is dispatched to each server in approximately the same ratio.

References

1. L. Kleinrock, "Time-Shared Systems: A Theoretical Treatment", J.A.C.M Vol.1, No.14, 242-261 (1967).
2. G. Yamazaki and H. Sakasegawa, “An optimal design problem for limited sharing systems, Management Science”, vol.33(8), pp.1010--1019 (1987).
3. E. Altman1, U. Ayesta, and B.J. Prabhu, “Load Balancing in Processor Sharing Systems”, Telecommunication Systems, June 2011, Volume 47, Issue 1-2, pp 35-48
4. H.L. Chen, J. Marden, and A. Wierman, “The effect of local scheduling in load balancing designs”, In Proceedings of IEEE INFOCOM, 2009.
5. V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt, “Analysis of join-the-shortest-queue routing for web server farms”, In Proceedings of Performance, page 180, 2007.
6. E. Altman1, U. Ayesta, and B.J. Prabhu, “Load Balancing in Processor Sharing Systems”, Telecommunication Systems, June 2011, Volume 47, Issue 1-2, pp 35-48.
7. M. Haviv and T. Roughgarden. “The price of anarchy in an exponential multi-server”, Operations Research Letters, 35:421–426, 2007.

8. Y.Shikata, W.Katagiri, and Y.Takahashi, “Prioritized Limited Processor-Sharing System with its Performance Analysis”, International Conference on Operations Research, August30 - 1, 2011 Zurich

MISTA 2015

An efficient simulated annealing for two-agent scheduling with exponential job-dependent position-based learning consideration

Jin Young Choi

1 Introduction

Two-agent single-machine scheduling problem can be found in various industrial applications where two agents compete for a single common machine to achieve their respective objectives. For example, we can consider a production system under maintenance planning. The production department (agent) wants to operate the system without any idle time in order to maximize the system utilization. On the other hand, the maintenance department (agent) calls for frequent pauses of the production system in order to reduce the number of unexpected breakdowns of the system. Therefore, two departments (agents) share the production system, while pursuing different objective functions.

Of particular interest is a two-agent single-machine scheduling problem with exponential job-dependent position-based learning effect. This means that each job has its own learning effect, implying that the learning in the production process of some jobs is faster than those of others. Moreover, the actual processing time of a job is expressed as an exponential decreasing function of learning effect and processing sequence. For example, performing similar tasks repeatedly can improve the skills of workers so that they can perform setups and handle raw materials faster, while reducing the actual processing time. This modeling concept is a plausible scenario in real-life manufacturing environment, deserving some attention.

2 Problem definition and a branch-and-bound algorithm

Two agents A and B have sets of jobs $J^A = \{J_1^A, J_2^A, \cdots, J_{n_A}^A\}$ and $J^B = \{J_1^B, J_2^B, \cdots, J_{n_B}^B\}$ to process, respectively, while competing for a single common machine. The objective of agent A is to minimize the total weighted completion time and agent B wants to keep the makespan, C_{max}^B, less than an upper bound U. Each job for agent A is assigned with a weight w_i^A and a normal processing time $p_i^A, 1 \leq i \leq n_A$. Each job for agent B has a normal processing time $p_j^B, 1 \leq j \leq n_B$. All jobs have job-dependent and

Jin Young Choi
Dept. of Industrial Engineering, Ajou University
E-mail: choijy@ajou.ac.kr

position-based learning effect, so that actual processing time of job $J_i^X, X \in \{A, B\}$ processed at the rth position in a sequence, $p_i^X(r)$, can be expressed as an exponential decreasing function $p_i^X(r) = p_i^X r^{-b_i^X}$, where $b_i^X > 0$ is a learning ratio of job J_i^X, $X \in \{A, B\}$. Then, the scheduling problem under consideration can be represented as

$$1\left|p_i^A(r) = p_i^A r^{-b_i^A}, \quad p_j^B(v) = p_j^B v^{-b_j^B}\right| \sum_{i=1}^{n_A} w_i^A C_i^A : C_{max}^B \le U, \tag{1}$$

where C_i^A is the completion time of job J_i^A.

[1] showed that $1\left|\left|\sum_{i=1}^{n_A} w_i^A C_i^A : C_{max}^B \le U\right.\right.$ is binary NP-hard. It implies that our problem in Equation 1 is at least binary NP-hard, and we need an efficient solution approach to solve it. In our work, we suggest a branch-and-bound ($B\&B$) algorithm to obtain an optimal solution, and a simulated annealing algorithm for a near optimal solution.

For a $B\&B$, we first develop four dominance properties based on a pairwise interchange comparison method as follows. Suppose that we have two schedules S_1 and S_2 such that $S_1 = (\pi, J_i^X, J_j^X, \pi')$ and $S_2 = (\pi, J_j^X, J_i^X, \pi')$, where π is a scheduled part of $(r-1)$ jobs and π' is a unscheduled part of $(n-r-1)$ jobs. Then, S_1 can dominate S_2 if (i) S_1 has smaller total weighted completion time for agent A than that of S_2, (ii) $C_l^X(S_1) \le U$ for $l \in \{i, j\}$ s.t. $J_l^X \in J^B$, and (iii) $C_j^X(S_1) < C_i^X(S_2)$. By defining $\delta_{ij}^{XX} = b_i^X - b_j^X$ and applying these conditions (i) – (iii) to four different cases in constructing S_1 and S_2, we have the following properties.

Property 1 For $J_i^X, J_j^X \in J^A$, if $\frac{w_j^A}{w_i^A}(r+1)^{\delta_{ij}^{AA}} \le \frac{p_i^A}{p_j^A} < \min\left\{1, \frac{\left[(r+1)^{b_j^A} - r^{b_j^A}\right]}{(r+1)^{b_i^A} - r^{b_i^A}}(r+1)^{\delta_{ij}^{AA}}\right\} r^{\delta_{ij}^{AA}}$, then S_1 dominates S_2.

Property 2 For $J_i^X \in J^A, J_j^X \in J^B$, if (i) $\frac{p_i^A}{p_j^B} < \frac{(r+1)^{b_j^B} - r^{b_j^B}}{(r+1)^{b_i^A} - r^{b_i^A}}\left[r(r+1)\right]^{\delta_{ij}^{AB}}$ and (ii) $t + p_i^A r^{-b_i^A} + p_j^B (r+1)^{-b_j^B} \le U$, then S_1 dominates S_2.

Property 3 For $J_i^X \in J^B, J_j^X \in J^A$, if (i) $\frac{p_i^B}{p_j^A} < \left[1 - \left(\frac{r}{r+1}\right)^{b_j^A}\right] r^{\delta_{ij}^{BA}}$ and (ii) $t + p_i^B r^{-b_i^B} \le U$, then S_1 dominates S_2.

Property 4 For $J_i^X, J_j^X \in J^B$, if $\frac{w_j^B}{w_i^B}(r+1)^{\delta_{ij}^{BB}} \le \frac{p_i^B}{p_j^B} < \min\left\{1, \frac{\left[(r+1)^{b_j^B} - r^{b_j^B}\right]}{(r+1)^{b_i^B} - r^{b_i^B}}(r+1)^{\delta_{ij}^{BB}}\right\} r^{\delta_{ij}^{BB}}$, then S_1 dominates S_2.

Moreover, we have three more feasibility properties of a sequence and one lemma to compute a lower bound, which are the components of the suggested $B\&B$ algorithm.

3 Design of simulated annealing algorithm

As an efficient near optimal solution approach, we suggest a simulated annealing (SA) algorithm. The main features of the algorithm can be described as follows.

Initial solution We first arrange the jobs for agent B ahead in generating an initial solution, to make C_{max}^B as small as possible, following the jobs for agent A.

We can consider different methods to make the partial sequences for the two agents. Specifically, we consider three methods to arrange jobs for agent A such as (i) IS_1^A = random order, (ii) IS_2^A = shortest normal processing time (SPT) order, and (iii) IS_3^A = shortest weighted normal processing time (WSPT) order. In the case of scheduling jobs for agent B, we suggest two methods to order jobs for agent B such as (i) IS_1^B = random order and (ii) IS_2^B = non-decreasing order of b_j^B. Therefore, we can consider $3 \times 2 = 6$ different methods to generate an initial solution.

Neighborhood generation For a given trial solution with the objective function value z_c, we select two jobs randomly and interchange them. If C_{max}^B is feasible, we call it next trial solution and compute its objective function value z_n. Otherwise, we reapply this procedure until we get a feasible one.

Move selection If $z_c > z_n$, we accept the next trial solution and update the current trial solution with it. Otherwise, we can accept it with the acceptance probability defined as $P_a = e^{\frac{z_c - z_n}{T}}$, where T is a control parameter, called the temperature.

Temperature schedule First, we set the initial temperature as $T_1 = c_1 \times z_c$, $(0 < c_1 < 1)$. Then, after performing a fixed number of iterations N at T_1, we decrease the value of T_1 by $(1 - c_2) \times 100\%$, represented as $T_2 = c_2 \times T_1$, $(0 < c_2 < 1)$. We can repeat this procedure for a fixed number of steps C. Hence, the temperature schedule can be expressed as $T_y = c_2 \times T_{y-1}, y = 2, 3, \cdots, C$, with N iterations at each T_y.

4 A numerical experiment and conclusions

We designed a numerical experiment to evaluate the performance of the suggested algorithms as follows. First, we considered four different values of $n = n_A + n_B$ as $n = 10, 12, 14, 16$ with $n_A = n_B$. The value of U was set by $U = (1+\alpha)U_{min} + \alpha U_{max}$, where U_{min} and U_{max} are the minimum and maximum value of the makespan that can be made using all jobs for agent B, respectively, and $\alpha (0 < \alpha < 1)$ is a real-valued parameter. We considered three different values of α as $\alpha = 0.25, 0.50, 0.75$.

Then, for each configuration of (n, α), we generated 50 problem instances and solved them using the $B\&B$ (or $SA(IS_h^A, IS_d^B), h = 1, 2, 3, d = 1, 2$. For each case, we calculated the mean, standard deviation, and maximum number of generated nodes, CPU time, and % error. The SAs showed good performance in the sense that they had low % errors in almost all configurations of (n, α) with a mean of less than 2 %. The CPU time is within 1.1 s in all configurations, that is obviously favorable over that of $B\&B$ in environments requiring real-time scheduling. Moreover, the CPU times were not affected by n by being increased linearly as n increases for a fixed value of α.

References

1. Agnetis, A., Billaut, JC, Gawiejnowicz, S., Pacciarelli, D., and Soukhal, A., Multiagent Scheduling - Models and Algorithms, Springer, 2014.

MISTA 2015

Strategic Idling and Dynamic Scheduling in Open-Shop Service Network: Case Study and Analysis

O. Baron • O. Berman • O. Berman • D. Krass • J. Wang

Abstract
In Open-shop service networks customers would like to obtain service from a set of stations, most of them without a specific order. This paper is motivated by XYZ (not the real name), a company in the healthcare service industry that operates a stochastic open shop network where the stations of the network administer medical tests that customers can take within several hours of the same day. According to senior management of XYZ there are two types of complaints about the service that customers of XYZ experience. One is with respect to the total time that customers spend in the system; the other is with respect to the long waiting time at a specific station. In fact the company believes that customers get upset when they wait more than 20 minutes for a particular station (such customers appear on the computer screen of the schedulers with red faces).

We focus in this paper on two types of service levels: the more traditional macro-level measures such as minimizing total waiting time or total system time (waiting plus service times) or minimizing total tardiness, and the "micro-level" measure of reducing excessive long waits at any individual workstation within the process. The only paper we are aware of that discusses systematically and analytically micro service level is [1] where a strategic idling (SI) scheduling policy is suggested.

The idea behind SI is that when a downstream station is very congested operating the upstream station in a normal rate may increase the congestion at the downstream station. Instead, idling the upstream station until the downstream station is less congested could be beneficial.

O. Baron
Rotman School of Management, University of Toronto
E-mail: opher.baron@rotman.utoronto.ca

O. Berman
Rotman School of Management, University of Toronto
E-mail: berman@rotman.utoronto.ca

D. Krass
Rotman School of Management, University of Toronto
E-mail: krass@rotman.utoronto.ca

J. Wang
Nanyang Business School, Nanyang Technological University
E-mail: wangjf@ntu.edu.sg

Therefore while work-conserving policies are optimal for macro-level measures, scheduling policies with SI might be helpful for the micro-level measure. In [1] we showed the benefits of SI for the two stations in tandem network where customers arrive to the network according to a Poisson process and services at the stations are exponentially distributed. In the current paper we examine whether similar ideas can be applied to a much more complex environment of a stochastic open shop network.

There was no official policy of using SI in XYZ. However using the empirical data we found statistical evidence that SI is in fact used by the schedulers to effectively manage the micro-level measure. This SI was done using only intuition of the schedulers of XYZ. We provide in this paper an efficient way to combine the SI and Dynamic Scheduling Policies (DSPs) so that the resulting policies can simultaneously address both macro- and micro-level measures.

For deciding which customer should be assigned to the next freed-up station we use 6 DSPs that include among others rules such as: "Longest System time first" and "Longest Current Waiting time first". In all of the 6 DSPs used the station that is just freed-up and has the highest remaining workload is assigned to a waiting customer.
Since the stochastic open-shop networks are very difficult to analyze analytically, we developed two simulation models. The first simulation model is based on the empirical data (ED) for arrivals and service times. We compared the micro and macro service levels for the following policies: ED; ED with no idling; the six DSPs with no idling and the six DSPs with SI. The main findings are: (1) ED with no idling results in better macro service levels than ED but with much worse micro-level performance, (2) The DSPs with no idling are much better than ED in their macro service levels but perform worse on micro-level than ED, (3) The DSPs with SI result in worse (but not by a lot) macro service levels than those without idling but are much better than the DSPs without SI and quite close to the ED policy in the micro-level performance. The second simulation model is based on randomly generated open-shop networks aiming to show benefits of using SI with DSPs for general networks. The results obtained with the second simulation are in line with those of the first model and show that combining DSPs with SI is a promising strategy in general stochastic open-shop environments.

References

1. O. Baron, O. Berman, D. Krass and J. Wang. Using strategic idleness to improve customer service experience in service networks. *Operations Research*, 62(1), 123-140. (2014).

MISTA 2015

A Matheuristic for Curriculum-Based Course Timetabling

Michael Lindahl · Matias Sørensen · Thomas R. Stidsen

1 Introduction

Finding efficient methods to schedule courses for universities has received a lot of attention from the Operations Research community and the problem exists in many variations, see the surveys Burke and Petrovic (2002); Schaerf (1999). This paper will address the problem known as Curriculum-Based Course Timetabling where the goal is to create a conflict-free timetable with all lectures planned. Conflicts occur if courses from the same curriculum or courses taught by the same teacher are planned simultaneously. The quality of a timetable is measured by the amount of violated soft constraints, for example if lectures from the same course, are scheduled in different rooms.

This problem received a lot of attention due to the second International Timetabling Competition (ITC 2007) where a formulation of the problem was provided by Gaspero et al (2007) together with a number of benchmark instances. This is the formulation and instances used to benchmark the algorithm proposed in this paper.

To solve this problem we use a combination of integer programming and heuristics. As shown by Burke et al (2008) integer programming is good at handling the hard constraints but has difficulties when the soft constraints are taken into account. A new model was formulated in Lach and Lübbecke (2012) where the problem is divided into two stages, where each stage consists of a integer program that is both smaller and easier to solve than the full model. This makes it possible to solve large instances. Mixed integer programming solvers have improved significantly over the last decade as shown by Bixby (2012), but usually do not perform as well as metaheuristics when time limits are short or the instances are large. Combining metaheuristics and integer

Michael Lindahl
Department of Management Engineering
Technical University of Denmark
E-mail: miclin@dtu.dk

Matias Sørensen
MaCom A/S
E-mail: ms@macom.dk

Thomas Stidsen
Department of Management Engineering
Technical University of Denmark
E-mail: thst@dtu.dk

programing into matheuristics has proven to work well on a large variety of problems as shown in Blum et al (2011). The purpose of the proposed matheuristic is to create a framework on top of the two stage model, which makes it possible to find high quality solutions, using less computational resources on problem of various sizes. This is to the authors knowledge the first time this is done in university course tabling.

2 Curriculum-based Course Timetabling

We use the ITC 2007 formulation of the Curriculum-based Course Timetabling from Gaspero et al (2007). The goal is to create weekly schedules for university courses by assigning each lecture to a room and a time period. Courses that are part of the same curriculum or are taught by the same teacher cannot be scheduled at the same time. Furthermore time periods are given where a teacher not is available to teach.

In order to determine the quality of a timetable a number of soft constraints are defined and the goal is to find solutions that violate these as least as possible. Each violation is associated with a number of penalty points and the objective is to minimize the sum of these points.

2.1 Mixed integer programming model

The model used as a basis for the matheuristic is the one proposed by Lach and Lübbecke (2012) that consists of two stages solved sequentially. The decomposition is exact with respect to hard constraints and thereby no feasible solutions are lost.

2.1.1 Stage I

The first stage determines at what time periods each lecture should be taught. It does not assign any rooms, but keeps track of how many rooms of different sizes are used in each time slot. This stage therefore takes all the objectives into account except for the room stability. The decision variable in this stage is defined as:

$$x_{c,p} = \begin{cases} 1 & \text{if course } c \text{ is planned at period } p \\ 0 & \text{otherwise} \end{cases}$$

The model consists of many other variables and constraints. We refer to the original article for a description of these.

2.1.2 Stage II

After each lecture has been assigned to a time period the second stage assigns a room to each lecture while minimizing the roomstability violation. This gives the following decision variable:

$$u_{c,p}v_{r,p} = \begin{cases} 1 & \text{if course } c \text{ is planned in room } r \text{ at period } p \\ 0 & \text{otherwise} \end{cases}$$

Which courses there should be planned in which time periods are constrained by the solution from stage I. The model consists of many other variables and constraints. We refer to the original article for a description of these.

3 Matheuristic

The difficult part of solving the two stage model is the first stage. The matheuristic is therefore used on this stage and the second stage is solved using the integer programming solver Gurobi directly.

The heuristic is a local search hill climber, which iteratively improves the solution. This is done by creating a neighborhood around the solution and search for an improving solution within the chosen neighborhood. A neighborhood is defined as a set of the decision variables that are allowed to change in the current iteration, others decision variables will be fixed to their value from the previous solution. This means that only a small part of the solution is altered in each iteration.

The neighborhood chooses variables that are related, this could for example be decision variables for courses that share a curriculum. Several neighborhoods with different focus are implement. The pseudocode for the algorithm can be seen in Algorithm 3.1.

Algorithm 3.1 Matheuristic

input:
    problem instance
    Set of neighbourhoods N and initial size S
output:
    Solution
$x^*_{c,p} \leftarrow$ Create Initial solution
Fix all decision variables
while stopping criteria not met **do**
    Pick neighbourhood $n \in N$ ▷ A neighbourhood is chosen at random
    $X(R)$ = NeighbourhoodCreator(n) ▷ Find connected decision variables
    Unfix variables $X(R)$
    $x^*_{c,p} \leftarrow$ Optimize sub problem
    Update S_n ▷ Evaluate the neighbourhood based on performance
end while
Solve StageII($x^*_{c,p}$)

4 Conclusion

To compare the matheuristic with state-of-the-art the ITC 2007 data sets are used.

The method is able to improve a solution more rapidly than solving the two-stage decomposition directly with Gurobi. An example of this is seen on Figure 1.

In Table 1 a comparison with the winner of the ITC2007, Müller (2009), is seen. It can be seen that mixed-integer-programming can give comparable results to using pure metaheuristics.

Because the underlying model is a MIP-model it is easy to incorporate new constraints. Because it uses a MIP solver it also takes advantage of future improvements

in this field. The method is therefore very promising as it combines problem specific knowledge with the power of MIP solvers making it possible to perform well on a large variety of instances and get solutions fast.

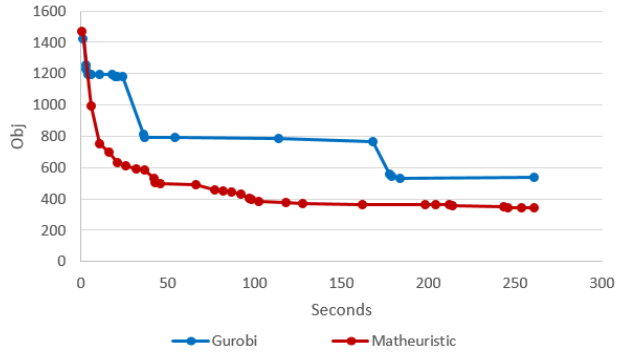


Fig. 1 An example of a run on the data set `comp05`. The matheuristic makes a lot more small improvements to the solution compared to Gurobi, which results in the matheuristic to find a better solution in less time.

Instance	Müller	Matheuristic
comp01	5.0	11.6
comp02	61.3	49.8
comp03	94.8	75.4
comp04	42.8	38.6
comp05	343.5	341.0
comp06	56.8	56.0
comp07	33.9	34.8
comp08	46.5	49.0
comp09	113.1	104.8
comp10	21.3	24.6
comp11	0.0	6.4
comp12	351.6	358.8
comp13	73.9	69.6
comp14	61.8	60.6
comp15	94.8	75.4
comp16	41.2	41.6
comp17	86.6	85.4
comp18	91.7	83.8
comp19	68.8	65.4
comp20	34.3	37.8
comp21	108.0	117.0
No. of best	9/21	12/21

Table 1 Prelimenary results: Comparison with the ITC2007 winner.

References

Bixby RE (2012) Optimization Stories, 21st International Symposium on Mathematical Programming Berlin, vol Extra, Journal der Deutschen Mathematiker-Vereinigung, chap A Brief History of Linear and Mixed-Integer Programming Computation, pp 107–121

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11(6):4135 – 4151

Burke E, Petrovic S (2002) Recent research directions in automated timetabling. European Journal of Operational Research 140(2):266 – 280

Burke E, Marecek J, Parkes A, Rudová H (2008) Uses and abuses of mip in course timetabling. In: Poster at the Workshop on Mixed Integer Programming, MIP2007, Montréal

Gaspero LD, Schaerf A, McCollum B (2007) The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. rep., School of Electronics, Electrical Engineering and Computer Science, Queens University SARC Building, Belfast, United Kingdom

Lach G, Lübbecke M (2012) Curriculum based course timetabling: new solutions to udine benchmark instances. Annals of Operations Research 194:255–272

Müller T (2009) Itc2007 solver description: a hybrid approach. Annals of Operations Research 172:429–446

Schaerf A (1999) A survey of automated timetabling. Artificial Intelligence Review 13:87–127

MISTA 2015

Solving the staff scheduling problem in a retail company

Pedro Fernandes • Armando Barbosa

1 Introduction

Throughout the years, many papers about staff scheduling and rostering problems have been published [1], [2], [3] and [4].

Due to the huge heterogeneity that can be found in the diverse sectors of activity that have to schedule employees, a considerable amount of models, approaches and solutions have been proposed [5], [6] and [7].

One study applied to the retail sector in particular can be found in [8]. It is also pertinent referring to the work presented in [9], where part-time employees are considered, although the focus of the paper is not restricted to the retail sector.

This extended abstract presents a summary of the work performed by Bullet Solutions over the last few years, which focused on solving a real staff scheduling problem in the retail sector.

2 Problem description

In the work presented in this paper, the staff scheduling problem of a large Portuguese retail company was addressed.

The company has 30 stores around the country, with a workforce of about 12.000 employees that must be scheduled, on a monthly basis.

Each store has its own group of employees, which are divided into teams (20 per store, more or less), according to their skills.

Each team has different needs throughout the day and two days with equal needs are rarely found. These needs can be mandatory (minimum needs) or normal (ideal needs).

Pedro Fernandes
Bullet Solutions, Porto, Portugal
E-mail: pedro.fernandes@bulletsolutions.com

Armando Barbosa
Bullet Solutions, Porto, Portugal
E-mail: armando.barbosa@bulletsolutions.com

The needs vary in 15 minute intervals, from an opening to a closing hour, which might also vary from team to team and from day to day. These needs can require a specific skill (or set of skills) or can be performed by anyone on the team.

Employees can have different skills (preferred and alternative) and contracts (with different workloads, constraints and benefits).

There are no daily fixed shifts (the number of employees that should be working in each interval of 15 minutes is determined by the needs) and this implies that an employee can have very diverse allocations along a week (e.g. 8 hours on Monday, starting at 9.00 AM; 6 hours on Tuesday, starting at 11.00 AM; 9 hours on Wednesday, starting at 8.00 AM, and so on). To prevent from having unmanageable schedules for employees, some restrictions can be imposed, such as a limit to the oscillation between the starting hours of each working shift on consecutive days.

In some cases an aid system between teams can be used, if one team is short on employees and another one has surpluses (according to the needs of both teams in a specific time interval). This aid can only be used if the employees have the required skills.

More than 15 different types of hard constraints (legal rules and company rules mainly imposed by the workers union) and 20 different soft constraints were identified as being used in the real case, as well as several exceptions.

The company felt that the staff scheduling process was far from being optimized, resulting in extra costs, bad service quality and frequent errors.

The time spent on the monthly scheduling activity was huge (about 20 team leaders in each of the 30 stores spending on average 3 days per month in the process; 20 team leaders * 30 stores * 3 days * 12 months = 21.600 days spent per year only in the creation of the original schedules).

Since the schedules were manually created, errors occurred frequently (serious errors, where the hard constraints were not respected and fines were applied to the company) and the introduction of changes was a painful process (normal changes such as shift swap between two employees or the substitution of an absent member of the staff), consuming extra precious time along the month.

The quality of the service presented to the customer was below expectations (bad distribution of the employees resulting in abnormal waiting times) and the entire scheduling process was not uniform among the stores and sometimes even among teams of the same store.

3 Proposed approach

The main objective of our work was solving the staff scheduling problem of the company, by creating a decision support system to encompass the entire process.

The proposed system has 4 main stages:

a) Forecast of resource working needs – staff allocation is managed according to the anticipated demand for the different times of the day, days of the week and months of the year, starting with the historical data available;

b) Automatic scheduling generation – respecting a set of hard constraints (legal or internal rules) and according to the selected soft constraints, the working schedules are generated. A sequential heuristic is used to build the initial schedules from scratch. Once the initial solution to the problem is found, the optimization phase is initiated, where better solutions are progressively searched. The search for new solutions (the improvement heuristics) is based on neighbourhood structures;

c) Intelligent manual editing – allows manual adjustments to the generated schedules, guided by an intelligent decision support system. The application provides information regarding the desired changes, returning only feasible solutions;

d) Daily monitoring of scheduling execution – facilitates the daily monitoring of the schedules' execution, enabling day to day changes, such as exchanges between employees, marking absences or replacements.

Although a complete system was developed, the main focus of our work was on the second stage of the problem – the automatic scheduling generation.

4 Preliminary results

In order to validate the developments of the different phases of the project, a test store was chosen. The choice fell on one of the biggest stores of the company that is also one of the hardest ones to deal with, due to the heterogeneity of situations that coexist and that must be addressed by the same system/algorithms.

The development of the heuristics that solve the staff scheduling problem was a real challenge, due to the huge number of rules and exceptions that exist in the company. After more than 10 months of testing and tuning, the team leaders finally felt comfortable with the scheduling results provided by the system.

At the end of the testing and tuning phase, some of the main conclusions that can be underlined are: the heuristics obtained much faster results and error free solutions (only feasible solutions are shown to the user, both in the generation process and in the posterior manipulation phase); the solutions presented by the heuristics were considered to be of better quality when compared with the manual ones (better use of the available resources, with a better distribution of the workforce according to costumers' needs, better management of the peak demands and better fulfilment of the general soft constraints of the problem).

At present, the test store is ready to "go-live" and the dissemination plan for the rest of the stores is being prepared.

We intend to present, at the Conference, details about the proposed model (hard and soft constraints involved, fundamental concepts and exceptions, among other useful information), the heuristics conceived to solve the problem and the main results of the project. At the time, we also expect to have additional information to share (real scenarios and results), at least from the store that will be already using the system.

References

1. J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester and L. De Boeck, "Personnel scheduling: A literature review", *European Journal of Operational Research*, vol. 226, no. 3, pp. 367–385 (2013)
2. H. K. Alfares, "Survey, Categorization, and Comparison of Recent Tour Scheduling Literature", *Annals of Operations Research*, vol. 127, no. 1–4, pp. 145–175 (2004)
3. E. K. Burke, P. De Causmaecker, G. Vanden Berghe and H. Van Landeghem, "The State of the Art of Nurse Rostering", *Journal of Scheduling*, vol.7, no. 6, pp. 441–499 (2004)
4. B. Cheang, H. Li, A. Lim and B. Rodrigues, "Nurse rostering problems – a bibliographic survey", *European Journal of Operational Research*, vol. 151, no. 3, pp. 447–460 (2003)
5. A.T. Ernst, H. Jiang, M. Krishnamoorthy and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models", *European Journal of Operational Research*, vol. 153, no. 1, pp. 3–27 (2004)
6. P. Brucker, R. Qu and E. Burke, "Personnel scheduling: Models and complexity", *European Journal of Operational Research*, vol. 210, no. 3, pp. 467–473 (2011)
7. A. Meisels and A. Schaerf, "Modelling and Solving Employee Timetabling Problems", *Annals of Mathematics and Artificial Intelligence*, vol. 39, no. 1–2, pp 41–59 (2003)
8. Ö. Kabak, F. Ülengin, E. Aktaş, Ş. Önsel and Y. I. Topcu, "Efficient shift scheduling in the retail sector through two-stage optimization", *European Journal of Operational Research*, vol. 184, no. 1, pp. 76–90 (2008)
9. M. Hojati and A. S. Patil, "An integer linear programming-based heuristic for scheduling heterogeneous, part-time service employees", *European Journal of Operational Research*, vol. 209, no. 1, pp. 37–50 (2011)

MISTA 2015

Ant Colony Optimisation for a Job Shop with Flexible Maintenance

Ivar Struijker Boudier · Kevin Glazebrook · Mike Wright · Paul Jennings

1 Introduction

Our scheduling problem is motivated by the scheduling challenges encountered at a particular facility in the nuclear power industry. The work passing through this facility can be modelled as a job shop problem, with a number of additional non-standard features. Existing research which does take into consideration such non-standard features tends to consider a single complication in isolation.

We have developed an ant colony optimisation algorithm for a generalisation of the job shop scheduling problem. It deals with a number of the non-standard features faced by schedulers at the aforementioned facility, including: flexible maintenance scheduling; jobs which merge into one, or split into many; jobs with release dates; jobs with precedence constraints; and the rescheduling of a schedule in progress, either in response to a machine breakdown, or to include new jobs which have become available.

2 Problem Outline

The problem is an extension of the standard job shop scheduling problem. We wish to schedule a fixed set of jobs. Each job consists of a number of operations to be processed in a given order. Each operation has to be carried out on a specified machine. Operations are assumed to have a known, deterministic processing time. It is assumed

Ivar Struijker Boudier
Lancaster University
E-mail: i.struijkerboudier1@lancaster.ac.uk

Kevin Glazebrook
Lancaster University
E-mail: k.glazebrook@lancaster.ac.uk

Mike Wright
Lancaster University
E-mail: m.wright@lancaster.ac.uk

Paul Jennings
The National Nuclear Laboratory
E-mail: paul.jennings@nnl.co.uk

each machine can process only one operation at a time. Operations are assumed to be non-preemptive. Each job has a release date. Jobs can have precedence constraints. Jobs have soft and hard due dates.

Each machine has a preventive maintenance programme. These flexible maintenance activities must start within a specified time window. The start and end of this time window are the soft and hard due date, respectively, for the start of the maintenance activity. If maintenance is not started by its hard due date, the affected machine must be shut down. This is due to strict safety regulations in the nuclear industry. Each maintenance event is scheduled as a single-operation job.

The jobs have soft and hard due dates. A penalty is incurred if the final operation of a job is not completed before its soft due date. The size of the penalty increases with the delay, and rises rapidly beyond the hard due date.

Some jobs can only start when multiple other jobs come together on completion. There are also jobs which, on completion, allow a number of other jobs to start. Precedence constraints are used to schedule jobs which merge or split like this.

The flexible maintenance activities receive penalties based on their scheduled starting time. Maintenance can start no earlier than its soft due date. If maintenance has not started by its hard due date, the relevant machine must be shut down. Therefore a missed hard due date for maintenance is penalised much more heavily than a missed hard due date for jobs. The algorithm aims to minimise the overall penalty cost. The penalty structure encourages the following ordering of priorities, starting with the highest: Schedule maintenances to start before their hard due dates; Schedule jobs to finish before their respective hard due dates; Reduce any remaining delay of jobs and maintenance activities.

Each of our test problems contain 20 jobs (some with merges or splits) and 10 maintenance activities, for a total of 102 operations. This is in the order of magnitude of the sets of work being scheduled at the facility.

3 Ant Colony Optimisation for the Job Shop Problem

In this section, 'job' refers to both jobs and maintenances. Ant colony optimisation (ACO) [2] algorithms construct a large number of solutions simultaneously, and this is repeated over many iterations. ACO algorithms build solutions one step at a time. For our scheduling algorithm this means that the operations are sequentially inserted into a partial schedule, until the schedule is complete. The operations are inserted according to probabilistic rules, which depend on the heuristic information η_{ij} and the pheromone level τ_{ij}. If operation i is the most recently inserted operation, then the probability p_{ij} that job j is progressed next is given by

$$p_{ij} = \frac{{\tau_{ij}}^{\alpha}{\eta_{ij}}^{\beta}}{\sum_{j \in \mathcal{N}_i} {\tau_{ij}}^{\alpha}{\eta_{ij}}^{\beta}} \quad \text{for } j \in \mathcal{N}_i, \quad \alpha, \beta \geq 0, \qquad [2]$$

where $\mathcal{N}_i$ denotes the set of jobs available at this stage of the schedule construction. A job is not available for scheduling if its precedence constraints are not satisfied, or if the job has been completed. A job may have a future release date, or may currently be busy, but this does not make it unavailable for scheduling. To ensure feasibility, the operation to be added to the partial schedule next is not chosen directly. When

operation i has just been added, the algorithm decides which job $j \in \mathcal{N}_i$ to progress next. The next operation is then defined to be the next available operation of job j.

The heuristic information η_{ij} for travelling salesman problems is often chosen to be the inverse of the distance from the current city i to each unvisited city j. In the case of our job shop, the distance from operation i to job j, d_{ij}, can be set as the length of time until the next operation of job j could be started, given the restrictions imposed by the current partial schedule. An operation can be started when its preceding operation (if any) has been completed, when the required machine is available and when the job's release date has been met. Therefore we define d_{ij} to be the difference between the latest of these three times, and the current time in the schedule construction.

Initially we used $\eta_{ij} = \exp\left(-d_{ij}^{+}\right)$, which can handle the case in which the distance is zero. Better results are obtained when the remaining slack of job j is also considered. The slack is the difference between the remaining time until the job's soft due date and the time required to complete it. In our heuristic information, the contribution of slack is weighted by the contribution of the distance. If slack is not weighted by the distance, the algorithm has the undesirable property of jumping to jobs which are not available for a long time, if their slack happens to be small. Our heuristic information is then as follows:

$$\eta_{ij} = \exp\left(-d_{ij}^{+}\right) \times \left\{1 + \exp\left(-\text{slack}_{j}^{+}\right)\right\}.$$

The pheromone levels τ_{ij} contain information on how rewarding it has previously been to insert an operation belonging to job j immediately after inserting operation i. All pheromone levels are equal initially and updated at the end of each iteration. Pheromone updating involves the standard ACO procedure of global evaporation, followed by both rank-based and elitist pheromone deposits [1].

4 Future Work

Our algorithm is already capable of solving problems with a number of non-standard features which have often been considered in isolation. In its current form it is implicitly assumed that there is infinite queuing capacity for jobs at each machine. Since storage capacity is very limited at the facility for which the algorithm is being developed, our research will now focus on how to modify the existing algorithm to produce feasible schedules in a no-storage scenario. The problem can then be modelled as a blocking job shop, with all the non-standard features described above. Rescheduling of partially completed schedules, in response to machine breakdowns or the arrival of new jobs, is also being considered.

Acknowledgements This research is jointly funded by the Engineering and Physical Sciences Research Council (EPSRC) and the National Nuclear Laboratory (NNL).

References

1. Bernd Bullnheimer and Richard F. Hartl and Christine Strauß, A New Rank Based Version of the Ant System - A Computational Study, Central European Journal for Operations Research and Economics, 7, 25-38 (1997)
2. Marco Dorigo and Thomas Stützle, Ant Colony Optimization. MIT Press, Cambridge Massachusetts (2004)

MISTA 2015

Complexity of minimizing the total flow time on parallel machines with interval data and minmax regret criterion

Maciej Drwal

1 Introduction

In this paper we apply the minmax regret approach to one of the basic multi-processor scheduling problems. We focus on the problem of scheduling on parallel identical machines to minimize the total flow time (i.e., the sum of completion times of all jobs) when the job processing times are uncertain, and known only to be contained within fixed intervals.

Since in practical applications the input data to most problems can be rarely given precisely, many researchers in combinatorial optimization proposed a robust solution approach. In the context of scheduling, very often processing times of tasks are not known in advance, but their values may fluctuate within certain bounds. In such settings, we aim to determine the solution that performs satisfactorily even in the worst possible scenario of events. One widely adopted approach is to minimize the maximum deviation of a solution from the optimum, which is also called the maximum regret criterion [1], [2], [3]. When the parameter uncertainty may have significant impact on the outcome of optimization, this approach appears to be very useful alternative to the traditional methods. Unfortunately, robust solutions are often more difficult to find than their deterministic counterparts.

The problem under consideration is denoted INTERVAL $P||\sum C_i$, using the standard scheduling theory notation [4], combined with a prefix indicating the uncertain version of the optimization problem. In the absence of the parameter uncertainty, this problem can be solved in polynomial time, e.g., by applying the *shortest processing time first* rule (sorting all jobs by processing times in nondecreasing order). However, its minmax regret version becomes NP-hard even for a single machine, i.e., INTERVAL $1||\sum C_i$. In [5] it is shown that even when midpoints of all intervals are equal and the number of jobs is odd, then finding the optimal robust sequence on a single machine is weakly NP-hard (surprisingly, for the even number of jobs this problem is polynomially solvable). Thus the case in which the number of machines is given as a part of the input can be no easier. Since some of the weakly NP-hard problems may admit practically efficient algorithms,

Maciej Drwal
Department of Computer Science, Wroclaw University of Technology, Wroclaw, Poland
E-mail: Maciej.Drwal@pwr.edu.pl

and since the instances used in the complexity proofs seem to be quite restricted, the important question is whether the general case of the problem is strongly NP-hard.

This problem remains open for the single machine case. Recently, Conde [6] indicated a simple reduction from the minmax regret assignment problem [7] of m jobs to m machines, which implies that in case of m parallel unrelated machines (INTERVAL $R||\sum C_i$) the problem is strongly NP-hard. Unfortunately, this reduction no longer applies to the case of parallel identical machines, since in that settings the assignment becomes trivial. In this paper we extend the aforementioned complexity results, showing that strong NP-hardness occurs even for identical machines. Moreover, we indicate that the parallel machines case has a strong resemblance to the single machine case.

2 Problem formulation

An instance of the considered scheduling problem $P||\sum C_i$ is given by the integer n, denoting the number of jobs, the integer m, denoting the number of machines (processors), and the set of processing times of each job: integers p_i for $i = 1, \ldots, n$. Each job has to be assigned to exactly one machine. Let π_j denote a vector of n_j integers, where $\pi_j(k)$ is the index of job scheduled on jth machine as kth to the last (jobs on each machine are scheduled starting from time zero and without idle times). The *completion time* of job is: $C_{j,k} = \sum_{i=k}^{n_j} p_{\pi_j(i)}$ ($C_{j,k} = 0$ if there is no such job). The objective is to minimize the sum of completion times (also called *total flow time*), expressed as:

$$F(\pi) = \sum_{j=1}^{m} \sum_{k=1}^{n_j} C_{j,k} = \sum_{j=1}^{m} \sum_{k=1}^{n_j} k p_{\pi_j(k)}, \tag{1}$$

where $\pi = [\pi_1, \ldots, \pi_m]$ is called a *schedule*. We will sometimes refer to this problem formulation as a *deterministic* version of the scheduling problem.

The definition of minmax regeret version of this problem with interval uncertainty, denoted INTERVAL $P||\sum C_i$, differs in that, instead of exact processing times, we are given only intervals $[p_i^-, p_i^+]$, $i = 1, \ldots, n$, to which the actual processing times belong. Denote by $S = [p_1^S, \ldots, p_n^S]$ any vector that satisfies $p_i^- \leq p_i^S \leq p_i^+$ for all $i = 1, \ldots, n$. Such a vector will be called a *scenario*. For any schedule and scenario we define the value of *regret* as: $Z(\pi, S) = F(\pi, S) - F^*(S)$, where $F(\pi, S)$ is the objective function (1) from the deterministic version of problem $P||\sum C_i$ with input data S, and $F^*(S)$ is the value of optimal solution of this problem. The objective of INTERVAL $P||\sum C_i$ is to minimize over schedules the maximum of regret over scenarios:

$$Z^* = \min_{\pi} \max_{S} \left(F(\pi, S) - F^*(S) \right).$$

A schedule that minimizes the maximum regret will be called *robust optimal*. An optimal solution of deterministic version of the problem, obtained by taking a worst-case scenario for π as an input data, is called the *worst-case alternative* for π.

3 Main results

The maximum regret $Z(\pi) = \max_S F(\pi, S)$ can be computed in polynomial time. The method is based on constructing appropriate instance of assignment problem on

bipartite graph with jobs in one partite set and positions on machines in the other one. The construction is very similar to the one presented in [6] for unrelated machines case, with the main difference that in the identical machines case the input data contains a single interval $[p_i^-, p_i^+]$ in place of m intervals given in unrelated machines case.

Before we state our main theorem, we need to develop some auxiliary results. In this extended abstract, we omit proofs of all lemmas, giving only the general idea of our method. From now on we consider only instances with n jobs and m machines where $m|n$, that is, there exists integer $n_0 > 1$, such that $n = n_0 m$. In particular, it is a known result for deterministic version of the considered problem, that any optimal schedule has always the numbers of jobs on each machine that differ by at most 1. It turns out that this extends to the maximum regret version.

Lemma 1 *If $m|n$ then in optimal robust schedule every machine is assigned the same number of jobs.*

Next, we exploit the fact that in robust optimal solutions, for any job the choice of machine is irrelevant: it is only required to select an appropriate position on any machine.

Lemma 2 *Let π be a schedule, where π is a $m \times n_0$ matrix (each ith column contains jobs scheduled as ith to the last on their respective machines). Consider a schedule π' obtained by switching a pair of elements in any column of π. Both schedules have the same maximum regret.*

Observe that the formulation of the considered problem remains valid when bounds of intervals p_i^- and p_i^+ are arbitrary (possibly negative) integers, and that adding the same constant to all bounds of intervals does not change the value of maximum regret. Consider instances with equal midpoints, that is $[p_i^-, p_i^+] = [-p_i^+, p_i^+]$. Consider a single machine j. Due to [5] we know that if the number of jobs on a machine is odd, $n_j = 2k_j + 1$, then optimal solution can be obtained as:

$$Z_j(\pi^*) = k_j \sum_{i=1}^{n_j} p_i^+ + \max\{P_1, P_2\}, \tag{2}$$

where (P_1, P_2) is a solution of the optimization version of the PARTITION problem (i.e., P_1 and P_2 are sums of two disjoint subsets of $2k_j$ smallest jobs, and the value $|P_1 - P_2|$ is minimal among all such 2-partitions). The job with the widest uncertainty interval is always inserted in the middle of the permutation and does not appear in P_1 or P_2. The remaining jobs are scheduled in such a way that the wider the interval, the closer it is to the middle of the permutation.

The following lemma states that it is always possible to permute columns of an optimal robust solution π and its worst-case alternative σ, to obtain an equivalent robust optimal solution, which in fact consists of m solutions of INTERVAL $1||\sum C_i$ problem.

Lemma 3 *There exists optimal robust schedule of* INTERVAL $P||\sum C_i$*, such that for any machine $j = 1, \ldots, m$, the schedule on machine j is the same as the optimal robust schedule in problem* INTERVAL $1||\sum C_i$.

The main theorem is based on the reduction from a variant of set partitioning problem (related to 3-PARTITION problem [8]). We define problem 4-PARTITION-INTO-PAIRS (problem 4-PP for short) as follows: given is a set of $4m$ positive integers a_i, $i = 1, \ldots, 4m$. The question is whether it is possible to partition the given set of integers into m disjoint quadruplets of integers $A_1, \ldots, A_m$, such that there exists a bijective function $f : \{1, \ldots, m\} \to \{1, \ldots, m\}$, such that:

$$\forall_{i \in \{1,\ldots,m\}} \; s(A_i) = s(A_{f(i)}) \;\text{ and }\; f(i) \neq i \;\text{ and }\; f(f(i)) = i,$$

where $s(A)$ is the sum of elements in A. In other words, we want to partition the set of integers into m 4-sets in such a way that all the 4-sets can be arranged in distinct pairs of equal sums. It can be shown that 4-PP is strongly NP-complete.

Theorem 1 *Problem* INTERVAL $P||\sum C_i$*, in which the number of machines is given as a part of the input, is strongly NP-complete.*

The sketch of a proof is as follows. We reduce an instance of 4-PP to INTERVAL $P||\sum C_i$ problem with $m/2$ machines and $n = 4m + m/2$ jobs. For each integer a_i we create a job with processing interval $[-a_i, a_i]$, and, additionally, we create $m/2$ jobs with processing intervals $[-B, B]$, where $B > \max a_i$. From Lemma 3 and equation (2) we know that each machine with 9 jobs gives optimal maximum regret:

$$Z_j(\pi_j) = 4(\sum_{i=1}^{9} a_{\pi_j(i)}) + \max\{s(A_{j1}), s(A_{j2})\},$$

where $s(A_{j1}) = a_{\pi_j(1)} + a_{\pi_j(2)} + a_{\pi_j(3)} + a_{\pi_j(4)}$, and $s(A_{j2}) = a_{\pi_j(6)} + a_{\pi_j(7)} + a_{\pi_j(8)} + a_{\pi_j(9)}$. It is enough to show that the instance of 4-PP has a solution if and only if $Z(\pi^*) = 4mB + \frac{9}{2}\sum_{i=1}^{4m'} a_i$. This can be seen by first noticing that exactly one job $[-B, B]$ must be assigned to each machine in the middle of job sequence (fifth position). Then the minimal maximum regret on each machine is obtained when both four-job subsets on the left and the right of the middle job have equal sums (computed by taking upper bounds of intervals $[-a_i, a_i]$).

4 Conclusions

In this extended abstract we showed that one of the easiest scheduling problems on parallel machines becomes strongly NP-hard when interval uncertainty in the input data is taken into consideration, and the minmax regret criterion is used to define a robust solution. This suggests that in order to handle uncertainty in practice, either less restrictive notion of robustness should be applied, or approximation algorithms and heuristics should be used. The design of efficient approximation algorithms for scheduling problems with uncertain parameters is the subject of further work.

References

1. P. Kouvelis, G. Yu, Robust discrete optimization and its applications, Springer, 1997.
2. H. Aissi, C. Bazgan, D. Vanderpooten, Min–max and min–max regret versions of combinatorial optimization problems: A survey, European Journal of Operational Research 197 (2) (2009) 427–438.

3. A. Kasperski, Discrete optimization with interval data: minmax regret and fuzzy approach, Springer, 2008.
4. R. Graham, E. Lawler, J.K. Lenstra R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 5 (1979) 287–326.
5. V. Lebedev, I. Averbakh, Complexity of minimizing the total flow time with interval data and minmax regret criterion, Discrete Applied Mathematics 154 (15) (2006) 2167–2177.
6. E. Conde, A MIP formulation for the minmax regret total completion time in scheduling with unrelated parallel machines, Optimization Letters 8 (4) (2014) 1577–1589.
7. H. Aissi, C. Bazgan, D. Vanderpooten, Complexity of the min–max and min–max regret assignment problems, Operations Research Letters 33 (6) (2005) 634–640.
8. M. Garey, D. Johnson, Computers and intractability. W.H. Freeman, 2002.

MISTA 2015

Scheduling the operation of a phosphate pipeline for OCP: A Case Study

Abstract

Kris Van Marcke • Osman Ali

1. Introduction

The scheduling problem described is part of a large and highly strategic project for OCP, the world's largest phosphate miner and producer. In the context of a supply chain optimization project Ordina undertakes for OCP, it also built a scheduler for scheduling operations of a pipeline that is transporting rough phosphate slurry coming from the mines to the plants located at the coast over a distance of 187 kilometers. The scheduling algorithm uses a multi-agent system metaphor.

2. Case Description

2.1. Situating OCP and the pipeline

OCP is the world's largest exporter of phosphate and phosphoric acids; and one of the largest exporters of phosphoric fertilizers. With a total production of 24 million tons per year, exported to over 40 countries it accounts for 25% of the Moroccan export. OCP exploits 4 mines (Khouribga, Benguerir, Youssoufia, Boucraa), 2 chemical plants (Jorf Lasfar, Safi), and exports from 4 ports (Casablanca, Jorf Lasfar, Safi, Lâayoune).

The new pipeline is taken into production between Khouribga and Jorf Lasfar. Four mining sites are connected via secondary pipes to one head station. From the Khouribga head station, a primary pipe transports the phosphate directly to the chemical plant from Jorf Lasfar 187 kilometers further. The total pipe length is 236 kilometers.

Kris Van Marcke
Ordina NV
E-mail: kris.vanmarcke@ordina.be

Osman Ali
Ordina NV
E-mail: osman.ali@ordina.be

2.2. Summary of the Supply Chain Optimization Project DLP

DLP aims at the global optimization of the downstream logistics supply chain project at OCP. It is a complex balancing exercise between *sales and sales commitments* to customers, *maritime activities* including vessel allocation, laycan allocation, docking, loading, …, the *production plan* in the chemical plants (which product will be produced when), the *production plan* in the mines (which phosphate quality to mine and when), *storage capacity* (in mines, plants, ports) and *transport plans* (between mines and ports, mines and plants, and plants and ports).

The main DLP optimization target is 'fulfillment': maximize the total volume shipped. The main bottleneck is logistics: the loading capacity in the ports, the storage capacity and the transportation capacity. All volumes must be planned to arrive just in time at the quay as the storage capacity in the port is very low. Mining capacity is virtually unlimited and OCP is increasing its production capacity yearly.

This planning puzzle is resolved by means of a linear program, which is executing the following planning decisions: allocation of laycans to vessels; when a vessel will dock, be loaded, sail; where to store finished goods and how to transport them just-in-time to the quay; when to produce which quality of finished goods; when and how to receive which quality of phosphate from the mine; how to transport it (train, pipe); how/when to import additional primary materials (like ammoniac, sulfur) via the port.

2.3. The Scheduling Case

The pipeline assures a continuous phosphate flow between the four mine sites of Khouribga and the chemical plant in Jorf Lasfar. In addition, it enables OCP to move from an isolated make-to-stock mining facility towards an integrated make-to-order approach with different phosphate qualities to be mined.

The scheduler assures that the OCP production plan receives the right volume of phosphate, with the right phosphate quality level (BPL), with the right flow rate, on the right day.

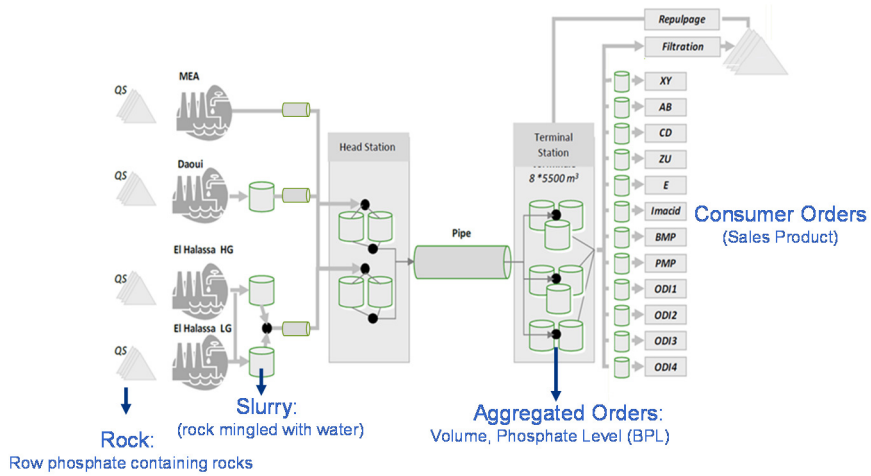


Figure 1: Product flow

There are four mining sites and hence four stocks of phosphate ore. In the four 'wash plants' different quality levels can be obtained. The result is phosphate slurry, i.e. phosphate mingled with water. Three of the four wash plants have local storage capacity for slurry. There are four secondary pipes to transport the slurry to the head station, where it can be stored in two distinct tank groups. From the head station the slurry is transported via the primary pipe to the terminal station, where it can be stored in three different tank-groups. From there it is used for consumer orders. The described product flow is depicted in figure 1.

The wash plant can control the quality level of its output by means of three different operations: washing (L), floating (F) and grinding (B). By correctly scheduling those three operations the correct output quality can be obtained. In the head station the feed balance between the slurry from the three input pipes is decided in order to get the correct quality into the primary pipe with the expected throughput.

3. The Scheduler

3.1. The Scheduling Objective

The input for the scheduler consists of the consumer demand (volume per quality plus input flow requirements) on a day by day basis; forecasted stock levels for the four mine sites, predicted unavailabilities, and monthly recipe objectives (defined by the central planning board).

A recipe is a concept that encompasses the different planning decisions to be taken. For a given output product quality and density, it defines a mix of input products from the different wash plants with a ratio, flow-rate and production steps. By defining monthly recipe objectives the central planning board can steer the scheduler to be in-line with the mining plan.

The scheduler is expected to choose the recipe and to schedule the corresponding batches on the different units (wash plants, tank-groups, pipes).

The scheduler takes into account the aggregated orders (customer orders are aggregated per day and per quality level) and BPL, the available/predicted volumes, objectives per recipe, set-up times and MES events. MES events provide real-time information about measured volumes, flow-rates and quality-levels. When real-time information deviates from the planned situation, the scheduler must be able to adapt the schedule to compensate the deviated values.

3.2. The Scheduling Algorithm

The scheduling algorithm is based on a multi-agent system (MAS) approach that uses a model of the process under consideration and coordinating agents to organize the search [1]. The approach is typically suited for scheduling of operations and resource allocation.

The MAS is structured according to the PROSA reference architecture [2]. It makes use of 3 scheduling agent types (agents representing all fragments of the scheduling problem) and 2 search agent types (agents that organize the search).

Every order to be scheduled is represented by an *order agent*. The order agent has the know-how to assess a solution with respect to completeness and quality. Every resource that can be allocated is represented by a *resource agent*, which manages the resource's local schedule. *Product agents* correspond to a process, i.e. they represent valid sequences of process steps or operations together with their resource requirements. Product agents possess the necessary know-how to determine, given a partial solution for an order, what are the next actions that must be undertaken to make the solution more complete.

Search agents consist of exploring agents and intention agents. *Exploring agents* are delegates of order agents, generated at regular intervals. Exploring agents interact with product agents and resource agents to construct a valid solution – for the order agent they work for – and ask for the availability of resource capacity. When sufficient solutions are explored, the order agent picks the most suited. Next, the order agents creates *intention agents* who go out to contact the resource agents to reserve the necessary capacity of resources and get an update on the execution constraints. When the resources turn out to be no longer available, the intention agent fails to finalize this selected solution. New solutions will be constructed by different exploring agents and at some point in time the order agent will select a new solution and re-generate intention agents for reserving capacities on the resources indicated in the new solution.

3.3. Application of the planning approach

The MAS algorithm is applied in the following way. Customer orders are grouped per product and day into aggregated orders. An order agent is created for every aggregated order. For every unit (wash plant, tank group, pipe) in the system, a resource agent is created.

For an aggregated order, there are between 5 and 15 possible routes, depending on the available recipes. Each recipe is represented by a product agent.

The aggregated orders are sorted according to priority. The corresponding order agents generate a sequence of exploring agents that construct a series of possible solutions in coordination with product agents and resource agents. When there are sufficient solutions available the order agents evaluate the solutions and select one. Next one intention agent is generated to interact with the resource agents to make the final resource capacity reservations.

Organizing the MAS like this generates an eager local search mechanisms: the orders are planned in sequence of priority and they block resource capacity before the next order gets a chance. This local search configuration gives satisfactory results and very rapidly.

Different search approaches can be imagined just building upon the same metaphor. Imagine the orders are not a priori sequenced, and all order agents launch their exploration delegates at the same time. Many solutions will be building-up, until one order agent decides that one solution is satisfactory and starts blocking the resources. At that point some of the other exploration agents have constructed solutions that are no longer feasible or satisfactory. Resource agents should give agents that have already asked for capacity an update on availability and timing. Exploration agents may decide to continue or to fail depending on the decision whether the plan built-up so far is still valid.

This solution is still an eager local search, but one that no longer depends on a-priori sequencing of the orders. The knowledge is now entirely encapsulated in the knowledge of the agents. The quality of the result is now depending on the quality of the acceptance function, i.e. when does an order agent finds a solution satisfactory. If it accepts a solution too easily, certain resources may be reserved for a solution that is not a very good global solution after all. By giving order agents information on their own priority in the overall search space, lower priority agents may postpone their decisions a bit so that the high priority agents get a better shot.

4. Conclusions

The scheduling case described in this paper is part of a pipeline constructing value with high strategic impact for OCP. The economical stakes are huge, and the stability and reliability of the scheduler are crucial.

The scheduling algorithm has been configured using a MAS metaphor, in such a way that it implements an eager local search process. This configuration gives good results very rapidly. We have argued that the same metaphor can be configured in slightly different ways to realize different search processes, which will always converge rapidly to a quality solution.

Therefore, in conclusion, we believe

- that an agent metaphor can be used for solving scheduling problems,
- that it is applicable in highly critical real live applications,
- that it converges rapidly to good results,
- that is it applicable to solve real-world closed-loop planning problems,
- that it opens perspectives for obtaining different search process on top of the same model.

References

1. O. Ali, et.al, Towards online planning for open-air engineering processes, Computers in Industry (64), issue 3, 242-251 (2012)
2. H. Van Brussel, et. al, Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry, (37), 255-274 (1998)

MISTA 2015

Online and semi-online scheduling of two job types on a set of multipurpose machines

Shlomo Karhi · Dvir Shabtay

1 Introduction

We study a set of scheduling problems, where $q = 2$ job types are to be processed on a set of m ($m \geq 2$) identical multipurpose machines. This set of problems can be formally presented as follows. A set of n jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ is to be processed non-preemptively on a set of m parallel machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each job J_j ($j = 1, ..., n$) is characterized by its processing time parameter, p_j, and by a subset $\mathcal{M}_j$ of machines ($\mathcal{M}_j \subseteq \mathcal{M}$) to which it can be assigned (subset $\mathcal{M}_j$ is referred to as the *processing set* of job J_j). We focus on the case where there are only two possible $\mathcal{M}_j$ subsets. The first is for jobs of type 1, while the second is for jobs of type 2. Let t_j be the type of job J_j ($t_j \in \{1, 2\}$). Without loss of generality, we assume that if job J_j is of type 1 ($t_j = 1$) then $\mathcal{M}_j = \{M_1, ..., M_k\}$ and if if job J_j is of type 2 ($t_j = 2$) then $\mathcal{M}_j = \{M_{s+1}, ..., M_m\}$, where $s < k \leq m$ and $s \geq m - k$. Our objective is to schedule the jobs to minimize either the *makespan* ($C_{\max} = \max\{C_1, ..., C_n\}$) or the *total completion time* (ΣC_j), where C_j is the completion time of job J_j for $j = 1, ..., n$. Following the traditional three-field notation our problem can be referred to by $P\left|\mathcal{M}_j, q = 2\right| F$ ($F \in \{C_{\max}, \Sigma C_j\}$) and an instance for this problem is defined by n, m, s, k the processing time p_j, and the type $t_j \in \{1, 2\}$ of each job $J_j \in \mathcal{J}$.The literature on multipurpose machine scheduling can be divided into two major streams: offline and online scheduling, depending on the knowledge that the scheduler has prior to making any scheduling decision. In contrast to *offline scheduling* where the scheduler has access to the entire instance of the problem prior to making any scheduling decision, in online scheduling the scheduler does not have this ability. Accordingly, for the online variant of our $P\left|\mathcal{M}_j, q = 2\right| F$ problem there is *uncertainty* about the number of jobs to be processed (n), their processing time (p_j for $j = 1, ..., n$) and type (t_j for $j = 1, ..., n$).

We study several variants of the online version of the $P\left|\mathcal{M}_j, q = 2\right| F$ problem ($F \in \{C_{\max}, \Sigma C_j\}$), assuming it follows the *online-list* paradigm, where the jobs are ordered in a list and are presented to the scheduler one by one. As soon as a job is presented to the scheduler he knows its processing time and type. According to the online-list paradigm, the scheduler has to assign the jobs by using an *online algorithm*,

e-mails: shlomo.karhi@biu.ac.il and dvirs@bgu.ac.il

Address(es) of author(s) should be given

where each job has to be assigned to some machine before the next job is presented and the assignment is irreversible.

We use the *competitive analysis* evaluation technique presented by Sleator and Tarjan [4] to evaluate the performance of online algorithms. For an input instance $\mathcal{I}$, let $F^A(\mathcal{I})$ denote the objective value produced by an online algorithm A and $F^*(\mathcal{I})$ denote the corresponding minimum objective value determined by an optimal offline algorithm OPT. According to the *competitive analysis* evaluation technique, Algorithm A is called ρ-*competitive* if there exists some nonnegative value v, independent of the instance, such that $F^A(\mathcal{I}) \leq \rho F^*(\mathcal{I}) + v$ for any input instance $\mathcal{I}$. Moreover, the *competitive ratio* of algorithm A, denoted by ρ_A, is the infimum of ρ such that A is ρ-competitive. We say that an online scheduling problem has a lower bound ρ if no online algorithm has a competitive ratio smaller than ρ, and an online algorithm is called optimal if its competitive ratio matches the lower bound of the problem.

2 Literature review and motivation

Both Park *et al.* [3] and Jiang *et al.* [1] provide an online algorithm with a competitive ratio of 5/3 for the special case of the $P2\left|\mathcal{M}_j, q=2\right|C_{\max}$ problem with $s=1$ and $k=m=2$. Jiang [2] study the special case of the $P\left|\mathcal{M}_j, q=2\right|C_{\max}$ problem, where $k=m$. He show that an algorithm that assigns each job J_j to the least loaded machine in $\mathcal{M}_j$ has a competitive ratio of $4-1/m$. Moreover, he present an alternative algorithm with a competitive ratio of $\frac{12+4\sqrt{2}}{7} \approx 2.522$, and prove that the problem has a lower bound of 2 on the competitive ratio of any online algorithm. The main disadvantage of the results obtained by Jiang [2] is that a constant competitive value is used, rather than one which fits itself to the exact values of s and m. Zhang *et al.* [5] overcome this disadvantage by providing an algorithm with a competitive ratio of

$$\rho = 1 + \frac{m(m-1)}{m(m-s)+s^2}. \tag{1}$$

Note that the value in (1) is less than $\frac{7}{3}$ for any $s \leq m$ and thus the competitive result obtained by Zhang *et al.* [5] is much better than that obtained by Jiang [2]. Zhang *et al.* also improve the lower bound presented by Jiang [2] by providing several lower bounds, each for a different set of s and m values.

Zhang *et al.* [5] results are restricted to an inclusive processing set structure (where $k=m$). Moreover, they consider only the case of arbitrary processing times and it might be that for more restricted processing times models, algorithms with better competitive ratio and tighter lower bounds can be computed. Thus, the main purpose of our research is to provide a competitive analysis, as a function of the processing set structure (i.e., as a function of s, k and m), for different variants of the $P2\left|\mathcal{M}_j, q=2\right|F$ problem.

3 Problems studied and Results

We study the following eight different variants of the $P\left|\mathcal{M}_j, q=2\right|F$ problem:

- *Variant 1:* $(P\left|\mathcal{M}_j, q=2, p_j=1\right|C_{\max})$: The online problem of minimizing the makespan with unit processing times.

- *Variant 2:* $(P\,|\mathcal{M}_j, q=2, p_j \in \{a_1, a_2\}|\, C_{\max})$: The online problem of minimizing the makespan with job-type dependent processing times, where a_i denote the processing time of any job of type i $(i = 1, 2)$.
- *Variant 3:* $(P\,|\mathcal{M}_j, q=2, p_j \in \{a_F, a_{NF}\}|\, C_{\max})$: The online problem of minimizing the makespan with machine-set dependent processing times; where a_F denote the processing time of any job assigned to the set of flexible machines $(\{M_{s+1}, ..., M_k\})$ and a_{NF} denote the processing time of any job assigned to the set of non-flexible machines $(\{M_1, ..., M_s\} \cup \{M_{k+1}, ..., M_m\})$.
- *Variant 4:* $(P\,|\mathcal{M}_j, q=2|\, C_{\max})$**:** The online problem of minimizing the makespan with arbitrary (but machine independent) processing times;
- *Variant 5:* $(P\,|\mathcal{M}_j, q=2, \text{known } n|\, C_{\max})$**:** The semi-online problem of minimizing the makespan with arbitrary processing times, where the number of jobs (n) is known in advance.
- *Variant 6:* $(P\,|\mathcal{M}_j, q=2, \text{known } \Sigma p_j|\, C_{\max})$**:** The semi-online problem of minimizing the makespan with arbitrary processing times, where the total processing time is known in advance.
- *Variant 7:* $(P2\,|\mathcal{M}_j, q=2, p_j=1|\, \Sigma C_j)$: The online problem of minimizing the sum of completion times with $m = 2$ machines and unit processing times.

Table 1 below summarizes the competitive results we obtain:

Variant	**Competitive ratio**	**Optimal**
Variant 1	$\frac{mk}{(m-s)k+s^2} \leq 4/3$	$\surd$
Variant 2	$\frac{mk}{(m-s)k+s^2} \leq 4/3$	$\surd$
Variant 3	$1+\frac{as(k-s)}{((k-s)+(m-k)a)(as+k-s)+s^2a^2} \leq 4/3^*$	$\surd$
Variant 4	$1+\frac{k(m-1)}{k(m-s)+s^2} < 7/3$	
Variant 5	1	$\surd$
Variant 6	$2-1/m$	
Variant 7	$1+\left(\frac{-\alpha+\sqrt{4\alpha^3-\alpha^2+2\alpha-1}}{2\alpha^2+1}\right)+O(\frac{1}{n})^{**}$	$\surd^{***}$
* $a = a_2/a_1 \geq 1$ ** $\alpha \approx 1.918$ ***The algorithm is asymptotically optimal.		

Table 1 Competitive ratio results for the seven variants we study.

References

1. Jiang, Y.W., He, Y., and Tang, C.M., Optimal Online Algorithms for Scheduling on Two Identical Machines Under a Grade of Service, *Journal of Zhejiang University Science A*, **7** (3), 309–314 (2006).
2. Jiang, Y., Online Scheduling on Parallel Machines with Two GoS Levels, *Journal of Combinatorial Optimization*, **16** (1), 28-38 (2008).
3. Park, J., Chang, S.Y., and Lee, K., Online and Semi-Online Scheduling of Two Machines under a Grade of Service Provision, *Operations Research Letters*, **34** (6), 692–696 (2006).
4. Sleator, D.D., and Tarjan, R.E., Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM*, **28**, 202-208 (1985).
5. Zhang, A., Jiang, Y., and Tan, Z., Online Parallel Machines Scheduling with Two Hierarchies, *Theoretical Computer Science*, **410**, 3597-3605 (2009).

MISTA 2015

The resource dependent assignment problem with a convex assignment cost function and its relation to scheduling with controllable processing times

Dvir Shabtay · Liron Yedidsion · Andrey Lisovoy

1 Introduction

Assignment problems deal with the question of how to assign a set of n *agents* to a set of n *tasks* such that each task is performed only once and each agent is assigned to a single task so as to minimize a specific predefined objective. An assignment is simply a *permutation* ϕ which maps each *element* i of $\{1,2,\ldots,n\}$ onto a unique element $\phi(i)$ of $\{1,2,\ldots,n\}$ and can be presented by a *permutation vector* $\phi=(\phi(1),\phi(2),...,\phi(m))$, where $\phi(i)=j$ means that agent j is assigned to task i in permutation ϕ. The most well-known assignment problem is the *linear sum assignment problem* ($LSAP$), where the cost of assigning agent j to a task i is given by a fixed parameter, c_{ij} for $i,j=1,...,n$ and the objective is to minimize $\sum_{i=1}^{n} c_{i\phi(i)}$. However, in many real-life applications of assignment problems, the assignment cost may be controllable by allocating resources to each agent for executing his task. Accordingly, Yedidsion *et al.* [8] present and analyze a new variant of an assignment problem, which they refer to as the resource dependent assignment problem ($RDAP$). In the $RDAP$ the cost of assigning agent j to task i is given by

$$c_{ij}=\xi_i p_j(u_j), \tag{1}$$

where ξ_i is task i's *assignment cost parameter*, u_j is the amount of resource allocated to agent j and $p_j(u_j)$ is the *assignment cost function* of agent j, which is a decreasing function of u_j.

A solution for the $RDAP$ is defined by a permutation ϕ and by a resource allocation vector $\mathbf{u}=(u_1,u_2,...,u_n)$ and the quality of a solution is measured by two different criteria. The first is the *total assignment cost* which is given by

$$c(A)=\sum_{i=1}^{n} c_{i,\phi(i)}(u_{\phi(i)})=\sum_{i=1}^{n}\xi_i p_{\phi(i)}(u_{\phi(i)}), \tag{2}$$

where $A=(\phi,\mathbf{u})$. The second is the *total resource consumption cost* which is given by

e-mails: dvirs@bgu.ac.il, lirony@ie.technion.ac.il and lisovoy1987@gmail.com

Address(es) of author(s) should be given

$$U(A) = \sum_{j=1}^{n} v_j u_j, \tag{3}$$

where v_j is the cost of assigning one unit of resource to agent j.

We focus on two different variations of the *RDAP*. In the first, *RDAP1*, we aim to find a solution A that minimizes $c(A) + U(A)$, while in the second, *RDAP2*, we aim to find a solution A that minimizes $c(A)$, subject to $U(A) \leq U_v$, where U_v is an upper bound on the total resource consumption cost.

2 Single machine scheduling problems with controllable processing times and their relation to the *RDAP*

Let us first present a formal definition of the set of single machine scheduling problems with controllable processing times. A set of n independent jobs, $J = \{1, 2, \ldots, n\}$, is to be processed on a single machine. The *processing time* of job j, p_j, is a function of the *amount of resource*, u_j, allocated to the processing operation. A solution is specified by a resource allocation vector $\mathbf{u} = (u_1, u_2, ..., u_n)$ and by a job permutation $\phi \in \Phi$ where Φ is the set of all $n!$ possible permutations of the n jobs. Similar to *RDAP*, the quality of a solution is measured by two criteria: The first, f, is a *scheduling criterion* and is dependent on the job completion times, and the second, U, is the *resource consumption criterion*. Among the possible f criterion are $f \in \left\{\sum_{j=1}^{n} C_j, \sum_{j=1}^{n} W_j, \sum_{j=1}^{n} E_j, \sum_{j=1}^{n} T_j, C_{\max}\right\}$, where C_j is the *completion time* of job j; $W_j = C_j - p_j$ is the *waiting time* of job j; d_j is the *due date* of job j; $L_j = C_j - d_j$ is the *lateness* of job j; $T_j = \max\{0, L_j\}$ is the *tardiness* of job j; $E_j = \max\{0, -L_j\}$ is the *earliness* of job j and $C_{\max} = \max_{j=1,\ldots,n}\{C_j\}$ is the maximal completion time (*makespan*). The U criterion is given by eq. (3), where here v_j is the cost of assigning one unit of resource to the operation of job j.

Similar to *RDAP*, we define two variations for each scheduling problem (denoted by *SP1* and *SP2*) that are identical to the two variations of *RDAP* (*RDAP1* and *RDAP2*), with the scheduling criterion f replacing the assignment cost $c(A)$. Yedidsion *et al.* [8] show that there is a large set of single machine scheduling problems in which their scheduling criterion, f, can be represented by the model in (1) with a specific set of ξ_j values associated with each one of them. Thus, the corresponding *SP1* and *SP2* problems on a single machine can be viewed as special cases of *RDAP*.

Consider, for example, a single machine scheduling problem with controllable processing time, where scheduling criterion is to minimize the total completion time, i.e., the case where $f = \sum_{j=1}^{n} C_j$. It is well-known that for the single machine problem $\sum_{j=1}^{n} C_j = \sum_{i=1}^{n} (n - i + 1) p_{\phi(i)}(u_{\phi(i)})$, where $j = \phi(i)$ implies that job j is assigned to position i in job processing sequence ϕ. It implies that the cost of assigning job j to position i is given by $c_{ij} = \xi_i p_j(u_j)$,where $\xi_i = (n - i + 1)$. Accordingly, the single machine scheduling problem with controllable processing time, where the scheduling criterion is to minimize the total completion time is a special case of the *RDAP* where $\xi_i = (n - i + 1)$ for $i = 1, ..., n$.

A subset of the set of scheduling problems that can be viewed as special cases of *RDAP* is presented in Table 1 below, where α, β, γ and δ are nonnegative parameters and $ETC = \alpha \sum_{j=1}^{n} E_j + \beta \sum_{j=1}^{n} T_j + \delta C_{\max}$.

The Scheduling Criterion	Positional Penalties (ξ_i)
$\sum_{j=1}^{n} C_j$	$n+i-1$
$\sum_{i=1}^{n}\sum_{j=i}^{n} \lvert C_i - C_j \rvert$	$(i-1)(n-i+1)$
$\sum_{i=1}^{n}\sum_{j=i}^{n} \lvert W_i - W_j \rvert$	$i(n-i)$
$ETC+\gamma\sum_{j=1}^{n} d_j$ (*)	$\alpha(i-1)+\gamma n+\delta$ for $i \le l^*$ $\beta(n-i+1)+\delta$ for $i > l^*$
$ET+\gamma\sum_{j=1}^{n} d_j$ (**)	$\alpha i+\gamma(n+1)+\delta$ for $i \le l^*-1$ $\beta(n-i)+\gamma+\delta$ for $i \ge l^*$
$ET+\gamma\sum_{j=1}^{n} d_j$ (***)	$\epsilon(n-i+1)+\delta$
$ET+\gamma_1 n\underline{d}+\gamma_2 nD$ (****)	$\alpha(i-1)+n\gamma_1+\delta$ for $i \le l_1^*$ $n\gamma_2+\delta$ for $l_1^* < i \le l_2^*$ $\beta(n-i+1)+\delta$ for $i > l_2^*$
(*) $d_j = d$ for $j=1,...,n$ and d is a decision variable. l^* can be computed in constant time.	
(**) $d_j = p_j + s$ for $j=1,...,n$ and s is a decision variable l^* can be computed in constant time.	
(***) Each job can be assigned a due date with no restrictions.	
(****) The scheduler can assign a common due window $\left[\underline{d}, \overline{d} = \underline{d} + D\right]$ where D is a constant, for the completion time of each job. l_1^* and l_2^* can be computed in constant time.	

Table 1 A subset of single machine scheduling problems which can be represented as special cases of the *RDAP*

3 Literature review and our objectives

Yedidsion *et al.* [8] study the *RDAP* with the following *linear* assignment cost function

$$p_j\left(u_j\right) = \overline{p}_j - b_j u_j, \quad 0 \le u_j \le \overline{u}_j < \overline{p}_j / b_j, \tag{4}$$

where $\overline{p}_j$ is the non-compressed assignment cost of agent j; $\overline{u}_j$ is the upper bound on the amount of resource that can be allocated to agent j; and b_j is the positive cost compression rate of agent j. They prove that *RDAP1* is solvable in $O(n^3)$ time, and that *RDAP2* is $\mathcal{NP}$-hard for *any* set of ξ_i parameters satisfying $\xi_i \neq \xi_j$ for any $i \neq j$. In a different paper, Shabtay *et al.* [6] provide a pseudo-polynomial time algorithms for solving *RDAP2* when the assignment cost function is given by (4).

Although commonly used in resource allocation problems, the linear function in (4) has the drawback that it fails to reflect the *law of diminishing marginal returns*. This law states that productivity increases at a decreasing rate with the amount of resource employed, i.e., that $dp_j(u_j)/du_j > 0$ and $d^2 p_j(u_j)/(du_j)^2 < 0$ for any $u_j > 0$. Thus, we analyze the *RDAP* by using the following *convex* decreasing assignment cost function which does reflect the *law of diminishing marginal returns*:

$$p_j\left(u_j\right) = B_j + \left(w_j/u_j\right)^k, \tag{5}$$

where w_j represents the *workload* of job j, B_j is a lower bound on the processing time of job j and k is a positive constant.

Eq. (5) is commonly used in resource allocation problems. For example, Flynn *et al.* [1] show that in computer microprocessors the processing time is reduced by approximately the cube root of the allocated power, implying that $k = 1/3$. Flynn's *et al.* cube-root rule for Complementary Metal–Oxide Semiconductor (*CMOS*) based devices was later adopted by many other researchers (see the survey paper by Irani and Pruhs [2]). Monma *et al.* [4] pointed out that the time required to perform many

actual government and industrial operations can be expressed by eq. (5) with $k = 1$, and that the time required to perform very large scale integration (VLSI) circuit design operations may also be presented by (5) with $k = 0.5$. Yao *et al.* [7] also use eq. (5) with $k = 0.5$ for modeling CPU time via energy consumption.

4 Analysis of *RDAP1* and *RDAP2* with a Convex assignment cost function

4.1 A polynomial time solution to *RDAP1*

Our objective here is to find a solution $A = (\phi, \mathbf{u})$ which minimizes

$$z(A) = c(A) + U(A) = \sum_{i=1}^{n} \xi_i (B_{\phi(i)} + \left(w_{\phi(i)}/u_{\phi(i)}\right)^k) + \sum_{j=1}^{n} v_{\phi(i)} u_{\phi(i)}. \quad (6)$$

Lemma 1 *When expressed as a function of the assignment decision, the optimal resource allocation,* $\mathbf{u}^*(\phi) = (u^*_{\phi(1)}, u^*_{\phi(2)}, ..., u^*_{\phi(n)})$*, for* RDAP1 *is*

$$u^*_{\phi(i)} = \left(\frac{\xi_i k}{v_{\phi(i)}}\right)^{\frac{1}{k+1}} \left(w_{\phi(i)}\right)^{\frac{k}{k+1}} \text{ for } i = 1, ..., n. \quad (7)$$

Proof By taking the derivative of eq. (6) with respect to $u_{\phi(i)}$, equating it to zero and solving it for $u_{\phi(i)}$, we obtain eq. (7). Since the objective is a convex function, these are necessary and sufficient conditions for an optimal resource allocation.

By substituting eq. (7) into the objective in eq. (6), we obtain that:

$$z\left(\phi, \mathbf{u}^*(\phi)\right) = \sum_{i=1}^{n} \xi_i B_{\phi(i)} + (k^{\frac{-k}{k+1}} + k^{\frac{1}{k+1}}) \sum_{i=1}^{n} (\xi_i)^{\frac{1}{k+1}} \eta_{\phi(i)},$$

where

$$\eta_{\phi(i)} = \left(w_{\phi(i)} v_{\phi(i)}\right)^{\frac{k}{k+1}} \text{ for } i = 1, ..., n. \quad (8)$$

Therefore, if we define the value c_{ij} by

$$c_{ij} = \xi_i B_j + (k^{\frac{-k}{k+1}} + k^{\frac{1}{k+1}}) (\xi_i)^{\frac{1}{k+1}} \eta_j \quad (9)$$

it represents the minimal possible cost resulting from assigning agent j to task i. Since each agent is assigned to a single task and each task is performed only once, *RDAP1* reduces to the *LSAP* and the following optimization algorithm can be applied to solve *RDAP1*:

An optimization algorithm for *RDAP1*

Step 1. Calculate the c_{ij} values by using eq. (9) with eq. (8).

Step 2. Solve the resulting *LSAP* to determine the optimal assignment, ϕ^*.

Step 3. Allocate the resources according to eq. (7) with $\phi = \phi^*$.

Theorem 1 *Algorithm 4.1 solves* RDAP1 *in* $O(n^3)$ *time.*

Proof The correctness of the algorithm follows from the analysis that appears in this section. Step 1 requires $O(n^2)$ time. Step 2 requires the solution of *LSAP* which takes $O(n^3)$ time (see, e.g., [5]) and Step 3 can be performed in linear time. Thus, the overall computational complexity of the algorithm is indeed $O(n^3)$.

4.2 The Analysis of *RDAP2*

Lee and Lei [3] study the *RDAP2* problem in a context of a scheduling problem with $\xi_i = n - i + 1$. They conclude that the general problem is probably $\mathcal{NP}$-hard although they fail to prove it. We, however, where able to prove that the following much more general theorem holds.

Theorem 2 RDAP2 *is* $\mathcal{NP}$*-hard for* any *given set of task cost parameters satisfying* $\xi_l \neq \xi_m$ for any $l \neq m$, *even for the special case where* $v_j = 1$ *for* $j = 1, ..., n$.

The following lemma is later used to reduce *RDAP2* to a purely combinatorial problem (the proof is omitted for the sake of brevity).

Lemma 2 *When expressed as a function of the assignment decision, the optimal resource allocation,* $\mathbf{u}^*(\phi) = (u^*_{\phi(1)}, u^*_{\phi(2)}, ..., u^*_{\phi(n)})$, *for* RDAP2 *is*

$$u^*_{\phi(i)} = \frac{(\xi_i)^{\frac{1}{k+1}} \left(w_{\phi(i)}\right)^{\frac{k}{k+1}}}{\left(v_{\phi(i)}\right)^{\frac{1}{k+1}} \sum_{j=1}^{n} (\xi_j)^{\frac{1}{k+1}} \eta_{\phi(j)}} U_v \text{ for } i = 1, ..., n. \tag{10}$$

By substituting eq. (10) into eq. (2) (with $p_{\phi(i)}\left(u_{\phi(i)}\right)$ given by (5)), we obtain that the minimal assignment cost can be expressed as follows:

$$c\left(\phi, \mathbf{u}^*(\phi)\right) = \sum_{i=1}^{n} \xi_i B_{\phi(i)} + U_v^{-k} \left(\sum_{i=1}^{n} (\xi_i)^{\frac{1}{k+1}} \eta_{\phi(i)}\right)^{k+1} = c_1(\phi) + U_v^{-k} \left(c_2(\phi)\right)^{k+1}, \tag{11}$$

where $c_1(\phi) = \sum_{i=1}^{n} \xi_i B_{\phi(i)}$ and $c_2(\phi) = \sum_{i=1}^{n} (\xi_i)^{\frac{1}{k+1}} \eta_{\phi(i)}$. Thus, *RDAP2* reduces to a purely combinatorial (and non-linear) problem of finding ϕ which minimizes (11).

We develop several algorithms to solve the reduced combinatorial problem. Among them is an exact branch and bound algorithm, and several heuristic algorithms. Among the heuristic algorithms is a unique approximation algorithm, which is based on solving a weakly polynomial number of *RDAP1's*. Furthermore, by the implementation of a set of experimental studies we show that our unique approximation algorithm can easily solve very large instances of *RDAP2* with an average gap of approximately zero between the value of the heuristic solution and the value of a tight lower bound.

References

1. Flynn, M. J., Hung, P., and Rudd, K. W., 1999, Deep-Submicron Microprocessor Design Issues, *IEEE Micro*, **19**(4), 11-22.
2. Irani, S., and Pruhs, K., 2005, Algorithmic Problems in Power Management, *SIGACT News*, **36**(2), 63-76.
3. Lee, C. Y., and Lei, L. 2001, Multiple-Project Scheduling with Controllable Project Duration and Hard Resource Constraint: Some Solvable Cases, *Annals of Operation Research*, **102**, 287-307.
4. Monma, C. L., Schrijver, A., Todd, M. J., and Wei, V. K., 1990, Convex Resource Allocation Problems on Directed Acyclic Graphs: Duality, Complexity, Special Cases and Extensions, *Mathematics of Operations Research*, **15**, 736-748.

5. Papadimitriou, C. H., and Stieglitz, K., 1982, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.
6. Shabtay, D., Steiner, G., and Yedidsion, L., A Pseudo-Polynomial Time Algorithm for Solving the Resource Dependent Assignment Problem, *Discrete Applied Mathematics*, , **182**, 115-121 (2015).
7. Yao, F., Demers, A., and Shenker, S., 1995, A Scheduling Model for Reduced CPU Energy, *IEEE Syposium on Foundations of Computer Science*, Los Alamitos, CA, 374–382.
8. Yedidsion, L., Shabtay, D., and Kaspi, M., 2011, Complexity Analysis of an Assignment Problem with Controllable Assignment, *Discrete Applied Mathematics*, **159**, 1264-1278 (2011).

MISTA 2015

The two-machine flowshop total completion time problem: A branch-and-bound based on Network-flow formulation

Boris Detienne · Ruslan Sadykov · Shunji Tanaka

1 Introduction

We consider the problem of scheduling a set $J = \{1, \ldots, n\}$ of jobs in a two-machine flowshop with the objective to minimize the sum of the completion times of jobs. The jobs are available at time zero and they should be processed first on machine 1, and then on machine 2. Each machine can process at most one job at a time. Let p_j^m denote the processing time of job j on machine m, where $m = 1, 2$. All the processing times are integer. Preemption of the processing of the jobs in not allowed on either machine. Let C_j^m denote the completion time of job j on machine m. According to the scheduling classification, the problem is denoted by $F2||\sum C_j$. It is known to be NP-hard in the strong sense [6]. It has been shown by Conway et al. [3] that there exists at least one optimal solution where both machines have the same sequence of the jobs. Thus, we may restrict the search to permutation schedules only.

The problem $F2||\sum C_j$ has been studied in the literature for many years. Akkan and Karabati [1] suggest a network flow formulation for the problem. They use a Lagrangian relaxation to obtain a lower bound which is used inside a branch-and-bound algorithm. This algorithm is able to solve instances with up to 60 jobs with small processing times (up to 10) and up to 45 jobs with large processing times (up to 100). In this work, we propose an improved branch-and-bound algorithm for the problem $F2||\sum C_j$ based on their work. To obtain stronger dual bounds, we use a network which is larger than the one used in [1]. Different dominance rules and filtering techniques are exploited in order to cope with the size of the network. The structure of the network allows us to compute the dual bound only once in the root, and then recompute the bound in linear time at every node of the enumeration tree. Thus, tens

Boris Detienne
INRIA team RealOpt and Mathematics Institute, Bordeaux University, Talence, France
E-mail: boris.detienne@math.u-bordeaux1.fr

Ruslan Sadykov
INRIA team RealOpt and Mathematics Institute, Bordeaux University, Talence, France
E-mail: ruslan.sadykov@inria.fr

Shunji Tanaka
Institute for Liberal Arts and Sciences, Kyoto University, Japan
E-mail: tanaka@kuee.kyoto-u.ac.jp

of millions of nodes can be checked in a reasonable time. Using the proposed algorithm, we were able to solve all instances of the problem $F2||\sum C_j$ with up to 100 jobs with large processing times.

2 Network formulation

In the following, $[k]$ denotes the index of the job in position k. The completion times $C^m_{[k]}$ of the job in position k, $k \in J$, on machines $m = 1, 2$ can be computed as $C^1_{[k]} = C^1_{[k-1]} + p^1_{[k]}$ and $C^2_{[k]} = \max\{C^1_{[k]}, C^2_{[k-1]}\} + p^2_{[k]}$

In [1], the authors introduce the notion of time lag between the processing of a same job on both machines to write an assignment model and a network flow model for the problem. This kind of models is also called *waiting time-based* [models] in [7]. The completion-to-completion lag L^c_k of the job in position k, $k \in J$ is defined as the time elapsed between the completion of the job on machine 1 and its completion on machine 2 : $L^c_k = C^2_{[k]} - C^1_{[k]} = \max\{0, L^c_{k-1} - p^1_{[k]}\} + p^2_{[k]}$. In order to design a convenient network model, the objective function can be expressed as:

$$\sum_k C^2_{[k]} = \sum_k (C^1_{[k]} + L^c_k) = \sum_k \left((n-k+1)p^1_{[k]} + L^c_k\right) \quad (1)$$

Our model is based on a transshipment type network $G(V, A)$, which extends the one proposed in [1]:

- Each node $v_{k,l,i} \in V$ of the network is associated with one position k in the sequence, and the start of job i when the completion-to-completion lag of the previous job is l. Two dummy nodes $(0, 0, \emptyset)$ and $(n+1, \emptyset, \emptyset)$ are added, representing the start and the end of the schedule, respectively.
- Each arc $(v_{k,l_1,i_1}, v_{k+1,l_2,i_2}) \in A$ from node v_{k,l_1,i_1} to node v_{k+1,l_2,i_2} is associated with the processing of job i_1 in position k, when the completion-to-completion lag of the previous job is equal to l_1, so that job i_1 ends with a completion-to-completion lag equal to l_2 and is immediately followed by job i_2. According to the expression of the objective function given by (1), the cost of using the arc is $c(v_{k,l_1,i_1}, v_{k+1,l_2,i_2}, j) = (n-k+1)p^1_{i_1} + l_2$.

The scheduling problem can be seen as the problem of finding a minimum cost flow of value 1 (a path) from the source node to the sink node, going through exactly one arc associated with each job.

Network reduction By dualizing these job occurence constraints, we obtain a Lagrangian lower bound. Given a vector of Lagrange multipliers, this bound can be computed by solving a simple shortest path problem in G. Using the same idea, a lower bound of the length of a feasible path that passes through a node (resp. an arc) is computed and the node (resp. the arc) is removed from the network if it is greater than an upper bound of the path length. This lower bound can be computed by applying dynamic programming in both forward and backward manners when the shortest path problem is solved [8]. More reductions are obtained by reinforcing the job assignment constraint in the shortest path problem for one job at a time [5]. Moreover, several dominance rules from [10, 4, 2] are used to remove some arcs in the graph.

3 Branch-and-bound algorithm

The set of possible job sequences is explored, by enumerating the set of feasible (with respect to the job occurence constraint) paths in graph G. We proceed from the left to the right in the graph. We perform a Depth-First-Search. In a preprocessing stage, an upper bound is computed using a dynasearch procedure [9], and graph G is reduced using a subgradient procedure inside which the network reduction procedures are applied. For each job j and node v of G, we compute the cost of a shortest path from v to the sink node going through exactly one arc representing j, as well as the cost of one going through no arc representing j. In order to evaluate a partial sequence σ represented by a path ending at node v, we compute a lower bound of the cost of extending σ into a feasible schedule. For each job j, we derive in constant time a lower bound in which the job assignment constraint is enforced for j. A job is a candidate for the next job in the sequence only if there is a corresponding arc in G and the resulting subsequence is not dominated according to several dominance rules, coming from the literature or extending some of them. Candidate jobs are processed in non-decreasing order of the distance from the corresponding terminal node to the sink.

4 Numerical results

The branch-and-bound algorithm solves to optimality all instances of our test bed, composed of randomly generated instances with up to 100 jobs with up to 100-unit long processing times (as in [1]). The hardest instance is solved in 7759 seconds, while all the others are solved in less than one hour on a laptop equipped with a 2.7GHz processor and 4GB RAM. The average computing time for 100-job instances is 502.6 seconds, and the average size of the search tree is 128.8 millions of nodes.

References

1. C. Akkan and S. Karabati. The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm. *European Journal of Operational Research*, 159(2):420–429, December 2004.
2. B.W. Cadambi and Y.S. Sathe. Two-machine flowshop scheduling to minimize mean flow time. *Opsearch*, 30(1):35–41, 1993.
3. R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, MA, 1967.
4. F. Della Croce, V. Narayan, and R. Tadei. The two-machine total completion time flow shop problem. *European Journal of Operational Research*, 90(2):227 – 237, 1996.
5. B. Detienne, S. Dauzère-Pérès, and C. Yugma. An exact approach for scheduling jobs with regular step cost functions on a single machine. *Computers & Operations Research*, 39(5):1033–1043, 2012.
6. M. R. Garey, D. S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.
7. A. Gharbi, T. Ladhari, M. K. Msakni, and M. Serairi. The two-machine flowshop scheduling problem with sequence-independent setup times: New lower bounding strategies. *European Journal of Operational Research*, 231(1):69–78, November 2013.
8. T. Ibaraki and Y. Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, July 1994.
9. S. Tanaka. An extension of the dynasearch to the two-machine permutation flowshop scheduling problem. In *Proceedings of the 2010 International Symposium on Flexible Automation*, page 6, 2010.
10. S.L. van de Velde. Minimizing the sum of the job completion times in the two-machine flow shop by lagrangian relaxation. *Annals of Operations Research*, pages 257–268, 1990.

MISTA 2015

Robust multi-mode resource constrained project scheduling of building deconstruction under uncertainty

Rebekka Volk • Felix Hübner • Frank Schultmann

1 Introduction

Buildings are characterized by their immobility, heterogeneity and uniqueness. Due to their long lifecycles, buildings undergo several decades and are refurbished, retrofitted, remediated or modernized by generations of users, residents and proprietaries to adapt the building to changing users' and environmental requirements. During their lifecycles, different building elements and products are installed, removed or changed due to building modification. Often, these modifications of the building structure, equipment and fittings as well as the deterioration and contamination of buildings are not documented. In addition, some buildings cannot economically be adapted to changing requirements. The buildings in question undergo deconstruction (and replacement) processes, often in spatially limited sites of dense urban areas and with limited resources available. The objective of the responsible stakeholders of the deconstruction is either makespan minimization or cost minimization or both depending on the type of building, the urgency or the preference of the responsible parties.
In building deconstruction, different scarce renewable resources (machines, staff) can be applied to perform so called jobs like separation, deconstruction, crushing, sorting and loading activities that might be performed several times due to reworks e.g. in the case of contaminations. Furthermore, technical or organizational precedence relations of activities have to be respected. Job shop scheduling on m machines where each job has its own predetermined route [1] with precedence constraints, makespan minimization and under resource-constraints ($Jm \mid prec \mid C_{max}$) seems the most appropriate scheduling type for this application case [2]. But as deconstruction belongs to the category of site fabrication, there are rather different modes jobs can be performed in than predetermined routes on a machine environment. Thus, here we consider a multi-mode project scheduling problem (MPS | prec | C_{max}) under resource constraints (MRCPSP) and with zero-lag finish-start precedence relations (notation according to [3,4]). Single-mode and multi-mode project scheduling problems are a generalization of job shop scheduling problems [2,3,5,6]. This problem class is NP complete [2].
The consideration of uncertainty is crucial in deconstruction projects to reduce disruptions and vulnerability of the project. Uncertainties in project scheduling might arise from work content, resource availabilities, precedence constraints in the project network etc. [7]. In deconstruction projects, main sources of uncertainty are processing times of activities and the existence of activities which depend on the building configuration (set of building elements and their specific properties such as different secondary raw materials or hazardous materials) and onsite resource and location availability and capacity (such as staff, hydraulic excavators, site equipment, containers etc.). Since uncertainty is mostly caused by building inherent elements, in a first step our approach restricts to the creation of a robust schedule which is the proactive consideration of potential disturbances resulting from building elements to avoid multiple changes in schedules. Other uncertainties onsite (resource availability) might be considered in a second step regarding

Rebekka Volk, Karlsruhe Institute of Technology (KIT), Institute of Industrial Production (IIP), E-mail: rebekka.volk@kit.edu
Felix Hübner (KIT, IIP), E-mail: felix.huebner@kit.edu
Frank Schultmann (KIT, IIP/University of Adelaide), E-mail: frank.schultmann@kit.edu

project execution e.g. via reactive scheduling and repairing methods in multi-period scheduling that are not considered in this contribution.
First, our contribution will give a short literature overview about existing scheduling models under the consideration of uncertainty. Subsequently, we will give a brief summary of our scheduling model for building deconstruction under uncertainty and first results. Our contribution is concluded by a short summary and outlook.

2 Literature review

Literature about project scheduling under uncertainty is extensive and several publications are dedicated to predictive-reactive, proactive (robust), fuzzy and stochastic scheduling [8]. Research focuses on the development of proactive schedules and reactive (reparation) strategies in the case of schedule infeasibility during project execution and some work has already been done in the field of proactive and reactive project scheduling for the single-mode RCPSP [9]. Main approaches focus on the consideration of duration variability and related buffer insertion procedures [10,11] or resource unavailability [12]. However, literature on proactive-reactive scheduling in multi-mode RCPSP (MRCPSP) is still quite rare [9]. Fuzzy scheduling is based on expert estimations of activity durations whereas stochastic scheduling is based on known distributions of activity durations. Scenario-based robust scheduling approaches are based on discrete robust optimization (e.g. [13]) and are considered in recent literature for RCPSP where uncertainty is modeled via scenarios using discrete or discretized probabilities [14,15].
In deconstruction projects, activity-based and location-based scheduling can be differentiated. Activity-based scheduling models consider activities explicitly, location-based scheduling describe activities only implicitly by their occupation of locations during activity completion. Activity-based problems with limited renewable (constrained over time periods and available after activity is terminated) and non-renewable (budgeted over the whole project) resources are formulated as RCPSP where scheduling and capacity planning is performed simultaneously [1,3,8,16]. Location-based approaches are often applied in construction projects [17,18] and thus seem promising for deconstruction application, too. Here, we consider a joint activity-based approach where locations are considered as resources that are required for activity performance.
Although robust RCPSP approaches and their problem variants [7,8,19,14,20] are numerous, applied works in deconstruction are rare [2,21–24]. Scheduling applications in deconstruction projects are mainly limited to deterministic approaches [2,22,23,25,26] yet although uncertainties are indispensable when it comes to deconstruction scheduling of decades old and often undocumented buildings at the end of their life cycle. Schultmann (2003) [22] formulates a fuzzy scheduling approach, that is divided into six crisp RCPS problems with optimistic, more or less expected and pessimistic values with different fuzzy set membership values (1, ε, λ). However, this approach does not cover all uncertainties decision makers are confronted with such as fuzzy due dates, fuzzy capacity constraints, uncertain composition of the components or fuzzy precedence relations [22]. As in building deconstruction, several potential scenarios of building configurations can be anticipated, that strongly influence activity durations and scheduling, a scenario-based approach seems promising.

3 Approach

In our approach, a MRCPSP is formulated and solved (B) for several potential scenarios of building configurations (A) and recommendations for decision making in deconstruction projects are given (C) (see Figure 1). Thus, our approach combines a scenario simulation, with robust optimization (scheduling). Due to the buildings' uniqueness the assignment of probabilities of occurrences, e.g. of activity durations or building element existence, is difficult. However, the creation of possible scenarios (building configurations) and related activity durations is possible. Thus, our approach is based on a scenario construction and expert estimates on optimistic, expected and pessimistic activity durations that can be represented by fuzzy sets. A stochastic

approach is theoretically possible, but assumptions on parameters of the related beta distribution can only be estimated.

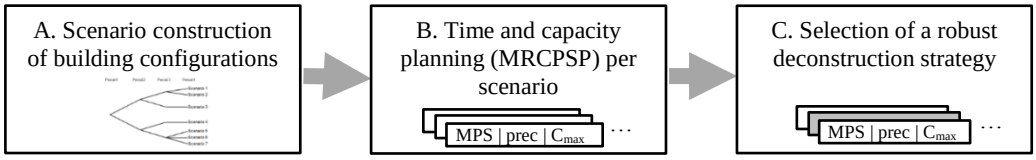


Figure 1: Model overview

3.1 Scenario construction of building configurations

In building deconstruction, several epistemic uncertainties are prevalent (aleatoric uncertainties are not considered in this work since they are hardly quantifiable). Relevant project-related scenario attributes such as the characteristics of building elements, the material of the elements and varying resource capacities contain epistemic uncertainties. Activity-related scenario attributes such as the activity durations also contain epistemic uncertainties which can be e.g. represented by a minimum, expected and maximum duration per activity and per building element unit.
Different scenarios are created via an initial building configuration of an examined building that is varied with other possible discrete project-related and activity-related scenario attributes as described above. In the presented case, each building element is assigned to a single 'deconstruction activity' and consequently, 'building element existence' and 'element material' are not only scenario attributes but also activity-related attributes. Thus, in this case a scenario consists of different occurring parameter values of the mentioned activity-related attributes based on the information just before project start. E.g. a building with reinforced concrete slabs versus timber slabs leads to different activities (existence) and resource demands (durations, modes) to be scheduled. If sample pre-testing revealed specific materials, varying material combinations for the respective building element are excluded from scenario construction.
Complete enumeration theoretically leads to more than $4*10^{12}$ scenarios with 10 different building element types that are permuted with 22 potential element materials (on average 4.7) per building element. We tested the approach so far with 10 to 100 scenarios.
For each scenario, project activities are derived from the respective enumerated building configuration with their belonging activity durations, precedence constraints and renewable resource demands (e.g. machine, staff). Depending on the parameter value 'element material' per activity, the expected activity duration is calculated via duration per element unit and per material and the related mode selection is adapted according to technical feasibility. For simplicity, project-related uncertainties such as uncertain resource availabilities are neglected in this model, yet.

3.2 Time and capacity planning (MRCPSP) per scenario

In deconstruction projects, activity j is planned with expected durations $E(d_{jm_j})$ of the respective scenario on limited resources q_j and is subject to acyclic precedence constraints. Deconstruction activities are performed with different resources such as hydraulic excavators, handheld pneumatic drills, chisels, crane and varying number of skilled staff and associated cost, so-called modes m. Thus, we formulate a classical MRCPSP with activity zero-lag finish-start precedences with the objective of minimizing project makespan. For each scenario, the optimal activity modes and the optimal schedule (start times t and resource usage of each activity) are determined in pre-calculated time windows between earliest finish (EF) and latest finish (LF). For reasons of simplicity and due to the application case, the model restricts to modeling of renewable (r) resources q_{jmr} that can be used in different modes m and equally qualified staff is assumed. A further technical constraint is the restricted selection of activity modes that depend on the prevalent building 'element material' that has to be deconstructed. Furthermore, on limited job-sites the definition of location-based activities is helpful to schedule working teams and

their resources in different parts the site to avoid obstructing accumulations of resources or material. In a consequence and due to safety reasons, location $l \in L$ is formulated as a specific renewable resource where every activity is at least occupying a location at a time ($q_{jm} \geq 1, \forall\, l \in L$) and in every location only one activity can take place simultaneously ($Q_{lt} = 1\ \forall\, l \in L$):

$$\min\ C_{\max} = \sum_{m=1}^{M_j} \sum_{t=EF_J}^{LF_J} t * x_{Jmt}$$

Subject to:

$$\sum_{m=1}^{M_j} \sum_{t=EF_j}^{LF_j} x_{jmt} = 1, \qquad for\ j = 1, \dots, J$$
Time and mode selection constraint

$$\sum_{m=1}^{M_i} \sum_{t=EF_i}^{LF_i} t * x_{im_it} \leq \sum_{m=1}^{M_j} \sum_{t=EF_j}^{LF_j} \left(t - E(d_{jm_j})\right) * x_{jm_jt}, \qquad for\ j = 2, \dots, J; i \in P_j;\ m_i \in M_i;\ m_j \in M_j$$
Precedence constraint relations

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} q_{jmr} \sum_{\tau=t}^{t+d_{jm}-1} x_{jm\tau} \leq Q_{rt} \qquad for\ r \in \{R, L\}, t = 1, \dots, T, q_{jm} \geq 1, Q_{lt} = 1\ \forall\, l \in L$$
Renewable resource constraint Q_r

$x_{jmt} \in \{0,1\}$ — $j = 1, \dots, J;\ m_j = 1, \dots M_j$, Boolean decision variable deciding on activity j starting at time t in mode m.

The model is implemented as a binary, linear integer problem (BILP) in MATLAB R2015b. The commercial CPLEX solver from IBM ILOG Optimization Studio 12.5.1 is used to solve the problem.

Usual problem sizes in deconstruction start at about 100 activities on 10-15 modes and 30-40 renewable resources. As this is a computably challenging problem and is considerably increased in large deconstruction projects by location resources, we performed first tests with problem instances with 17 real activities, 9 modes, 11 resources and 4 locations that showed promising results. On average, this includes 4.2 potential modes per activity, 2.7 different resources per mode, a resource factor RF=0.24 and a resource strength RS=0 (without locations)[1]. The latter indicates resource scarcity, so that some activities have to be scheduled consecutively [4].

3.3 Selection of a robust, proactive deconstruction strategy

Quality robustness aims at the minimization of the deviation from the best-case scenario objective value (here: makespan), while solution robustness covers the minimization of schedule deviation between scenarios [8,27] to increase preparedness for the worst-case. In deconstruction projects, the focus mostly lies on the compliance with time constraints regarding the project deadline [2] due to tight time schedules of owners and general contractors that will reuse the parcel of land or remaining building parts after the deconstruction is completed. However, changes in schedule are associated with additional setup time and cost to organize necessary resources. Thus, on the one hand quality robustness with a reasonably good objective value under any likely scenario [14] seems appropriate to plan deconstruction projects. On the other hand, a solution-robust (stable) schedule is preferable from a time-based and also from an organizational point of view. Our approach aims at proactively finding a total solution-robust schedule x where all absolute regrets (earliness and tardiness) $AR_k(x) = 0$ for all scenarios $k = 1, \dots, K$ and at the same time finding a solution that comprises the 'most' quality-robust schedule.

In this step, the generated optimum schedules are transformed into deconstruction strategies (sequence of activities and resources/locations used). Then, the deconstruction strategies are

[1] $RF = \frac{1}{n|R|} \sum_{j=1}^{J} \sum_{r \in R} \delta(q_{jr}), \forall r \in \{R\}$, with $\delta(q_{jr}) = \begin{cases} 1, for\ q_{jr} > 0 \\ 0, for\ q_{jr} = 0 \end{cases}$ and $RS = \frac{Q_{rt} - Q_r^{min}}{Q_r^{max} - Q_r^{min}} = 0,\ \forall r \in \{R\}$, where Q_r^{min} and Q_r^{max} are lower and upper bounds of resource capacities. See [4] for further information on definitions of control parameters RF and RS.

applied on all scenarios and the respective project makespan is calculated. The deconstruction strategies of all scenarios are aggregated, assessed and compared with each other via several robustness criteria. Applied robustness criteria are the mean, the variance and the standard deviation of project makespan, as well as Laplace, maxi-min, maxi-max, Hurwicz and Savage-Niehans (regret) criteria. Other possible criteria can be found e.g. in [7]. Results include rankings of alternative deconstruction strategies with respect to deconstruction strategy frequency and mean objective value, mean and standard deviation of objective value and mean and variance of objective value. According to the decision makers' risk preferences, recommendations for the adequate project strategy are given.
The presented approach is based on previous works of MRCP scheduling under uncertainty. The difference to known approaches is the strong relation to the presented application case in deconstruction projects, as well as the extension by a scenario construction to get more suitable activity durations, the consideration of locations in MRCPSP and the combination of the two robustness criteria solution robustness and quality robustness. A proactive approach is more practical for the given application case, as the MRCPSP consists of many activities, modes and resources whose status has to be updated manually when the schedule becomes infeasible. A reactive or dynamic scheduling might become interesting, if auditing and controlling of project status is automated or supported via optical sensors.

4 Conclusion

In the field of building deconstruction the consideration of uncertainties is crucial for project planning, scheduling and management. However, our literature review shows that in this application case most approaches insufficiently apply project scheduling methods under resource constraints and uncertainty. The model results show that the consideration of uncertainties in different building configurations and activity durations via scenarios has an impact on project scheduling, resource management and decision making in deconstruction projects. Furthermore, the consideration of robustness criteria and decision makers' risk preferences leads to other preferred strategies and schedules.
Further work might be concentrated on the extension of the approach with respect to reactive or dynamic scheduling (e.g. schedule repair or rolling horizon). Also, the examination of non-renewable resource constraints such as project budget might be included and the assumption that each building element is assigned to several activities might be included in an extended case study. Scheduling of multi-projects, multi-skills and dynamic scheduling aspects with unexpected events might be considered, too.

Acknowledgements

We would like to thank the anonymous reviewers for their recommendations that noticeably improved the comprehensiveness and clarity of our paper. This work is supported by the Federal Ministry of Education and Research (BMBF) of Germany and the project ResourceApp as part of funding "r³ - innovative technologies for resource efficiency". Furthermore, we thank Dr. Julian Stengel for his brilliant work and his numerous helpful suggestions and inputs.

References

1. Pinedo, M. *Scheduling - Theory, Algorithms and Systems*. (Springer, 2011).
2. Schultmann, F. *Kreislaufführung von Baustoffen - Stoffflußbasiertes Projektmanagement für die operative Demontage- und Recyclingplanung von Gebäuden*. (Erich Schmidt Verlag (Reihe Baurecht und Bautechnik), 1998).

3. Brucker, P., Drexl, A., Möhring, R., Neumann, K. & Pesch, E. Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research* **112,** 3–41 (1999).
4. Neumann, K., Schwindt, C. & Zimmermann, J. *Project Scheduling with Time Windows and Scarce Resources - Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions*. (Springer, 2002).
5. Brucker, P. *Scheduling Algorithms*. (Springer Verlag, 2004).
6. Kolisch, R. *Project Scheduling under Resource Constraints - Efficient Heuristics for Several Problem Classes*. (Springer, 1995).
7. Hazir, Ö., Haouari, M. & Erel, E. Robust scheduling and robustness measures for the discrete time/cost trade-off problem. *European Journal of Operational Research* **207,** 633–643 (2010).
8. Herroelen, W. & Leus, R. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research* **165,** 289–306 (2005).
9. Deblaere, F., Demeulemeester, E. & Herroelen, W. *Exact and heuristic reactive planning procedures for multimode resource-constrained projects*. 45 (Research Center for Operations Management Department of Decision Sciences and Information Management Faculty of Business and Economics Katholieke Universiteit Leuven (Belgium), 2008). at <http://www.econ.kuleuven.be/fetew/pdf_publicaties/KBI_0818.pdf>
10. Van de Vonder, S., Demeulemeester, E. & Herroelen, W. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research* **189,** 723–733 (2008).
11. Van de Vonder, S., Demeulemeester, E., Herroelen, W. & Leus, R. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics* 227–240 (2005).
12. Lambrechts, O., Demeulemeester, E. & Herroelen, W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *J. Sched.* **11,** 121–136 (2008).
13. Aissi, H., Bazgan, C. & Vanderpooten, D. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research* **197,** 427–438 (2009).
14. Artigues, C., Leus, R. & Talla Nobibon, F. Robust optimization for resource-constrained project scheduling with uncertain activity durations. *Flexible Services and Manufacturing Journal* **25,** 175–205 (2013).
15. Mulvey, J., Vanderbei, R. & Zenios, S. Robust optimization of large scale systems. *Oper. Res.* **43,** 264–281 (1995).
16. Hartmann, S. & Briskorn, D. A survey of variants and extensions of the resource constrained project scheduling problem. *European Journal of Operational Research* **207,** 1–14 (2010).
17. Seppänen, O., Ballard, G. & Pesonen, S. The Combination of Last Planner System and Location-Based Management System. *Lean Construction Journal* **2010,** 43–54 (2010).
18. Kenley, R. & Seppänen, O. *Location-Based Management for Construction - Planning, Scheduling and Control*. (Spon Press, 2010).
19. Chtourou, H. & Haouari, M. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & Industrial Engineering* **55,** 183–194 (2008).
20. Demeulemeester, E. & Herroelen, W. *Robust Project Scheduling*. **3,** (now Publishers Inc., 2009).
21. Schultmann, F. & Rentz, O. Environment-oriented project scheduling for the dismantling of buildings. *OR Spektrum* **23,** 51–78 (2001).
22. Schultmann, F. Dealing with uncertainties in (de-)construction management - the contribution of fuzzy scheduling. in *CIB Report 287 'Deconstruction & Materials Reuse' - Proceedings of the 11th Rinker International Conference on Deconstruction and Materials Reuse (ed. A. Chini)* 73–82 (2003).
23. Sunke, N. Planning of Construction Projects: A Managerial Approach. (Siegen, 2009).

24. Schultmann, F. & Rentz, O. Fuzzy Scheduling for the Dismantling of Complex Products. in *Operations Research Proceedings 2002, Selected Papers of the International Conference on Operations Research (SOR 2002), Klagenfurt* (eds. Leopold-Wildburger, U., Rendl, F. & Wäscher, G.) 302–307 (Springer Verlag, 2003).
25. Schultmann, F., Sindt, V., Ruch, M. & Rentz, O. Schadstofforientierte Erfassung und Demontage von Gebäuden. *Abfallwirtschaftsjournal* **3,** 38–42 (1997).
26. Spengler, T. in *Industrielles Stoffstrommanagement - Betriebwirtschaftliche Planung und Steuerung von Stoff- und Energieströmen in Produktionsunternehmen, zugl. Karlsruhe, Universität, Habilitationsschrift, Dissertation 1998* **54,** (Erich Schmidt Verlag (ESV), 1998).
27. Scholl, A. *Robuste Planung und Optimierung - Grundlagen, Konzepte und Methoden, Experimentelle Untersuchungen*. (Physica-Verlag, 2001).

MISTA 2015

Approaches to modeling job-shop problems with blocking constraints

Julia Lange

1 Introduction

Considering blocking constraints in job-shop problems is motivated by the idea of tackling train scheduling problems. In a train scheduling problem a set of trains is regarded in a given network with different routes, entering and desired leaving times. All trains have to be scheduled, so that the sum of their tardiness according to the leaving times is minimal. This is done by deciding in which order and with which starting times the trains pass the track sections. For more detailed problem descriptions see for example [9] and [5].
A specific characteristic of train scheduling problems is the appearance of blocking constraints. These constraints refer to situations in which a train occupies a track section longer than necessary until the succeeding section is free to travel. Further conditions regarding deadlocks and the train length (see e.g. [5]), headways (see e.g. [3]), acceleration and deceleration (see e.g. [1]) and others can be included in the problem to match real-world situations but will not be regarded here.
The given problem structures can be transferred to a blocking job-shop scheduling problem. The track sections are taken as machines M_k, whereby the trains refer to jobs J_i with different technological orders and processing times on the machines. Every job has a given release and (desired) due date and the objective function is to minimize the sum of tardiness of all jobs. In the majority of the literature dealing with job-shop scheduling problems the optimization criterion is the minimization of the makespan (see e.g. [7] and [4]). But during the last two decades the observation of more complex tardiness-based sum criteria gained interest especially with regard to customer needs in train scheduling and other transportation problems.
The given train scheduling problem is shown to be NP-complete in [2] even without blocking constraints. Therefore a variety of solution approaches like meta-heuristics (see e.g. [8] and [10]), graph-based modeling and solution methods and problem-fitted branch and bound procedures (see e.g. [3]) are presented in the literature to obtain good, feasible solutions.
In the following the performance of different types of modeling approaches and IP

Julia Lange
Institute of Mathematical Optimization, Otto-von-Guericke-Universität Magdeburg
E-mail: julia.lange@ovgu.de

formulations for the blocking job-shop scheduling problem is compared. Two problem representations are modeled by the use of different assignment variables to evaluate the effect of routing flexibility and the choice of variables on the optimal objective function value and computation times.

2 Railway networks and their representation

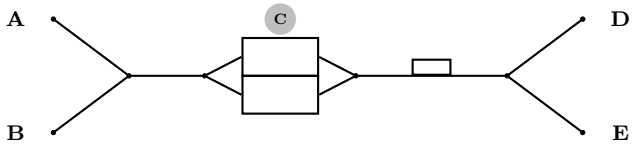


Fig. 1 Single track network

The networks regarded mainly consist of bidirectional single tracks. The only possibilities for trains to overtake in these single-track networks are stations with different numbers of parallel tracks and sidings. A simple example is given in Figure 1. The network consists of 4 bounding nodes A, B, D and E, at which trains enter and leave the network at given release and desired due times. C is a station with 2 parallel tracks and the single track between station C and the branch D,E has a parallel siding.
A single track section is transferred to a machine M_k. The representation of stations with parallel tracks and sidings basically depends on including or excluding the platforming decision. Two possibilities to represent parallel tracks are given in Figure 2.
Applying the *Parallel-Machine Approach* (PMA) each track refers to one machine M_k and the route of the train (the machines a job is to be processed on) has to be given in advance. This means that the platforming decision for a train in a station or siding is not done within the optimization but made in a previous planning phase. On the contrary, the *Machine-Unit Approach* (MUA) transfers parallel tracks to machine units $l = 1, \ldots, m_k$ of one machine M_k. With this the number of machines in the problem is reduced and the platforming decision is included in the optimization program. Since the exact assignment of trains to tracks or jobs to machines is not given in advance, a job-shop problem with flexible routes is considered. The integrated flexibility is expected to improve the optimal solution against the PMA by working with an optimal job-to-machine-unit-assignment and to extend the size of the model and with this its solving time.

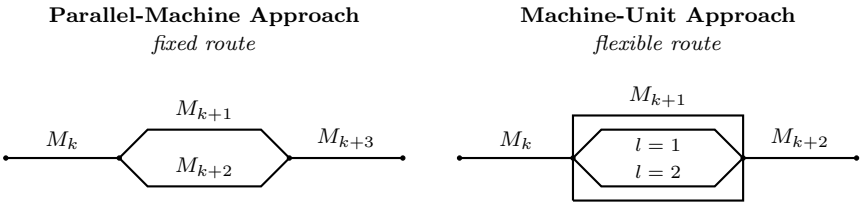


Fig. 2 Representation of parallel tracks

3 Model approaches and blocking restrictions

In the following two different types of decision variables are used to model the job-shop problems described above. Both are defined based on Operations O_{ij}, which refer to the j-th operation of job J_i or the j-th passing of a track section of a train. This is done to enable trains to pass a certain track section more than once on their routes. Such a situation appears each time a train visits a terminus (terminal station) as an intermediate station on its route. The determination of the order in which jobs are processed on a machine or trains pass a track section can be modeled by *precedence variables*

$$y_{iji'j'k} := \begin{cases} 1 & \text{if operation } O_{ij} \text{ is scheduled before } O_{i'j'} \\ & \text{on machine } M_k \\ 0 & \text{else.} \end{cases}$$

applied first by Manne in 1960 (see [6]) and *order variables*

$$x^r_{ijk} := \begin{cases} 1 & \text{if operation } O_{ij} \text{ is scheduled as the } r\text{-th operation} \\ & \text{on machine } M_k \\ 0 & \text{else.} \end{cases}$$

used first by Wagner in 1959 (see [11]).

Both model approaches are implemented considering *blocking constraints*. These restrictions account for trains blocking a track section, after passing it, until the next track section on their route is available. This refers to manufacturing of parts which exceed warehouse capacities in weight or height and therefore cannot be stored in intermediate buffers while processing. Mathematically, the starting time of an operation $O_{i'j'}$ is defined to be greater or equal to the starting time of the successor of the operation O_{ij} if $y_{iji'j'k} = 1$ on machine M_k holds.

4 Test instances and computational results

To generate test instances based on the railway network given in Figure 1 eight characteristic trains are defined. Their properties and routes are chosen to represent typical train movements. The travel times are determined so that passenger trains with medium speed, express trains with high speed and freight trains with low speed are depicted. Two of these trains traverse station C like a terminus and form the necessity of an operation-based modeling. The platforming decision for the PMA is done arbitrarily in advance.

To do the first computational experiments five instances with 10 trains of different types are generated. The decisions regarding the occurrence of train types and the release times are made randomly, whereby it was assured that each characteristic (Passenger/Express/Freight train and terminus/traversing) appears at least once in each instance. The computational results reached by PMA and MUA as well as by applying precedence variables (PrecVar) and order variables (OrderVar) are summarized in Table 1.

The instances were solved using ZIMPL with SCIP 3.1.0 and CPLEX 12.6.1. The computation time in these first calculations was limited to three hours. For each of the instances the best result obtained is given and results denoted with asterisk state optimal objective function values.

Overall it is clear to see that the appliance of precedence variables outperforms the use of order variables. This might be due models formed with precedence variables

Instance	Parallel-Machine Approach				Machine-Unit Approach			
	PrecVar		*OrderVar*		*PrecVar*		*OrderVar*	
	$\sum T_i$	Time	$\sum T_i$	Time	$\sum T_i$	Time	$\sum T_i$	Time
10_1	138*	5.8 s	-	180 min	80*	36.45 min	621	180 min
10_2	90*	1.39 s	-	180 min	82*	18.8 min	735	180 min
10_3	72*	0.97 s	804	180 min	61*	2.56 min	953	180 min
10_4	41*	0.62 s	466	180 min	27*	7.97 s	-	180 min
10_5	71*	1.33 s	940	180 min	42*	24.07 s	940	180 min

Table 1 Computational results

including more variables but less constraints than those constituted with order variables. Furthermore, the results show that including flexibility with the platforming decision effects total tardiness as well as computation time significantly. A simultaneous determination of an optimal job-to-machine-assignment and optimal starting times decreases minimal total tardiness.
In future research the trade off between flexibility and computation time will be studied as well as improving the solution procedure by the help of heuristic methods and dominance relations. Additionally, a closer look will be taken on theoretical reasons for the results observed above.

References

1. Burdett, R. L. and Kozan, E., A disjunctive graph model and framework for constructing new train schedules, European Journal of Operational Research, 200, 85–98 (2010).
2. Cai, X. and Goh, C. J., A fast heuristic for the train scheduling problem, Computers and Operations Research, 21, 499–510 (1994).
3. D'Ariano, A., Pacciarelli, D. and Pranzo, M., A branch and bound algorithm for scheduling trains in a railway network, European Journal of Operational Research, 183, 643–657 (2007).
4. Fattahi, P., Mehrabad, M. S. and Jolai, F., Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, Journal of Intelligent Manufacturing, 18, 331–342 (2007).
5. Liu, S. Q. and Kozan, E., Scheduling trains as a blocking parallel-machine job shop scheduling problem, Computers and Operations Research, 36, 2840–2852 (2009).
6. Manne, A., On the job-shop scheduling problem, Operations Research, 8, 219–223 (1960).
7. Mascis, A. and Pacciarelli, D., Job-shop scheduling with blocking and no-wait constraints, European Journal of Operational Research, 143, 498–517 (2002).
8. Mattfeld, D. C. and Bierwirth, C., An efficient genetic algorithm for job shop scheduling with tardiness objectives, European Journal of Operational Research, 155, 616–630 (2004).
9. Oliveira, E. and Smith, B. M., A job-shop scheduling model for the single-track railway scheduling problem, Research Report Series - University of Leeds - School of Computer Studies, 21 (2000).
10. Pranzo, M. and Pacciarelli, D., An iterated greedy metaheuristic for the blocking job shop scheduling problem, Research Report Universita degli Studi Roma Tre, RT-DIA-208-2013 (2013).
11. Wagner, H. M., An integer linear-programming model for machine scheduling, Naval Research Logistics Quarterly, 6, 131–140 (1959).

MISTA 2015

A dynamic model of task processing for scheduling problems without additional resources

Mateusz Gorczyca · Adam Janiak · Maciej Lichtenstein

1 Introduction

There is a very large amount of scheduling papers concerning problems where no additional resources are needed (besides processors) to complete the task. To the best of our knowledge, all mathematical models used to describe processing without additional resources are static ones. In other words, the rate of task processing is not given explicitly in the mathematical description of task. Usually, the task processing time is simply defined as a function of a single variable instead.

Take for example the static model of task introduced by Biskup [2]. In this model, time $p_i(r)$ needed to process the i-th task depends only on r, which is simply a position of the task in the sequence. Precisely, it is a product of the constant factor p_i (defined as normal processing time) and r^a, where $a > 0$ is a learning factor. Task processing time $p_i(r) = p_i r^a$ decreases with task position in the sequence.

Another example is the model of task processing described by Alidaee and Ahmadian [1], where the processing speed is given by function $v : \{1, \ldots, n\} \to [0; 1]$. This speed depends on the number of already processed tasks and does not change during processing of the task. The processing time of the i-th task $p_i(r) = p_i/v(i)$.

Since the task execution is usually a dynamic process, the static model of task is just an approximation. This approximation is supposed to have a convenient form (as in the above examples) and acceptable accuracy, which usually contradict each other. Thus, there arise a question if it is possible to include the dynamics of processing in the task model and still have a tractable scheduling problem. Such a dynamic model of task can be general enough to include many static models as its special case. If so, then the methodology obtained for scheduling problems with such a model can be useful for a broad class of problems.

In the next section we propose a dynamic model of processing for which the instantaneous rate of processing depends on two factors: the total contribution of the tasks already completed by the processor and the influence of the currently processed task. Further, we show that models of Biskup as well as model of Alidaee and Ahmadian are exemplary special cases of the presented model.

Mateusz Gorczyca · Adam Janiak · Maciej Lichtenstein
Systems Research Institute, Polish Academy of Sciences, Poland
E-mail: {mateusz.gorczyca,adam.janiak,maciej.lichtenstein}@pwr.wroc.pl

2 The model of task processing

The set $\mathcal{T} = \{1, 2, \ldots, n\}$ of non-preemptive tasks is to be processed on a single variable-speed processor. The processor can process at most one task at a time. The processing of task i is described as changing its *(processing) state* x_i from 0 to the final state denoted by $\hat{x}_i > 0$. It is assumed that the each processed task i being in state x_i has a measurable instant *influence* $e_i(x_i)$ on the processing rate. Moreover, each completed task adds a *contribution* $\hat{e}_i := e_i(\hat{x}_i)$ to the processing rate. For a given task i processed after completion of tasks from a given subset $T \subset \mathcal{T}$, the task processing rate is defined by the following dynamic model:

$$\frac{dx_i(t)}{dt} := f\left(\sum_{k \in T \cup \{0\}} \hat{e}_k + e_i(x_i(t))\right), \tag{1}$$

where:

- function $e_i : [0, \hat{x}_i] \to [0, e_i^{max}]$ represents the influence (of task i in state $x_i(t)$ on the processing rate),
- value $\hat{e}_0 \geq 0$ represents the initial contribution (which can be the contribution of the tasks processed before tasks from the set $\mathcal{T}$, service tasks, etc.),
- function $f : [\hat{e}_0, \hat{e}_0 + \sum_{i=1}^{n} e_i^{max}] \to [0, \infty)$ characterizes the processing rate and takes into account both contribution of the completed tasks and influence of the processed task.

It is assumed that $e_i(\cdot)$, $i = 1, \ldots, n$, and $f(\cdot)$ are piecewise continuous and non-negative. The exemplary influence function and rate function is presented, respectively, in upper-left and upper-right plot in Figure 1.

Denote the total contribution of the already processed tasks (except initial contribution) by $e := \sum_{k \in T} \hat{e}_k$. After some simple transformations and integration we obtain the following formula for the task processing time:

$$p_i(e) = \int_0^{\hat{x}_i} \frac{dx_i}{f\left(\hat{e}_0 + e + e_i(x_i)\right)}. \tag{2}$$

The processing time in dependance of e for influence and rate functions from Figure 1 is presented in the lower-left plot in the same figure. As it follows from the above formula, the performance time is equal to the integral of $1/f(e(x))$ over interval $[0, \hat{x}]$. If e is constant, $1/f(e)$ can be interpreted as a time needed to change the task state by one unit. The value of $1/f(e)$ for the considered example is given in the lower-right plot in Figure 1.

2.1 Special cases

The presented model of task processing is a special case of many models considered in the literature. It directly reflects model of Alidaee and Ahmadian [1] after the following substitution:

$$\hat{x}_i = p_i, \quad i = 1, \ldots, n, \tag{3}$$

$$\hat{e}_0 = 0, \tag{4}$$

$$\hat{e}_i(x_i) = 1, \quad 0 \leq x_i \leq \hat{x}_i, \quad i = 1, \ldots, n, \tag{5}$$

$$f(e) = v(i), \quad e = 1, \ldots, n, \tag{6}$$

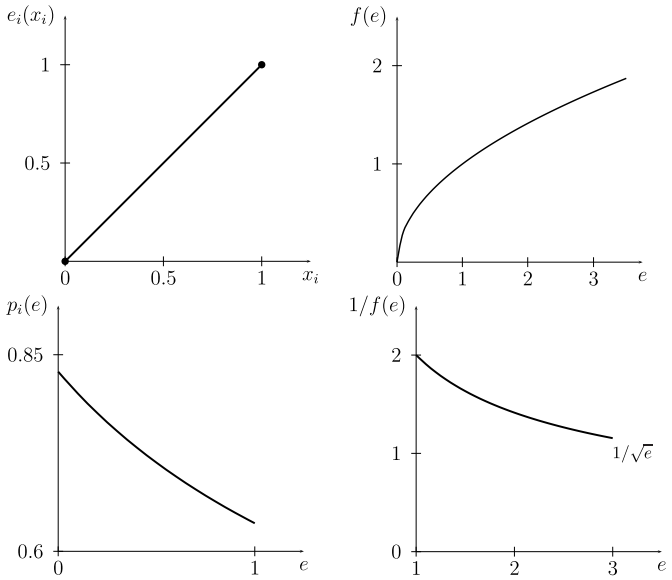


Fig. 1 The exemplary influence function (upper-left), processing rate function $f(e)$ (upper-right), processing time (lower-left) and function $1/f(e)$ (lower-right).

and the model of Biskup [2] after:

$$\hat{x}_i = p_i, \quad i = 1, \ldots, n, \tag{7}$$

$$\hat{e}_0 = 1, \tag{8}$$

$$e_i(x_i) = \begin{cases} 0 & \text{for} \quad 0 \leq x_i < \hat{x}_n \\ 1 & \text{for} \quad x_i = \hat{x}_n \end{cases}, \quad i = 1, \ldots, n, \tag{9}$$

$$f(e) = e^{-a}. \tag{10}$$

3 Conclusion

The model presented above can be very useful in describing task processing where the dynamics is essential. As it was shown above with the special cases, the presented model is also very flexible. The methodology for solving scheduling problems with the proposed model and makespan criterion will be presented at the conference.

Acknowledgements This research has been supported by Polish National Science Center, decision No. DEC-2011/03/B/ST6/00364.

References

1. Bahram Alidaee and Ahmad Ahmadian, Scheduling on a single processor with variable speed, Information Processing Letters, 60, 189-193 (1996)
2. Dirk Biskup, Single-machine scheduling with learning considerations, European Journal of Operational Research, 115, 173-178 (1999)

MISTA 2015

Vector space decomposition for linear programming

Jacques Desrosiers · Jean Bertrand Gauthier · Marco E. Lübbecke

1 Introduction

Dantzig proposed the *Primal Simplex algorithm* in the summer of 1947. The race for shaving seconds off the resolution process started soon after. While researchers keep breaking records, all of the methods steer the algorithm away from its original simplicity. It turns out that all of the methods return on investment. Amongst the most promising ones, we find those of which that account for the phenomenon of *degeneracy* one way or another. The Primal Simplex algorithm may cycle and this is only possible under degeneracy. An important survey on anti-cycling schemes and pivot-selection rules can be found in [17] where a very large number of these are examined.

All known primal algorithms base their stopping criteria on the optimality conditions provided by the dual variables and the reduced cost values. Since no two algorithms share the same iterative process, there must be different levels of exposure that can be harvested from dual feasibility. This presentation focusses on the results of a recent paper [11] where we propose pivot-selection rules guided by dual feasibility considerations.

2 Literature review

Such concerns about degeneracy appeared within the *Column Generation algorithm*, the counterpart of the Primal Simplex algorithm for linear programs with a very large number of variables. Over the last three decades, column generation has been widely used to solve complex problems for routing and scheduling applications, see [13]: In

Jacques Desrosiers
GERAD & HEC Montréal; 3000, Côte-Sainte-Catherine ; Montréal, Canada, H3T 2A7
E-mail: jacques.desrosiers@hec.ca

Jean Bertrand Gauthier
GERAD & HEC Montréal; 3000, Côte-Sainte-Catherine ; Montréal, Canada, H3T 2A7
E-mail: jean-bertrand.gauthier@hec.ca

Marco E. Lübbecke
RWTH Aachen University, Kackertstraße 7, D-52072 Aachen, Germany
E-mail: marco.luebbecke@rwth-aachen.de

urban transportation systems (for buses and drivers, handicapped and elderly people in adapted transportation); in airline, rail and maritime industries (for various fleet assignment; aircraft, locomotive and ship routing and scheduling; daily, weekly and monthly crew schedules); for traffic assignment in satellite communication systems; for real-time dispatching of automobile service units; and many other fields.

Degeneracy in column generation has been dealt with using *Dual Variable Stabilization*, see for example [5], and [1] for a stabilized column generation framework and the many references therein. Another remarkable approach is the *Improved Primal Simplex algorithm* (IPS) [7,15,16]. This algorithm (1) dynamically updates a working basis of reduced size (indeed, the cardinality of the subset of non degenerate variables), (2) fixes the dual variables of an associated row-reduced problem, and (3) selects a convex combination of variables entering the basis in a pricing problem derived from a linear transformation of the original problem based on the current working basis. IPS not only prevents degenerate pivots at every iteration but also maintains the basis status of the current solution.

Set partitioning models are well suited for many practical combinatorial optimization problems such as vehicle routing and crew scheduling applications [4,3]. Unfortunately, they largely suffer from degeneracy. Historically speaking, the *Dynamic Constraint Aggregation* (DCA) of [8,6] was specifically designed to overcome this situation in the context of solving the linear relaxation of the set partitioning problem by column generation. Although DCA is a precursor of IPS, its methodology does not follow the same decomposition scheme. The slight difference is that the linear transformation used to derive both the row-reduced and the pricing problems does not necessarily rely on a subset of the current basic columns. Indeed, a set of linearly independent columns, leading or not to a feasible solution, is sufficient. A survey on recent tools for primal degenerate linear programs can be found in [10] whereas [2] presents an implementation of a *stabilized DCA* algorithm for set partitioning problems.

3 Contribution

Consider the linear program (LP) with lower and upper bounded variables:

$$\begin{aligned} z^\star := \min \quad & \mathbf{c}^\intercal \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \quad [\boldsymbol{\pi}] \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned} \tag{1}$$

where $\mathbf{x}, \mathbf{c}, \mathbf{l}, \mathbf{u} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, and $m < n$. We assume matrix $\mathbf{A}$ is of full row rank and LP is feasible and bounded. The vector of dual variables $\boldsymbol{\pi} \in \mathbb{R}^m$ associated with the equality constraints appears within brackets on the right-hand side.

In the presentation, we describe a vector space decomposition framework which, given a feasible basic solution, (1) fixes the values of a subset of dual variables, and (2) optimizes the remaining ones for finding the *smallest value* of the reduced cost. It turns out that the dual formulation of this pricing problem selects a convex combination of variables to move from one solution to the next. The way to divide these two subsets of dual variables (either fixed or optimized) relies on the choice of a vector subspace basis and opens a wide spectrum of algorithmic possibilities.

The vector space decomposition framework has many well known algorithms as special cases. In the following, we list some of these algorithms together with the most important results.

- The most known special case of the proposed framework is the Dantzig's rule for the Primal Simplex algorithm. It considers all dual variables fixed and selects a single entering variable of minimum reduced cost. This myopic strategy commonly results in degenerate pivots and unfortunately may not converge.

- At the other extreme when none of the dual variables are fixed, we stumble upon the *Minimum Mean Cycle-Canceling algorithm* (MMCC) originally devised for network flow problems. This algorithm is strongly polynomial [12, 14], and [9] further improves on the complexity results.

- Given a basic solution to LP, selecting as the vector subspace basis all the columns associated with the nondegenerate variables leads to the IPS algorithm.

- The proposed framework closes the theoretical gap between IPS and DCA, showing that the latter is also a special case of a unified primal framework.

- Based on the values taken by the variables at every iteration, the unified primal framework partitions the set of constraints in two parts; one set in the row-reduced problem, the other set in the pricing problem. This strategy can be seen as a dynamic application of the Dantzig-Wolfe decomposition principle, this time driven by the current value of the variables, not by the structure of the formulation.

- Properties of this framework allows identifying a number of subspace bases for which the derived a *family of algorithms* that totally avoids degenerate pivots. One such algorithm is IPS that finds movements along the edges of the polyhedron defined by the constraints, from one vertex to another, while improving the value of the objective function at every iteration.

- We also show that *all other members of this family* (amongst which belongs MMCC) not only identify improving edge directions but also non-edge ones, that is, *directions that are interior to the polyhedron*. Moreover, these interior directions are nonnegative combinations of the edge directions identified by IPS.

This methodological presentation proposes a better understanding of the primal algorithms for linear programming and, more importantly, of the phenomenon of degeneracy which can always be eliminated in various ways. It unifies within the same framework a variety of specialized algorithms for linear and network programs. Hopefully, this added comprehension will lead to the design of much better exact and heuristic algorithms for large scale applications.

References

1. Hatem M. T. Ben Amor, Jacques Desrosiers, and Antonio Frangioni. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics*, 157(6):1167–1184, 2009.

2. Pascal Benchimol, Guy Desaulniers, and Jacques Desrosiers. Stabilized dynamic constraint aggregation for solving set partitioning problems. *European Journal of Operational Research*, 223(2):360–371, 2012.
3. Guy Desaulniers, Jacques Desrosiers, Irina Ioachim, Marius M. Solomon, François Soumis, and Daniel Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In TeodorGabriel Crainic and Gilbert Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Springer, New York, NY, USA, 1998.
4. Jacques Desrosiers, Yvan Dumas, Marius M. Solomon, and François Soumis. Time constrained routing and scheduling. In Michael Ball, Tom Magnanti, Clyde Monma, and George Nemhauser, editors, *Handbooks in Operations Research and Management Science, Vol. 8: Network Routing*, volume 8, chapter 2, pages 35–139. Elsevier, Maryland Heights, MO, USA, October 1995.
5. Olivier du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.
6. Issmail Elhallaoui, Guy Desaulniers, Abdelmoutalib Metrane, and François Soumis. Bi-dynamic constraint aggregation and subproblem reduction. *Computers & Operations Research*, 35(5):1713–1724, 2008.
7. Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and François Soumis. An Improved Primal Simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, 23:569–577, 2011.
8. Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.
9. Jean Bertrand Gauthier, Jacques Desrosiers, and Marco E. Lübbecke. About the minimum mean cycle-canceling algorithm. *Discrete Applied Mathematics*, 2014.
10. Jean Bertrand Gauthier, Jacques Desrosiers, and Marco E. Lübbecke. Tools for primal degenerate linear programs: IPS, DCA, and PE. *EURO Journal on Transportation and Logistics*, 2015.
11. Jean Bertrand Gauthier, Jacques Desrosiers, and Marco E. Lübbecke. Vector space decomposition for linear programs. Les Cahiers du GERAD G-2015-16, HEC Montréal, Montreal, QC, Canada, March 2015.
12. Andrew V. Goldberg and Robert Endre Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.
13. Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
14. Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, 1994.
15. Vincent Raymond, François Soumis, and Abdelmoutalib Metrane. Improved primal simplex version 3: Cold start, generalization for bounded variable problems and a new implementation. Les Cahiers du GERAD G-2009-15, HEC Montréal, Montreal, QC, Canada, 2009.
16. Vincent Raymond, François Soumis, and Dominique Orban. A new version of the Improved Primal Simplex for degenerate linear programs. *Computers & Operations Research*, 37(1):91–98, 2010.
17. Tamás Terlaky and Shuzhong Zhang. Pivot rules for linear programming: A survey on recent theoretical developments. *Annals of Operations Research - Annals OR*, 46-47(1):203–233, 1993.

MISTA 2015

A GRASP based approach to dynamic scheduling of parallel heat treatment furnaces in a manufacturing company

Adil Baykasoglu • Fehmi Burcin Ozsoydan

1 Introduction

Scheduling is an active research area due to its central role for manufacturing systems. There are copious reported studies focusing variants of scheduling problems in the related literature [1]. As one of them, parallel machine scheduling problem (PMSP) has numerous applications in industry. The present study addresses an industrial application of identical PMSP, where jobs have planned release times and sequence-dependent set-up times between jobs are considered along with machine eligibility constraints. Moreover, dynamic events like job arrivals or cancellations make the handled problem more complex. In the firm, where the project is being carried out, respectable amount of machine components (mainly bolts and nuts) are manufactured particularly for automotive industry. In one of the stages of this manufacturing system, five similar parallel heat treatment furnaces are employed in order to satisfy the required durability of the machine components. According to the numerous examinations and analysis performed in the plant, it was shown that, this stage of the system is a bottleneck for the firm and findings demonstrated that, even a minor improvement in the scheduling process of those furnaces, might yield to a considerable amount of economical savings for the firm.

In the mentioned problem, each job (order from customers) has three distinctive attributes namely, raw material type, diameter of the component and its quality class, which determine the required temperature to be applied and the processing time. Based on these distinctive attributes, required temperature to be treated and required processing time (duration of heat treatment) are estimated by heat treatment engineers in a fuzzy manner. The reason to avoid using deterministic set-up times can be explained as follows.

For the problem of interest, consecutive jobs might require different levels of temperature. However, heating or cooling of the furnaces differs considerably. Heating up the furnaces from 400^0C to 600^0C is not the same task as cooling from 600^0C to 400^0C. Thus, the required setup time from job *i* to job *j* differs from the setup time between jobs *j* and *i*. Moreover, like current temperature of the medium, there exist some other uncontrollable

Adil Baykasoglu
Dokuz Eylul University, Izmir, Turkey
adil.baykasoglu@deu.edu.tr

Fehmi Burcin Ozsoydan
Dokuz Eylul University, Izmir Turkey
burcin.ozsoydan @deu.edu.tr

aspects which are affecting the required setup times. For this reason, a fuzzy rule based system (via ANFIS-Adaptive neuro-fuzzy inference system) is developed in order to estimate the sequence depended setup times.

Another fact, affecting the scheduling process is the release times of jobs, which are actually the corresponding completion times of the previous manufacturing process, cold forming. By using this information, which is kept in enterprise resource planning (ERP) system of the firm, more efficient scheduling scheme is aimed. However, there are also many disturbances in the system like dynamic events, including changing due dates, changing priorities, arrivals or cancellation of jobs, etc. The proposed dynamic scheduling system is also able to handle such dynamic events. Several performance indicators are also considered in developing effective schedules dynamically.

In the present work, a Greedy Randomized Adaptive Search Procedure (GRASP) [2] based optimization approach is proposed for the stated problem. We tried to utilize the "constructive solution generation scheme" and the "multi-start" ability of GRASP along with several improvements in order to develop an easy to implement yet effective dynamic scheduling algorithm.

2 Solution approach

The main reason for using a constructive approach is related to easiness in developing an effective algorithm to handle dynamic events (unplanned order arrivals, changes in parameters, cancellations etc.). Let's consider a standard evolutionary algorithm like genetic algorithms. As one can expect, the length of a chromosome is dependent on number jobs and number of machines. However, as a result of dynamic events like unplanned job arrivals or job cancellations, a repair or revision is required in the solution representation (encoding) strategy, which requires additional effort. On the other hand, if a solution is constructed from scratch at each generation (where job list, restricted candidate list, etc. are updated according to dynamic events) rather than using standard chromosome representation, handling dynamism becomes easier, which is actually an intrinsic feature of GRASP.

The algorithm of interest, GRASP has two main phases namely, construction and local improvement phases. In construction phase, a greedy rule is used for sorting jobs with respect to their greedy values. The next job to be scheduled is randomly selected from among restricted candidate list (RCL), which is a subset of all sorted jobs whose greedy value exceeds a threshold value defined by GRASP parameters [2]. In order to evaluate greedy values, the greedy function presented in [3], is adopted here. The function is as follows:

$$I_j = \frac{1}{p_j} exp\left[-\frac{max(d_j - p_j - t, 0)}{k_1 \bar{p}_m}\right] exp\left[-\frac{s^*_{i,j}}{k_2 \bar{s}_m}\right] \quad (1)$$

where

$$s^*_{i,j} = \begin{cases} s_{i,j} & if \quad r_j \le t, \\ r_j - t + s_{i,j} & otherwise \end{cases} \quad (2)$$

and I_j is the priority index, d_j is the due date, r_j is the release time, p_j is the process time of job *j*, respectively, *t d*enotes the current time, $s_{i,j}$ is the setup time between jobs *i* and *j*, $\bar{p}_m$ and $\bar{s}_m$ denote, respectively, the average processing time and the average setup time of the unprocessed jobs on machine *m*, and k_1 and k_2 are look-ahead parameters given by rules. Note that the first exponent gives higher priority to jobs, which are closer to their due date, and the second exponent prioritizes jobs, which have a small setup time relative to the average setup time and are ready for processing.

Subsequent to evaluating priority values of jobs, normalized priority values g_j of each job is obtained as given in (3) [3]. Next, RCL is generated using formulation presented in (4), where α is fixed to 0,3 [3]. Finally a random job is selected from RCL and scheduled to the first released available machine.

$$g_j = \frac{I_j}{I_{max}} \quad (3)$$

$$g_j \in [g_{min} + (1-\alpha)(g_{max} - g_{min}), g_{max}] \quad (4)$$

In the second phase, where the constructed solution is attempted to be improved, inter-machine/intra-machine swaps and insertion local moves [3] are performed. If any improvement is achieved, improved solution is accepted. These two phases are repeated until a termination criterion like achieving maximum number of generations is met. As we already mentioned, the lists and parameters utilized in GRASP algorithm are updated in ERP system of the company, if a dynamic event occurs. The developed optimization procedure is coded in MATLAB and JAVA. Integration of the developed algorithm to the ERP system of the firm is still under progress. From the preliminary tests of the proposed algorithm, it is observed that the proposed approach is capable of handling dynamic events and generating acceptable schedules. An illustration of the proposed scheduling procedure is depicted in Figure 1. As it can be seen from this figure, all of the required data is recorded and execution of algorithm is performed through ERP system of the firm.

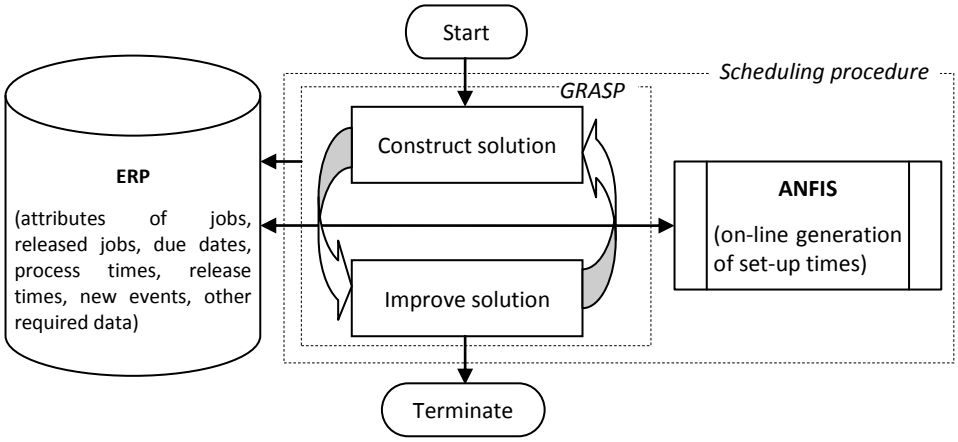


Figure 1. An illustration of the proposed approach.

3 Conclusion

The current work focuses on a real-life scheduling problem encountered in heat treatment process of a manufacturing company, where parallel furnaces need to be scheduled dynamically. Sequence dependent setup times, dynamic job arrivals/cancellations, changes in parameters and possible break down or maintenance of machines are some of the distinctive features of the current scheduling problem. For handling the mentioned dynamic parallel machine scheduling problem, a GRASP based scheduling system is developed. The preliminary findings and results obtained from the proposed systems are promising. The efficiency of the proposed approach is currently being tested by using real data.

Acknowledgements The current work is supported by "Republic of Turkey, Ministry of Science, Industry and Technology" under project number: SANTEZ 0293.STZ.2013-2.

References

1. Pinedo, M., Scheduling: theory, algorithms, and systems, page numbers. Prentice Hall, New Jersey (2002).
2. Feo, T., Resende, M. G. C., Greedy randomized adaptive search procedures, Journal of Global Optimization, 2, 1–27, (1995).
3. Armentano, V. A., Filho, M. F. F., Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach, European Journal of Operational Research, 183, 100–114, (2007).

MISTA 2015

The Discrete Parallel Machine Makespan Scheduling-Location Problem

Corinna Heßler · Kaouthar Deghdak

1 Introduction

Scheduling-Location (ScheLoc) Problems combine the two well-studied fields of location planning and scheduling theory. The goal of ScheLoc Problems is to simultaneously locate a set of machines and schedule a set of jobs on the machines such that some scheduling objective is optimized. Each job is stored in a job location and for processing on a machine the job first has to be moved to the corresponding machine location. The arrival time of the job at the machine location is the time the job becomes available for processing, i.e., it is the release date of the job. Therefore, this release date is dependent on the location of the job and the machine. In ScheLoc Problems pure scheduling objectives are considered, like minimizing the makespan. However, its optimal value is still dependent on the location decisions because the availability of jobs depends on the machine locations.
Although location planning is more of a strategic decision, while machine scheduling is situated in the operational level of the supply chain, there are still many applications where solving the two problems simultaneously is required to overcome the suboptimal results of a sequential approach. One example of an application can be found in the mining industry where crushers are used in processing the minerals. Those crushers are movable and can change their location after processing a given set of valuable minerals. An overview of applications can be found in [4].
The ScheLoc Problem was introduced in [2,3]. In these studies the Single Machine Network (SMN) ScheLoc Problem is considered, i.e., the problem where the location of a single machine in a network is to be found, and some polynomial time algorithms for special cases are reported. Single Machine Planar (SMP) ScheLoc Problems were treated by [1,4,5] and polynomial time algorithms for special cases developed. More recently a polynomial time algorithm for a Network ScheLoc Problem with preemption

Corinna Heßler
Department of Mathematics, Optimization Research Group, Technical University of Kaiserslautern
E-mail: c.hessler@mathamatik.uni-kl.de

Kaouthar Deghdak
Université François-Rabelais de Tours, CNRS, LI EA 6300, OC ERL CNRS 6305, Tours, France
E-mail: deghdak@univ-tours.fr

was proposed in [6].
The Parallel Machine ScheLoc Problem was defined in [2] (for the network case) and [4] (for the general case), however, there are no specialized algorithms for these problems. In this study we consider the Discrete Parallel Machine Makespan (DPMM) ScheLoc Problem, i.e., the problem of selecting a fixed number p of machine locations from a finite set of possible locations, in each of them one of p parallel machines is located, and schedule a given set of jobs on the machines respecting location-dependent release dates such that the makespan is minimized. We propose several clustering heuristics and a local search to solve this problem.

2 Problem Definition

Let $\mathcal{N} = \{1, \ldots, n\}$ denote the set of jobs, $\mathcal{M} = \{1, \ldots, m\}$ the set of possible machine locations and $D \in \mathbb{R}^{n \times m}$ the matrix of distances, i.e., $D(i,k) = dist(i,k)$ is the distance between the location of job i and possible machine location k. Furthermore, let p_i be the processing time of job i and p the number of machines to be located. The release date r_{ik} of job i if processed on a machine in location k is given by the distance between the corresponding locations $r_{ik} = dist\,(i,k)$. The DPMM problem consists of selecting exactly p locations from $\mathcal{M}$ and scheduling all jobs $i \in \mathcal{N}$ on the selected machines such that the location-dependent release dates r_{ik} are respected and $C_{\max} = \max\{C_i | i \in \mathcal{N}\}$ is minimized where C_i is the completion time of job i.

3 Clustering Heuristics

The DPMM ScheLoc Problem consists of three different subproblems: Locating the machines, assigning the jobs to the machines, and scheduling the assigned jobs on each machine. Once the machine locations have been chosen and the jobs have been assigned to the machines, the remaining scheduling problem can be solved optimally in polynomial time since it reduces to p Single Machine Makespan Problems. Therefore, we focus on heuristics that solve the first two subproblems.
If the scheduling data (r_{ik}, p_i) is ignored these two subproblems reduce to an uncapacitated discrete location problem for which various clustering type heuristics exist (see e.g., [7]). Formally, the clustering problem is defined as follows: Given a discrete set of possible locations $\mathcal{M}$, a set of demand locations $\mathcal{N}$ with demand d_i for each $i \in N$ and a distance function $dist\,(i,k)$ for $i \in \mathcal{N}$ and $k \in \mathcal{M}$, find a subset $\mathcal{C} \subset \mathcal{M}$ of size p called cluster centers and an assignment $C_k \subset \mathcal{N}$ of demand locations to cluster centers $k \in \mathcal{C}$ such that $\cup_{k \in \mathcal{C}} C_k = \mathcal{N}$, $C_k \cap C_l = \emptyset$ for all $k, l \in \mathcal{C}$, and $\sum_{k \in \mathcal{C}} \sum_{i \in C_k} d_i dist\,(i,k)$ is minimized. In case of uncapacitated clusters, each demand location i is assigned to the closest cluster center C_k independent of the values d_i.
Given the solution to the uncapacitated clustering problem we can compute a solution to the DPMM ScheLoc Problem by solving p single machine problems. However, to improve the solution of the clustering we have to include the scheduling data in the clustering decisions. To do this we consider the location part (choosing the cluster centers) and the clustering part (finding the assignment sets C_k) and solve the problems sequentially in one of the following forms:

1. First choose all cluster centers then assign the jobs to the cluster centers.
2. First cluster the jobs then assign each cluster to a cluster center.

3. Iteratively choose a cluster center and assign jobs to it until p clusters are obtained.

In each of the subproblems we can integrate the scheduling data. For example we can choose cluster centers k such that $\max_{i \in \mathcal{N}} r_{ik} + p_i$ is minimal to avoid machines being located far from jobs that have large processing times. To choose the clusters we would for example like to balance the sum of processing times assigned to each machine, i.e., we want to minimize $\max_{k \in \mathcal{C}} \sum_{i \in C_k} p_i$. Since the scheduling data cannot be optimally integrated we identified and tested various such criteria.
The drawback of the clustering heuristics is that once a decision has been made it will not be altered afterwards. That means that a poor decision at the beginning of the heuristic will have a large impact on the result of the heuristic.
To overcome this weakness we combined clustering heuristics 1 and 2 to a local search algorithm similar to the algorithm proposed in [7] for the uncapacitated clustering problem. The algorithm alternatingly leaves the clusters unchanged while trying to improve the cluster centers or leaves the cluster centers unchanged while trying to improve the assignment. If an improving move is found this is implemented and the procedure is iterated until no improving move is found.

Tests have been performed for all clustering heuristics as well as the local search with different starting solutions on instances with up to 300 jobs, 60 locations and 50 machines. The local search was tested with different starting solutions, namely the results of the clustering heuristics and a randomly generated solution. The tests showed that the local search is able to significantly improve the solutions of the clustering heuristics within short computation time. For small instances of up to 30 jobs, 10 locations and 8 machines solutions were compared to the optimal results provided by a commercial IP solver. The tests showed that for those small scale problems the clustering heuristics provide optimal or near optimal results. For larger instances the commercial IP solver is not able to provide a feasible solution within reasonable runtime.

Acknowledgements Partially supported by the Federal Ministry of Education and Research Germany, grant DSS_Evac_Logistic, FKZ 13N12229 and by the French National Research Agency as ANR-11-SECU-002-01 (CSOSG 2011).

References

1. Elvikis D., H.W. Hamacher and M.T. Kalsch, Simultaneous Scheduling and Location (ScheLoc): The Planar ScheLoc Makespan Problem, J of Sched, 12, 361-374 (2008)
2. Hennes H., Integration of Scheduling and Location Models, PhD Thesis, University of Kaiserslautern, 2005
3. Hennes H., H.W. Hamacher, Integrated Scheduling and Location Models: Single Machine Makespan Problems, Stud in Locat Anal, 16, 77-90 (2007)
4. Kalsch, M. T., Scheduling - Location (ScheLoc) Models, Theory and Algorithms, PhD Thesis, University of Kaiserslautern (2009)
5. Kalsch M.T., Z. Drezner, Solving scheduling and location problems in the plane simultaneously, Comput and Oper Res, 37, 256-264 (2010)
6. Kaufmann, C., A Polynomial Time Algorithm for an Integrated Scheduling and Location Problem, Proceedings of the 14th International Conference on Project Management and Scheduling, 124-128 (2014)
7. Maranzana, F. E., On the location of supply points to minimize tranport costs, Oper Res Q, 15, 261-270 (1964)

MISTA 2015

A new lower bound for optimisation of energy consumption of robotic cells

Libor Bukata · Přemysl Šůcha · Zdeněk Hanzálek

1 Introduction

Robotic cells, broadly used in automotive industry for assembling and welding, are usually designed under time pressure with the only optimisation requirement to satisfy the production cycle time. As a result, there is a huge potential for the energy saving as robots are usually moving at a maximal speed leading to long waiting times [1]. It is a little surprising that even though the problem is so widespread, there is no effective solution for energy optimisation of robotics cells which would result in money savings and improvements of green credentials of industrial companies.

And therefore, this work can be perceived as a first step in designing of an efficient exact algorithm for the above mentioned problem since a completely new lower bound, which is the most crucial part of these algorithms, based on the Interior Point Method (abbr. IPM) is proposed. The preliminary results revealed that the bound is both fast and tight.

The abstract is structured in the following way. In the next section the related work is discussed shortly. Section 3 provides a brief explanation of the lower bound. And finally, the last section is devoted to experiments and conclusion.

2 Related Work

The research on energy optimisation of robotic cells can be classified into the following categories: local optimisation, and global optimisation. In the local optimisation (e.g. [2]) individual robot trajectories are optimised with respect to the speed, acceleration, and trajectory go-through points in order to reduce the required energy. The global optimisation aims to minimise the aggregated power consumption of the whole robotic cell by considering e.g. different order of operations, speed of robot movements, and stationary positions of robots at the same time.

Libor Bukata · Přemysl Šůcha · Zdeněk Hanzálek
Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Prague, Czech Republic
E-mail: bukatlib@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz

The state-of-the-art work of Wigström et al. [3] can be perceived as a pioneer in the global optimisation of robotic cells. The authors created a model of the robot and solved the optimal control problem by using Dynamic Programming. The trajectory path was fixed and the time optimal trajectory obtained from ABB Robot Studio was required. Afterwards, the locally optimised trajectories were used as an input for the global solver to find a solution that is energy efficient and satisfying demanded production cycle time. Although their algorithm is the first one considering the global energy aspects only small instances were solvable in a reasonable time.

The lower bound presented in this abstract can be perceived as a supporting procedure for exact algorithms. It should enable them to solve industrial-sized problems of the energy optimisation from the global point of view. Even though, the lower bound is a pure continous optimization problem, the full problem is a scheduling one since there are many discrete variables determining e.g. order of operations, power modes, and so on.

3 Lower Bound Based on the Interior Point Method

As our lower bound must be fast and as tight as possible, we have neglected some aspects that we have identified as less important for energy reasoning. In particular, the robots are considered to be independent, i.e. synchronisations are ignored, selection of power saving modes and robot configurations is not taken into account, and collision zones are not addressed. These aspects must be resolved during the branching.

The mathematical formulation in Equation (1) clearly outlines the problem solved for each robot. Basically, the goal of the bound is to find a lower estimation of the robot consumed energy for its movements and waitings with respect to the fixed production cycle time. For each robot movement or waiting there is a convex function $f_i(d_i)$ (see e.g. [3]) that corresponds to the dependence of energy consumption on its duration d_i.

$$\begin{aligned} \text{minimise} \quad & \sum_{i=1}^{n} f_i(d_i) \\ s.t. \quad & \sum_{i=1}^{n} d_i = \mathrm{CT} \\ & \underline{d_i} \leq d_i \leq \overline{d_i} \qquad \forall i \in \{1, \ldots, n\} \end{aligned} \tag{1}$$

Even though, the problem could be solved by general solvers, it is beneficial to use our Adapted Interior Point method (abbr. AIPM) since we have revealed that it is possible to exploit the problem properties in the way of the special form of Hessian matrix. The inversion of the matrix, used in the embedded Newton method, can be accomplished in $O(n^2)$ time complexity that is much faster than general algorithms, and as a consequence, the bound can be evaluated in a fraction of time.

4 Preliminary Results and Conclusion

To verify the correctness of our approach we have implemented the lower bound in Matlab and compared it with a Linear Programming (abbr. LP) formulation, which was solved by IPM, where the convex functions were substituted by piece-wise linear

functions. By the method of sampling, i.e. a lower or upper envelope of the convex function, the optimal solution of the LP formulation is either the lower or upper bound. As it is shown in Fig. 1 the calculated lower bound of AIPM is always lower than the LP upper bound.

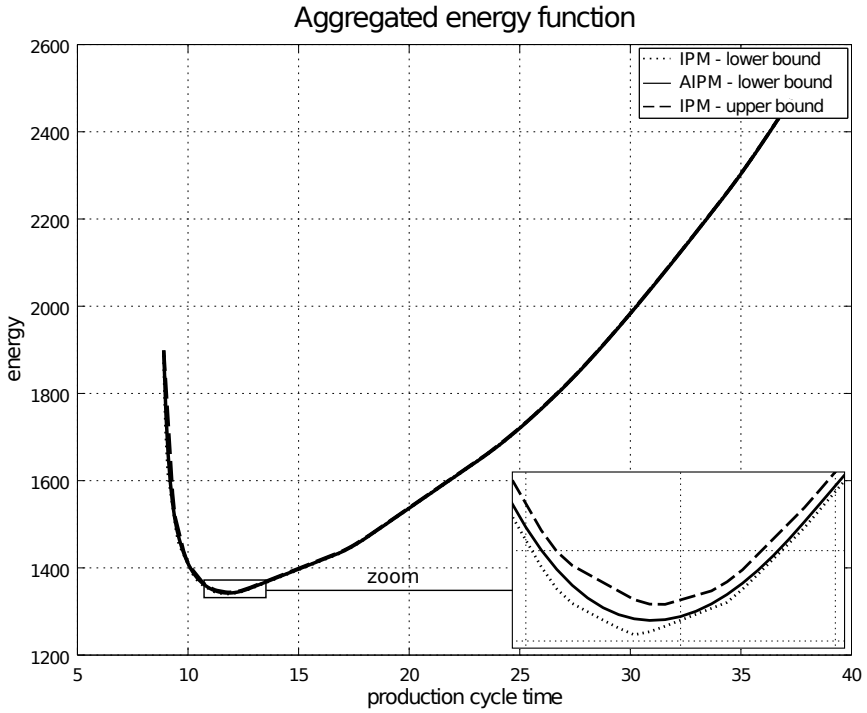


Fig. 1 Dependence of the energy consumption on the production cycle time.

Our preliminary results obtained from 4 randomly generated instances are presented in Table 1 where 'LP size' is the size of the LP formulation in the form of 'number of constrains' x 'number of continuous variables'. It can be seen that our AIPM outperforms the Matlab implementation of IPM about 2.11 to 8.5 in terms of speedup. Even though the acceleration is not so high at first sight, we expect that the algorithm will be boosted at least about a factor of ten by re-implementing it to native code. Moreover, the exact inversion seems to be very suitable for code vectorisation.

number of energy functions	IPM (LP)	LP size	AIPM	IPM/AIPM
3	22.0 ms	60 x 6	2.57 ms	8.56
6	25.6 ms	120 x 12	5.28 ms	4.85
9	30.0 ms	180 x 18	7.82 ms	3.84
60	67.8 ms	1200 x 120	32.2 ms	2.11

Table 1 Preliminary results for the lower bound.

So, in summary, we have proposed the Adapted Interior Point Method exploiting the problem specific properties. The resulting lower bound is faster than general methods and seems to be suitable for effective bounding in exact algorithms.

Acknowledgements This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS13/209/OHK3/3T/13.

References

1. D. Meike, M. Pellicciari, and G. Berselli. Energy efficient use of multirobot production lines in the automotive industry: Detailed system modeling and optimization. *Automation Science and Engineering, IEEE Transactions on*, 11(3):798–809, July 2014.
2. Koen Paes, Wim Dewulf, Karel Vander Elst, Karel Kellens, and Peter Slaets. Energy efficient trajectories for an industrial ABB robot. *Procedia CIRP*, 15(0):105 – 110, 2014. 21st CIRP Conference on Life Cycle Engineering.
3. O. Wigstrom, B. Lennartson, A. Vergnano, and C. Breitholtz. High-level scheduling of energy optimal trajectories. *Automation Science and Engineering, IEEE Transactions on*, 10(1):57–64, Jan 2013.

MISTA 2015

A heuristic procedure for the personnel task re-scheduling problem

Broos Maenhout · Jeroen Burgelman · Mario Vanhoucke

1 Introduction

When composing a duty timetable, a personnel planner makes different assumptions with respect the demand for staff, the duration of tasks and the availability of personnel. However, personnel schedules are often disrupted by unexpected events as a result of the uncertainty of demand, uncertainty of arrival and/or uncertainty of capacity ([11]). These schedule disruptions may render the personnel schedule infeasible and adaptations to this original roster are necessary to restore its workability.

In this research, we consider the personnel re-scheduling problem that restores the feasibility and/or workability of a personnel roster. As a reaction to disruptions, a new feasible schedule needs to be constructed that contains as few as possible deviations compared to the original roster. In the literature, all re-scheduling problems have been studied in a deterministic setting assuming that all disruptions are known at the moment of decision-making. Different network-based mathematical formulations have been devised by [7,8] for the nurse re-rostering problem. [9], [10], [5] and [6] developed meta-heuristic evolutionary algorithms as a solution methodology for this personnel shift scheduling problem. Personnel re-scheduling is also studied in the transportation industry. [2] and [1] discuss a literature overview for the real-time task-based crew recovery problem in the airline industry and the railway sector respectively.

Different from the literature, we investigate in this research the personnel re-scheduling problem in a dynamic setting where the personnel planner does not know if and when disruptions will arise. We consider the general personnel task scheduling problem ([4]) where the disruptions may the result of three sources of uncertainty, i.e.

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: Broos.Maenhout@Ugent.be

Jeroen Burgelman
Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: Jeroen.Burgelman@Ugent.be

Mario Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Belgium
Vlerick Business School, Belgium
University College London, United Kingdom
E-mail: Mario.Vanhoucke@Ugent.be

uncertainty of capacity, arrival and/or demand. In this setting, we propose a heuristic algorithm that makes use of different recovery procedures depending on the type of disruption.

2 Problem description

We consider the Personnel Task Scheduling Problem (PTSP) ([4]), which assigns a set of fixed contiguous tasks to a set of available homogeneous workers who work according to shifts. All shifts have a fixed length of 8 hours. The tasks have fixed start and finish times. Tasks cannot be interrupted and restarted later in the planning horizon but can be continued by another worker. Different from the literature, the personnel task scheduling problem under study considers a planning horizon of multiple days. In this context, time-related constraints are imposed upon the construction of a feasible schedule for an individual employee.

In this research, we consider the related reactive personnel task re-scheduling problem. As disruptions occur unexpectedly during the execution of the personnel roster and not all disruption information is known at the beginning of the planning period, re-scheduling decisions need to be taken in a dynamic setting. We consider three types of disruptions, i.e.

- Uncertainty of capacity: The personnel resources may become unavailable to work a particular shift, day or longer period of time.
- Uncertainty of demand: Due to variability in demand, the demand can be lower or higher than expected.
- Uncertainty of arrival: The processing time of tasks can be lower or higher than expected.

The personnel planner may react to these disruptions and restore the workability by adapting the personnel roster with as minimal schedule changes as possible, which embodies the objective function. This re-rostering decision is threefold, i.e. the decision to re-schedule or not at a certain point in time, the length of the re-scheduling period and the type of recovery procedure.

3 Solution methodology

The proposed solution methodology imitates the real-life decision process. Information on the uncertainty of capacity, demand and arrival is generated at certain disruption information points through *simulation*. Taking this knowledge into account, a re-scheduling decision is undertaken by means of a *heuristic optimisation procedure* at certain re-rostering decision points to restore the workability of the personnel roster.

Simulation of uncertainty
Disruptions are generated by a Monte Carlo simulation through independent Bernouilli trials. For each type of uncertainty, probability distributions with adequate parameters are conjectured in a simulation experiment. Multiple simulation runs will enable us to examine the robustness of different solution strategies. The (un-)availability of employees is modelled as a discrete time stochastic process such that a capacity disruption can last for several days. The demand for staff is simulated for all tasks or new tasks

are generated using a particular distribution (e.g. the Poisson distribution). There is also variability in the task duration, which is modelled by a triangular distribution ([3]).

A heuristic optimisation procedure
The re-scheduling is undertaken by an evolutionary heuristic with problem-specific operators that considers only a limited number of tasks, personnel members and days when re-scheduling. Moreover, the implemented variable neighbourhood search incorporates problem-specific information where the applied recovery technique is dependent upon the type of encountered disruption, the problem characteristics and possible phase transitions in problem complexity.

4 Computational experiments

We provide computational insights into our heuristic procedure and shows the benefits of the adaptive variable neighbourhood search compared to a non-dedicated re-scheduling method. Moreover, we utilise the proposed procedure as a simulation tool and investigate the impact of different policy decisions and strategies.

References

1. Cacchiani, V., Huisman, D., Kidd, M., Kroon, L., Toth, P., Veelenturf, L., and Wagenaar, J., An overview of recovery models and algorithms for real-time railway rescheduling, Transportation Research Part B, 63, 1537 (2014)
2. Clausen, J., Larsen, A., Larsen, J., and Rezanov, N., Disruption management in the airline industry: Concepts, models and methods, Computers & Operations Research, 37, 809821 (2010)
3. Johnson, D., The triangular distribution as a proxy for the beta distribution in risk analysis, The Statistician, 46, 387398 (1997)
4. Krishnamoorthy, M. and Ernst, A., The personnel task scheduling problem, Applied Optimization, 52, 343368 (2001)
5. Maenhout, B. and Vanhoucke, M., An evolutionary approach for the nurse reros- tering problem, Computer & Operations Research, 38, 14001411 (2011)
6. Maenhout, B. and Vanhoucke, M., An artificial immune system based approach for solving the nurse re-rostering system, Lecture Notes in Computer Science, 7832, 97 108 (2013)
7. Moz, M. and Pato, M. (2003). An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals of Operations Research, 119:285301 (2003)
8. Moz, M. and Pato, M., Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models, Annals of Operations Research, 128:179197 (2004)
9. Moz, M. and Pato, M., A genetic algorithm approach to a nurse rerostering problem, Computers & Operations Research, 34:667691 (2007)
10. Pato, M. and Moz, M., Solving a bi-objective nurse rerostering problem by using a utopic Pareto genetic heuristic, Journal of Heuristics, 14, 359374 (2008)
11. Van den Bergh, J., Belien, J., De Bruecker, P., Demeulemeester, E., and De Boeck, L., Personnel scheduling: A literature review. European Journal of Operational Research, 226, 367385 (2013)

MISTA 2015

A heuristic procedure to proactively increase employee substitutability and personnel roster robustness

Jonas Ingels · Broos Maenhout

1 Introduction

The management and planning of personnel is a widely studied topic in the operational research literature [10,11,5,4]. In personnel planning, three hierarchical phases can be distinguished, i.e. the strategic staffing phase, the tactical scheduling phase and the operational allocation phase [2,5]. In the staffing phase, the personnel mix and budget required to meet the service demand in the long term is determined. This phase focuses on the determination of the personnel characteristics, e.g. the competencies and degree of employment, and subsequently constrains the decision freedom in the tactical scheduling phase. During this phase, a medium-term personnel roster is constructed based on predictions and assumptions. However, these predictions and assumptions may prove to be incorrect in the operational allocation phase This uncertainty negatively impacts the workability of the original roster and requires adjustments to be executed in the short-term.

In personnel scheduling, three sources of uncertainty exist, i.e. uncertainty of demand, uncertainty of capacity and uncertainty of arrival [4]. Organisations need to adopt a proactive and a reactive approach to deal with this uncertainty.
During the scheduling phase, organisations can proactively build in robustness in their personnel rosters. This improves the absorption and adjustment ability of the original roster when unexpected events occur in the operational allocation phase. The robustness of a personnel roster can be improved by using capacity buffers and time buffers. Time buffers can be used in project management and personnel task scheduling problems [9,15]. Capacity buffers include the scheduling of reserve crew or the definition of preferred requirements that are higher than the minimum requirements [7,6,17]. Alternatively, unit crewing can also improve the roster robustness by keeping crew with different skills together for as long as possible in a pairing [16]. Unit crewing

Jonas Ingels
Faculty of Economics and Business Administration, Ghent University
E-mail: jonas.ingels@ugent.be

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University
E-mail: broos.maenhout@ugent.be

is especially useful when there are limited opportunities to recover from disruptions by resource substitution. In general, resource substitution is a good indicator for the robustness in crew and aircraft scheduling [8]. Shebalov and Klabjan [14] proactively maximise the number of move-up crews that can exchange tasks to overcome operational disruptions. The authors state that crew substitution is a cost-efficient option when disruptions occur.
Thus, robustness involves both stability and flexibility. On the one hand, a stable schedule exhibits a high absorption ability and subsequently a small number of changes when the operating environment changes. On the other hand, a flexible schedule has a high adjustment ability and provides sufficient change possibilities to efficiently recover from unexpected events [12,8].

In this paper, we consider a personnel shift scheduling problem in which employees are assigned to cover the demand for shifts and skills. We proactively improve the short term adjustment ability or flexibility of the original personnel roster by maximising the substitution possibilities between different personnel members. An employee substitution is available when at least one employee can take over the skill-shift assignment of another employee on a particular day. The total number of substitution possibilities for a skill and shift on a day is the product of the number of assigned employees to the skill-shift combination and the number of employees that can be reassigned to that combination on the same day.

2 Methodology

We adopt a proactive approach to improve the personnel roster robustness by maximising the number of build-in substitution possibilities between personnel members. This proactive approach entails the application of a population-based evolutionary algorithm to construct a personnel roster for a medium-term period. The algorithm encompasses three stages: i.e. the construction of an initial population of rosters, local search and combination of personnel rosters.
First, we construct initial rosters by exploiting a two-phase algorithm. In the first phase, we build a days-on days-off schedule for every employee. In the second phase, we assign shifts to the employees based on the days-on days-off schedule.
Next, we increase the number of substitution possibilities through local search. During the local search, a neighbourhood structure is selected and the personnel roster is searched based on a chosen guiding order. We propose three neighbourhood structures, i.e. a cell-by-cell search, a horizontal schedule search and a vertical schedule search.
Furthermore, we apply an evolutionary cycle where the population elements are subject to roster combination and roster repair. The roster combination includes employee- and day-based crossovers [13]. The repair function ensures the satisfaction of the time-related constraints. After the execution of each evolutionary cycle, we update the population and repeat the cycle until a stop criterion is met.

We validate the improved robustness of a personnel roster by applying a three-step methodology [3,1]. After the construction of the personnel roster in the first step, we test its applicability in the operational allocation phase and evaluate the proactively build-in adjustment ability of the tactical personnel roster. Based on the associated

costs, we assess different strategies to improve the adjustment ability in the operational allocation phase.

3 Computational experiments

We generate test instances for 10, 20 and 40 employees for a planning horizon of 7 days and a personnel shift scheduling problem with a multi-skilled heterogeneous workforce. These test instances can be classified as easy or hard depending on the imposed time-related constraints. The hard test instances include all the imposed time-related constraints, i.e. one assignment per day, forward rotation of shift assignments, minimum and maximum number of assignments and maximum consecutive number of assignments. In the easy test instances, we assume that all duties are day duties relaxing the forward rotation constraint.
The hard test instances are used to show the validity of each individual building block of our algorithm. The easy test instances allow us to show the validity of the complete algorithm. For this purpose, we compare the results of our algorithm with the results obtained with optimisation software.
We compare the robustness of three strategies, i.e. a pure maximisation of the number of substitution possibilities, a maximisation of substitution possibilities with a constraint on the allowed cost increase and a maximisation of the lowest number of substitution possibilities on a given day during the planning horizon. These strategies are evaluated based on the number of unresolved shortages, the number of executed substitutions and the total assignment cost.

References

1. Abdelghany, K., Abdelghany, A., Ekollu, G.: An integrated decision support tool for airlines schedule recovery during irregular operations. European Journal of Operational Research **185**(2), 825 – 848 (2008)
2. Abernathy, W., Baloff, N., Hershey, J.: A three-stage manpower planning and scheduling model a service sector example. Operations Research **21**, 693–711 (1973)
3. Bard, J., Purnomo, H.: Hospital-wide reactive scheduling of nurses with preference considerations. IIE Transactions **37**, 589–608 (2005)
4. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**, 367–385 (2013)
5. Burke, E., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The state of the art of nurse rostering. Journal of Scheduling **7**, 441–499 (2004)
6. De Causmaecker, P., Vanden Berghe, G.: Relaxation of coverage constraints in hospital personnel rostering. Lecture Notes in Computer Science **2740** (2003)
7. Dowsland, K., Thompson, J.: Solving a nurse scheduling problem with knapsacks, networks and tabu search. Journal of the Operational Research Society **51**, 825–833 (2000)
8. Dück, V., Ionescu, L., Kliewer, N., Suhl, L.: Increasing stability of crew and aircraft schedules. Transportation Research Part C: Emerging Technologies **20**(1), 47 – 61 (2012)
9. Ehrgott, M., Ryan, D.M.: Constructing robust crew schedules with bicriteria optimization. Journal of Multi-Criteria Decision Analysis **11**(3), 139–150 (2002)
10. Ernst, A., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D.: An Annotated Bibliography of Personnel Scheduling and Rostering. Annals of Operations Research **127**, 21–144 (2004)
11. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**, 3–27 (2004)
12. Ionescu, L., Kliewer, N.: Increasing flexibility of airline crew schedules. Procedia - Social and Behavioral Sciences **20**, 1019 – 1028 (2011)

13. Maenhout, B., Vanhoucke, M.: Comparison and hybridization of crossover operators for the nurse scheduling problem. Annals of Operations Research **159**, 333–353 (2008)
14. Shebalov, J., Klabjan, D.: Robust Airline Crew Pairing: Move-up Crews. Transportation Science **40**, 300–312 (2006)
15. Tam, B., Ehrgott, M., Ryan, D.M., Zakeri, G.: A comparison of stochastic programming and bi-objective optimisation approaches to robust airline crew scheduling. OR Spectrum **33**(1), 49–75 (2011)
16. Tam, B., Ryan, D., Ehrgott, M.: Multi-objective approaches to the unit crewing problem in airline crew scheduling. Journal of Multi-Criteria Decision Analysis **21**(5-6), 257–277 (2014). DOI 10.1002/mcda.1517. URL http://dx.doi.org/10.1002/mcda.1517
17. Topaloglu, S., Selim, H.: Nurse scheduling using fuzzy modelling approach. Fuzzy Sets and Systems **161**, 1543–1563 (2010)

MISTA 2015

Structural properties of an open problem in preemptive scheduling

Bo Chen · Ed Coffman · Dariusz Dereniowski · Wiesław Kubiak

1 Introduction

Structural properties of optimal preemptive schedules have been studied in a number of recent papers. These papers focus mainly on two characteristics: the minimum number of preemptions necessary, and the minimum required sizes of *shifts*, i.e., the sizes of uninterrupted intervals bounded by job preemptions, starts, resumptions and completions. These two characteristics have been investigated for a large class of preemptive scheduling problems, but so far only crude bounds for them have been derived for specific problems. This paper sharpens the bounds on these characteristics for a well-known open problem in the theory of preemptive scheduling: Instances consist of in-trees of n unit-execution-time (UET) jobs with release dates r_j, and the objective is to minimize the total completion time $\sum C_j$ on two processors. For simplicity we denote this problem by $\mathcal{P}$; in the standard 3-field notation the problem specification is $P2|pmtn, in\text{-}trees, r_j, p_j = 1|\sum C_j$. $\mathcal{P}$ is among the current, tantalizing "threshold" problems of scheduling theory. Our brief literature survey in the next section reveals that the more interesting generalizations lead to NP-hard problems and nontrivial simplifications lead to tractable problems with polynomial-time solutions.

For problem $\mathcal{P}$, we show that the number of preemptions necessary for optimality need not exceed $2n-1$ and that the minimum shift sizes need not be less than 2^{-2n+1}. These bounds are obtained by combinatorial analysis of optimal preemptive schedules rather than by the analysis of polytope corners for linear-program formulations of the problem, an approach to be found in earlier papers. In particular, our approach establishes, as the centerpiece of the paper, a normal form for optimal preemptive schedules, whereby minimal shift sizes of a job

B. Chen
Centre for Discrete Mathematics and Its Applications (DIMAP) and Warwick Business School, University of Warwick, Coventry, UK. E-mail: bo.chen@wbs.ac.uk

E. Coffman
Departments of Electrical Engineering and of Computer Science, Columbia University, New York, USA. E-mail: coffman@cs.columbia.edu

D. Dereniowski
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland. E-mail: deren@eti.pg.gda.pl

W. Kubiak
Faculty of Business Administration, Memorial University, St. John's, Canada. E-mail: wkubiak@mun.ca

are exponentially decreasing functions. Specifically, the first interval between a job's start and its first preemption must be a multiple of $1/2$. Subsequent intervals between the preemptions, starts and completions of jobs must have sizes that are multiples of $1/4$, $1/8$, and in general, the i-th preemption must occur after a shift having a size that is a multiple of 2^{-i}. The bounds on structural characteristics follow immediately from the normal-form theorem of Section 3. Finally, we show that there exists a sequence of instances of $\mathcal{P}$ indexed by n such that in any corresponding sequence of optimal preemptive schedules, the maximum of the number of preemptions required by jobs grows logarithmically in n.

In summary, our results on the structural characteristics of the preemptions in optimal schedules for $\mathcal{P}$

- greatly reduce the solution space of optimal preemptive schedules for $\mathcal{P}$,
- improve our understanding of possible preemption structures as a step towards determining the complexity of $\mathcal{P}$. On the one hand, normal-form optimal schedules may lead us to a polynomial-time algorithm. On the other hand, the fact that a single job may need as many as order $\log n$ preemptions could be useful in proving NP-completeness,
- significantly improve the lower bound on the resolution of $\mathcal{P}$.

In the next section we briefly review the literature bearing on our problem in the theory of preemptive scheduling. Section 3 formally states our new results, then describes the general approach of the proofs. The section concludes with a sequence of instances that exhibit a growth logarithmic in n of the maximum number of preemptions needed by individual jobs in optimal preemptive schedules.

2 Background

In broad terms, this section considers optimal preemptive scheduling for $n > 1$ jobs with release dates $\{r_j\}$ on $m > 1$ parallel processors under the total completion time criterion ΣC_j. Within this class of problems, various choices for the parameters m, the job execution times $\{p_j\}$, and the precedence constraints $\prec$ will be made and the tractability of the resulting problems assessed.. The problems are denoted collectively by $P|pmtn, \prec, r_j, p_j|\sum C_j$. $\mathcal{P}$ has been studied by many as a natural sequel to other problems, each with more constrained instances and a polynomial-time optimization algorithm. As primary examples, in $\mathcal{P}$ the number m of processors can be reduced to 1, Baptiste et al [1]; precedence constraints can be eliminated, Herrbach, Lee, and Leung [7]; differing release dates can be eliminated, Coffman, Sethuraman, and Timkovsky [6]; and in-tree precedence constraints can be replaced by out-tree constraints, Baptiste and Timkovsky [3].

Research in optimal preemptive scheduling has often turned to the study of measures of schedule structure, particularly with NP-hard problems and open problems like $\mathcal{P}$. Two characteristics have been of special interest, one being the smallest number of preemptions needed by any optimal schedule, which dates back to the origins of preemptive scheduling theory, McNaughton [8]. The more recent one concerns the durations of uninterrupted intervals of job execution in optimal preemptive schedules. These durations, the shifts introduced in Section 1, are bounded by scheduling events, i.e., job starts, completions, and preemptions. Under the UET assumption, the smallest shift required for the optimality of a preemptive schedule of a given problem instance is known as the *resolution* of that instance. The smallest resolution over all problem instances is the *problem* resolution. These characteristics have been investigated for a large class of preemptive scheduling problems by Baptiste et al [2] who give general bounds on these quantities. Coffman, Ng and Timkovsky

[5] provide further bounds on the resolutions of various scheduling problems. In particular, they show for our problem $\mathcal{P}$ a resolution upper bound of $2^{-(n-1)/3}$ and a corresponding lower bound of $(n+2)^{-(2n+1)/2}$. In [10], [2] and [5] bounds are obtained on problem resolutions by analyzing the corners of feasibility regions of linear programs designed for the problem. Our approach is combinatorial and does not make use of the theory of linear programming. Our approach gives a lower bound exponentially small in n, which is a significant improvement for intrees over the essentially factorially small bound that follows from the results in [5]. Results are formalized more precisely in the next section.

3 Results

An analysis of optimal preemptive scheduling has often come down to proving that for every optimal schedule there exists an equivalent one in some given, highly structured normal form, e.g., a sequence of small blocks each having a very simple structure. Equivalence here means that the value of the scheduling criterion is unchanged. This general approach to preemptive scheduling problems dates back to research done in the late '60's by Muntz and Coffman and published in [9]. The ultimate result was an optimization algorithm for UET, 2-processor, makespan scheduling with the precedence relation arbitrary and preemptions allowed. It is further illustrated in the work of Coffman and Garey [4] on the same basic problem instances. They proved a bound on the relative increase in optimal makespans when preemptions are disallowed. The proofs of strong normal-form results are usually quite daunting, with lengthy, intricate case analysis being typical. This remark also applies to the proof of our new normal-form theorem. This last result for the instances of $\mathcal{P}$ is given next.

Theorem 1 *For every instance of problem $\mathcal{P}$ there exists a normal-form optimal schedule in which every shift is a multiple of* $1/2^{2n}$.

The proof of this result shows that an optimal schedule can be constructed as a sequence of *blocks*, each with at most three jobs. No job starts or completes inside a block but there is at least one job start or completion at the beginning of a block, and at least one job completion at the end of a block. A block is called *l-normal* if each job duration in the block is a multiple of $1/2^{l+1}$, and the block length is a multiple of $1/2^{l}$. In a normal-form schedule the first block must be 1-normal, the second 2-normal and so on. The proof verifies that, in a normal-form schedule with q blocks, each preemption occurs at a multiple of $1/2^{q+1}$, where $q \leq 2n-1$. The ultimate goal is to show that there exists an optimal schedule that has normal form. The proof is by contradiction. The initial assumption is that an optimal schedule is also *maximal* in the sense that it has a latest possible *abnormality* point i, i.e., a latest block i which is not i-normal. The proof shows that such a block must have exactly three jobs. One completes at the end of the block and has an $(i+1)$-normal duration, but the durations of the other two are not $(i+1)$-normal. These two jobs then trigger an *alternating* chain of jobs to which they also belong. The completion times of the jobs in the chain are not $(i+1)$-normal which makes it possible under normal-form block circumstances to either extend the chain by one job or prove that the abnormality point must exceed i; this is the key result of the proof. Thus, a contradiction appears in either case since the number of jobs is finite and the schedule is maximal. The normal-form block circumstances here mean that the alternating chain does not end with a certain structure that we call an *A-configuration*, a configuration that prevents an extension of the alternating chain. However, it is shown that there always exists a maximal schedule which does not include an A-configuration. The main result of

the paper follows and states that there is a normal schedule that is optimal for $\mathcal{P}$.

Finally, we exhibit sequences of problem instances indexed by n for which the rate at which the maximum number of preemptions required by jobs in optimal preemptive schedules of these instances increases logarithmically in n. Let A_i, $i \geq 0$, be a set of four jobs, denoted by $a_1^i, a_2^i, a_3^i, a_4^i$ such that $r(a_j^i) = 2i$ for each $j \in \{1, 2, 3\}$, $r(a_4^i) = 2i + 1$ and $a_j^i \prec a_4^i$ for each $j \in \{1, 2, 3\}$. Then, define the set of jobs $\bigcup_{i=0}^{p} A_i$, where $a_4^i \prec a_4^{i+1}$ for each $i \in \{0, \ldots, p-1\}$. (See Figure 1.)

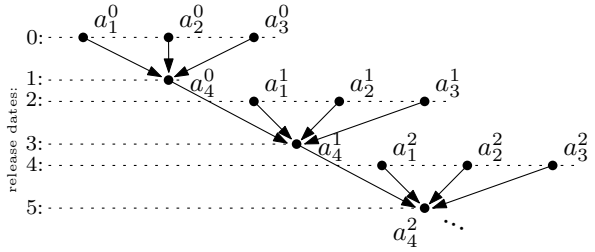


Fig. 1 The precedence constraints and job release dates, $p \geq 2$

It can be proved that the job a_4^p should complete exactly at $2p + 3 - 1/2^{p+1}$ in any optimal schedule. This is done by first proving that no valid schedule (optimal or not) can complete a_4^p earlier, and then by proving that starting a_4^p later leads to a schedule that cannot be optimal. Omitting further details, we have the following result supported by Figure 1.

Theorem 2 *With I an instance of $\mathcal{P}$, define $\rho(I)$ as the largest integer such that every optimal preemptive schedule for I has at least one job that is preempted $\rho(I)$ times. Then there exists a sequence of instances $\{I_n\}_{n\geq 1}$ of $\mathcal{P}$ indexed by the number n of jobs such that*

$$\rho(I_n) = \Omega(\log n).$$

References

1. P. Baptiste, P. Brucker, S. Knust, and V.G. Timkovsky. Ten notes on equal-processing-time scheduling. *4OR*, 2(2):111–127, 2004.
2. P. Baptiste, J. Carlier, A. Kononov, M. Queyranne, S. Sevastyanov, and M. Sviridenko. Properties of optimal schedules in preemptive shop scheduling. *Discrete Applied Mathematics*, 159(5):272–280, 2011.
3. P. Baptiste and V.G. Timkovsky. On preemption redundancy in scheduling unit processing time jobs on two parallel machines. *Oper. Res. Lett.*, 28(5):205–212, 2001.
4. E. G. Coffman, Jr. and M. R. Garey. Proof of the 4/3 conjecture for preemptive vs. nonpreemptive two-processor scheduling. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 241–248, New York, NY, USA, 1991. ACM.
5. E.G. Coffman Jr., C.T. Ng, and V.G. Timkovsky. How small are shifts required in optimal preemptive schedules? *Journal of Scheduling*, pages 1–9, 2013.
6. E.G. Coffman Jr., J. Sethuraman, and V.G. Timkovsky. Ideal preemptive schedules on two processors. *Acta Inf.*, 39(8):597–612, 2003.
7. Lee A. Herrbach and Joseph Y.-T. Leung. Preemptive scheduling of equal length jobs on two machines to minimize mean flow time. *Operations Research*, 38(3):487–494, 1990.
8. R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.
9. R.R. Muntz and E.G. Coffman. Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.*, 18(11):1014–1020, 1969.
10. N. W. Sauer and M. G. Stone. Rational preemptive scheduling. *Order*, 4:195–206, 1987.

MISTA 2015

Solving resource constrained shortest path problems with branch-and-cut

Markó Horváth · Tamás Kis

1 Introduction

The resource constrained shortest path problem (RCSPP) is an extension of the familiar shortest path problem. Given a directed graph $D = (V, A)$, where V is a finite set of nodes, and $A \subseteq V \times V$ is a finite set of arcs, a cost function $c : A \to \mathbb{Z}$ on the arcs, and two selected nodes $s, t \in V$, with $s \neq t$. In addition, we also have m resource functions $w^r : A \to \mathbb{Z}$ $(r = 1, \ldots, m)$ on the arcs representing certain resource consumptions, and for each resource r we have a resource limit $W^r \in \mathbb{Z}$. In the (elementary) resource constrained shortest path problem a minimal cost directed (elementary) path P from s to t is sought, such that the resource consumptions along the path do not exceed the resource limits.

The RCSPP has several application fields in practice, see e.g., [1, 5]. Several types of methods have been proposed to solve variants of RCSPP, for an overview, see e.g., [3]. We are focusing on methods based on polyhedral characterization of the set of feasible solutions and in particular those using cutting planes. For instance, Avella et al. [1] use cutting planes to solve RCSPP with negative cycles. In Jepsen et al. [4], a model with resource weights on the nodes is considered, and generalized subtour elimination inequalities are proposed to ensure elementary paths. The authors also suggest generalized capacity inequalities, but computational experiments show that they are not effective in practice. Garcia [2] propose several types of valid inequalities for the polytope of feasible solutions of RCSPP, some of which being valid for RCSPP with cycles, while others only for the acyclic special case. A detailed computational evaluation shows the merits of the various classes in solving the two variants of the problem by branch-and-cut.

Our main results are new cutting planes for solving RCSPP by branch-and-cut, along with exact and heuristic separation methods. Our results extend those of Garcia [2]. We also prove that a class of inequalities proposed by Garcia is NP-hard to separate. In the following sections we briefly summarize these results.

Markó Horváth and Tamás Kis
Institute for Computer Science and Control, Hungarian Academy of Sciences
E-mail: {marko.horvath, tamas.kis}@sztaki.mta.hu

2 New valid inequalities for the RCSPP polytope

We can formulate the RCSPP by a mathematical program in which there is a binary decision variable x_a on each arc $a \in A$ indicating whether the path sought goes through the arc or not.

$$\min \sum_{a \in A} c_a x_a \tag{1}$$

$$\text{s.t.} \sum_{a \in \delta^{out}(i)} x_a - \sum_{a \in \delta^{in}(i)} x_a = \begin{cases} 1 \text{ if } i = s \\ 0 \text{ if } i \in V \setminus \{s,t\} \\ -1 \text{ if } i = t \end{cases}, \quad i \in V \tag{2}$$

$$\sum_{a \in A} w_a^r x_a \leq W^r, \quad r = 1, \ldots, m \tag{3}$$

$$x \in \{0,1\}^A \tag{4}$$

In the above formulation, $\delta^{in}(i)$ and $\delta^{out}(i)$ denote the set of arcs entering and leaving node i, respectively. The objective function (1) expresses the total cost of the path sought. Constraints (2) along with (4) ensure that the feasible solutions are paths. The resource usage of the paths are bounded by (3). If we need elementary paths, and the graph D contains cycles, then additional means are needed to eliminate cycles in the solution. The polytope of feasible solutions of RCSPP is $Q^{RCSPP} = \{x \in \mathbb{R}_+^A : (2) \text{ and } (3) \text{ hold for } x\}$.

Fix a pair of arcs e_1, e_2 of D with $e_1 \neq e_2$, and let $i_1 := tail(e_1)$, $j_1 := head(e_1)$ and $i_2 := tail(e_2)$, $j_2 := head(e_2)$. A necessary condition for a resource-feasible path π visiting e_1 and e_2 in this order to exist is that for each resource r, the inequality

$$\sigma_{s,i_1}^r + w_{e_1}^r + \sigma_{j_1,i_2}^r + w_{e_2}^r + \sigma_{j_2,t}^r \leq W^r \tag{5}$$

holds, where $\sigma_{i,j}^r$ is the length of the shortest path from node i to j with respect to resource weights w^r. Assuming that there is no directed path from the head of e_2 to the tail of e_1, any directed s–t path containing both e_1 and e_2 must go through e_1 first, and then through e_2. In the new *cut-based inequalities*, we fix a pair of arcs with the above condition, and consider cuts (in turn) separating s from i_1, j_1 from i_2, and j_2 from t, respectively. Those cuts give rise to new classes of valid inequalities for Q^{RCSPP} by observing that an s–t path P through e_1 and e_2 can pass through only such an arc of, say, an s–i_1 cut which yields a resource feasible path with respect to resource weights w^r. The form of the new cuts is

$$x(F) \geq x_{e_1} + x_{e_2} - 1, \tag{6}$$

where F is an appropriate subset of arcs of an s–i_1 cut, or i_2–j_1 cut, or j_2–t cut. In fact, this class of inequalities extends that of Garcia, in which only one arc is fixed.

Our second class of new valid inequalities is based on infeasible subpaths. Again, we fix a pair of arcs e_1 and e_2 such that there is no directed path from the head of e_2 to the tail of e_1, and then we consider a subpath P' between the head of e_1 and the tail of e_2 which is resource infeasible for one of the resources. Now, any resource feasible directed P path through e_1 and e_2 must pass these arcs in this order due to the above condition, and must leave P' at some point between e_1 and e_2. This observation gives rise to a new class of valid inequalities of the same form as (6), but with F chosen from

those arcs leaving P'. This class generalizes the infeasible subpath based inequalities of Garcia, as those inequalities have been defined with respect to one fixed arc only. We have also shown that both Garcia's inequalities and ours in this class are NP-hard to separate, and provided a separation heuristic.

3 Variable fixing and primal heuristic

There are many ways to improve the performance of a branch-and-bound procedure. For instance, finding a feasible integer solution may strengthen the actual upper bound, hence we developed a depth-first-search based feasible solution search heuristic to find an s–t path which is feasible for all resource constraints when a fractional solution to the LP-relaxation is available. One can also improve the branch-and-bound procedure by strengthening the LP relaxation by fixing some variables. We developed a cost based variable fixing method to fix some variables to 0.

4 Computational results

We implemented our RCSPP solver in C++ language by using the FICO XPRESS callable library as a branch-and-cut framework. We used randomly generated instances to perform our computational experiments. We have compared the cuts of Garcia to our generalizations, and found that our generalized infeasible subpath based inequalities give better results than those of Garcia, but there is no clear advantage of using our generalization of cut based inequalities. We have also found that a good combination of cutting planes, primal heuristics and variable fixing methods gives significantly better results than FICO XPRESS after tuning.

Acknowledgements This work has been supported by the OTKA grant K112881, and by the NFÜ grant ED_13-2-2013-0002. The research of Tamás Kis has been supported by the János Bolyai research grant BO/00412/12/3 of the Hungarian Academy of Sciences.

References

1. Avella, P., Boccia, M., and Sforza, A., Resource constrained shortest path problems in path planning for fleet management, Journal of Mathematical Modeling and Algorithms, 3, 1–17 (2004)
2. Garcia, R., Resource Constrained Shortest Paths and Extensions, PhD thesis, Georgia Institute of Technology (2009)
3. Irnich, S. and Desaulniers, G., Shortest path problems with resource constraints, 33–65. Springer US (2005)
4. Jepsen, M., Petersen, B., Spoorendonk, S., A Branch-and-Cut Algorithm for the Elementary Shortest Path Problem with a Capacity Constraint, Technical report, DIKU (2008)
5. Steinzen, I., Gintner, V., Suhl, L., and Kliewer, N., A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots, Transportation Science, 44(3), 367–382 (2010)

MISTA 2015

Lower bounds for Scheduling Problems with production and consumption of resources

Abderrahim Sahli · Jacques Carlier · Aziz Moukrim

1 Introduction

The Event Scheduling Problem with Consumption and Production of Resources (ESPCPR) [4] is an extension of the Resource Constrained Project Scheduling Problem (RCPSP) where activities are replaced by events which can produce or consume the resources. ESPCPR consists in scheduling a group of events limited by precedence and resource constraints in order to minimize the makespan. Some other authors have worked on models similar to the ESPCPR. We can quote the works of Neumann and Schwindt [6] and of Laborie [5].

The aim of this paper is to introduce for ESPCPR new lower bounds whose calculation serves two goals. First, lower bounds can help in deleting useless nodes in a branch-and-bound tree in order to decrease the computation time. Second, they may help in evaluating the efficiency of heuristic methods when optimal solutions are not available.

The paper is organized as follows. We define in Section 2 the ESPCPR. In Section 3, we briefly present the Cumulative Scheduling Problem (CuSP) and we show how we can get a relaxation from ESPCPR to CuSP. In Section 4, we present our lower bounds and we conclude the paper in Section 5.

2 Problem description

An instance $I = (X, U, a, v)$ of ESPCPR is defined by a set $X = A \cup \{0, n+1\}$ of events where $A = \{1, 2, ..., n\}$ is the set of real events and 0 (resp. $n+1$) is the fictitious beginning (resp. termination) event of the project, U is the set of precedence relations on the set X of events, a is the vector of resource production and consumption of events, and v is the matrix of *time lags*. The number of resource units produced or consumed by event i is defined by a_i, where a_0 corresponds to the initial resource units of the project. If $a_i < 0$, then event i consumes $|a_i|$ resource units, whereas if $a_i > 0$, it produces a_i resource units. A *schedule S* is a function assigning an occurrence time t_i to each event $i \in X$. The time lag from event i to event j is defined by v_{ij}. If $v_{ij} \geq 0$, then event j cannot occur before time $t_i + |v_{ij}|$,

Abderrahim Sahli · Jacques Carlier · Aziz Moukrim
Sorbonne universités, Université de Technologie de Compiègne, CNRS, laboratoire Heudiasyc UMR 7253, CS 60 319, 57 avenue de Landshut 60 203 Compiègne cedex, France
E-mail: abderrahim.sahli, jacques.carlier, aziz.moukrim@hds.utc.fr

where t_i is the occurrence time of event i. If $v_{ij} < 0$, this implies that event i has to occur no later than time $t_j + |v_{ij}|$. At any time the resource availability has to remain positive or null. The makespan of a schedule S can be computed as $C_{max} = t_{n+1}$. A schedule is feasible if it satisfies all precedence and resource constraints. An optimal schedule is a feasible schedule which minimizes the makespan. In the case of multiple resources, K is the set of resources and a_i^k defines the quantity of resource k produced or consumed by event i, where $k \in K$.

3 Cumulative Scheduling Problem (CuSP)

In the Cumulative Scheduling Problem, a set I of n activities has to be scheduled without preemption in order to minimize the makespan. Each activity i has a processing time or duration d_i, a release date r_i, a tail or latency duration q_i and a resource requirement e_i, the availability of the resource is equal to R. Carlier and Pinson in [3] have adapted the Jackson's Pseudo-Preemptive Schedule (JPPS) to this problem in order to compute a lower bound with $O(n \log n + nm \log m)$-time complexity.

This problem is of prime interest for solving more complex scheduling problems like the ESPCPR. In fact, we can get a relaxation for ESPCPR based on CuSP as follows. We separate all events into two subsets. Let C contain all consumption events and P contain all production events. For each consumption event c, we compute its earliest starting time EST_c and store it as r_c. For each production event p, we compute its latest starting time LST_p and denote $q_p = l_{0,n+1} - LST_p$, which means latency duration or tail where $l_{0,n+1}$ is the length of the longest path from the beginning event 0 to the termination event $n+1$. For a production event p and a consumption event c, if there exists a positive path from c to p, we denote l_{cp} the length of the longest path between c and p. So a bipartite graph G' is established $G' = (C \cup P, U')$ where $U' = \{(c,p) | c \in C, p \in P, l_{cp} \in \mathbb{N}^*\}$, it defines a transportation problem (l_{cp} is the benefit).

A solution of the transportation problem is computed and transformed into a CuSP where each assignment of the resource is regarded as an activity. The resource flow between two events is converted into the resource required by the activity and l_{cp} into the duration of the activity. r_c is the release date and q_p is the tail.

Production and consumption events, which are not included in the solution, can also be transformed into activities by setting a hypothetic makespan C_{max}. Let P' (resp. C') be the set of the remaining production (resp. consumption) events and a'_p (resp. a'_c) the new quantity to produce (resp. consume) by event p of P' (resp. c of C'). For each production event p (resp. consumption event c), it corresponds an activity i with release date $r_i = 0$ (resp. $r_i = LST_c$), a processing time $d_i = EST_p$ (resp. $d_i = C_{max} - LST_c$), a tail $q_i = C_{max} - EST_p$ (resp. $q_i = 0$) and a resource capacity requirement $e_i = a'_p$ (resp. $e_i = -a'_c$). The resource availability of this new instance which we denote CuSP(C_{max}) is equal to $\sum_{p \in P'} a'_p$.

4 Lower bounds for ESPCPR

The first lower bound we present is a destructive bound based on JPPS that we compute as follows. First we fix a hypothetic makespan C_{max} and we extract an instance of the Cumulative Scheduling Problem CuSP(C_{max}) as explained in the previous section. Then we apply JPPS to the corresponding instance to get a lower bound $JPPS(C_{max})$. If $C_{max} < JPPS(C_{max})$ then $C_{max} + 1$ is a lower bound for ESPCPR.

The second lower bound is based on the Shifting Algorithm. The Shifting Algorithm [1] [2] was introduced to solve the Financing Problem in polynomial time ($O(n\log n)$). This problem is a special case of the ESPCPR, where the dates of production events are given. So to compute this bound, first we make a relaxation from ESPCPR to the Financing Problem by setting the production events at their earliest starting times and the consumption events at their latest starting times according to $l_{0,n+1}$. Then, we apply the Shifting Algorithm to the corresponding instance. At last, we take the makespan as a lower bound for ESPCPR.

Another destructive bound can be computed as follows. We fix the value of C_{max} and we set the date of production events at their earliest starting times and the date of consumption events at their latest starting times. If a resource conflict is detected then $C_{max}+1$ is a lower bound [2] [6].

The last lower bound is a destructive bound based on flow that we compute as follows. First, we introduce a bipartite graph $G_I = (P \cup C, U_I)$. The first part of the graph is the set of all production events P and the second part is the set of all consumption events C. We fix a hypothetic makespan C_{max} and we set the date of production events at their earliest starting times and the date of consumption events at their latest times. We consider an arc between a production event p and a consumption event c, if event p can start before event c which is obviously impossible if there exists a strictly positive path from c to p. If the flow problem defined by the graph G_I does not admit any solution then $C_{max}+1$ is a lower bound for ESPCPR.

5 Conclusion

In this paper, we have studied the Event Scheduling Problem with Consumption and Production of Resources (ESPCPR). We have shown how we can get a relaxation from ESPCPR to the Cumulative Scheduling Problem and the Financing Problem. Moreover, we have proposed three new lower bounds for this problem. We have tested them on the benchmark of Neumann and Schwindt [6]. Our lower bounds are very close to the optimal solution makespans. In fact, they reach them for more than 90.29% instances with 1.12% average deviation in percent. As a perspective, we aim to build a branch-and-bound method to solve the ESPCPR using our lower bounds.

Acknowledgements This work was carried out and funded in the framework of the Labex MS2T. It was supported by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02). It was also partially supported by the National Agency for Research, under ATHENA project (Reference ANR-13-BS02-0006).

References

1. Carlier, J., Rinnooy Kan, A.H.G., Financing and Scheduling, Oper. Res. Letters, 1, 52-55(1982)
2. Carlier, J., Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité, Doctorat d'état, Habilitation thesis, Université Pierre et Marie Curie (Paris VI), (1984)
3. Carlier, J., Pinson, E., Jackson's pseudo-preemptive schedule and cumulative scheduling problems, Discrete Applied Mathematics, 145, 80-94(2004)
4. Carlier, J., Moukrim, A., Xu, H., The Project Scheduling Problem with Production and Consumption of Resources: a list-scheduling based algorithm, Discrete Applied Mathematics, 157, 3631-3642(2009)
5. Laborie, P., Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, Artificial Intelligence, 143, 151-188(2003)
6. Neumann, K., Schwindt, C., Project Scheduling with inventory constraints, Mathematical Methods of Operations Research, 56, 513-533(2002)

MISTA 2015

"Floating Patients" method based on scheduling algorithm for emergency department's service improvement

Guy Wachtel • Amir Elalouf

1 Introduction

A hospital's emergency department (ED) is the place where patients receive initial diagnosis and treatment and is responsible for assigning incoming patients to appropriate departments in the hospital, or for referring them to general practitioners or to specialists for further treatment. Overcrowding in EDs is a serious problem in healthcare systems around the world and in Israel in particular. A common solution for handling overcrowding is to determine a maximum level of occupancy for the ED and, once this level is exceeded, to turn away arriving patients and ambulances (by referring them to other EDs in the area) and reject non-urgent patients. Those "turn-aways" and rejections not only inconvenience patients but result in loss of income for hospitals and EDs.

In addition, overcrowding in EDs has a negative influence on the quality of medical care as well as on hospital profits [1]. ED overcrowding and patient length-of-stay (LOS) have therefore been attractive subjects for operations and health care researchers for many years, and numerous approaches have been developed to improve ED work flow. Most studies have focused on forecasting patient volume, scheduling physicians' and nurses' shifts, medical process chains and planning resource utilization. These works have relied largely on three approaches: queuing and scheduling, mathematical and dynamic programming, and economics.

Another current of literature uses algorithmic approaches to derive recommendations for improving the quality of care provided by EDs. Most of the papers in this current address the problem of increasing the accuracy of triage examination. Ballard et al. [2] validated an algorithm for categorizing patients' severity in the New York University ED. Lowe and Fu [3], from another angle, tested the ability of ED algorithms to detect changes in ED use. They found that even if an algorithm can efficiently identify the severity of different patients' conditions and various patient characteristics, it is less useful than other methods in predicting differences in patients' access to care (length of stay). An algorithm-based study by Yeh and Lin [4] departs from the focus on triage examination, proposing a genetic algorithm that seeks

Guy Wachtel
Department of Management, Bar Ilan University, Ramat Gan, Israel
E-mail: Guy.Wachtel@outlook.co.il

Amir Elalouf
Department of Management, Bar Ilan University, Ramat Gan, Israel
E-mail: Amir.Elalouf@biu.ac.il

to improve nurses' shift scheduling for the purpose of enhancing ED care quality. The genetic algorithm approach is useful for planning and determining the allocation of resources in the ED within the constraints of the hospital's budget. Yeh and Lin's [4] results imply that it is possible to improve quality of care merely by adjusting nurses' schedules, even without increasing the number of nurses in the system.

In this paper we assume that the ED determines a maximal (fixed or dynamic) value for patients' length of stay, and that patients who cannot be evaluated in the ED in a timely fashion are redirected for treatment into other hospital departments. The latter approach (referred to as the "floating patient" method) is practiced, for example, in Israel. We suggest a dynamic programming (DP) algorithm and corresponding FPTAS (Fully Polynomial Time Approximation Scheme) to schedule patients' examination and treatment in the ED while taking into account maximal LOS because of the need for fast decision in the ED environment. Our work is based on the assumption that appropriate scheduling and programming approaches can speed up patient handling procedures, thereby reducing the amount of time that patients spend in the ED, improving service quality, increasing patient throughput and, correspondingly, increasing the ED's revenue and profit. We will extend this problem to more closely resembling real life by taking into account stochastic factors, time-flow and uncertainty regarding the patients' actual medical requirements during their stay in the ED. To demonstrate an application of our approach, we carry out simulations using data collected from actual observations in an ED.

2 Problem description

As we address a minimizing version of the JSDP (Job Sequencing with Deadlines Problem) we can define the problem as follows: A set of n independent, non-preemptive, patients has arrived at the ED, $\{J_1,J_2,\ldots,J_n\}$ and is to be evaluated by a physician. We assume that the physician is assigned a group of patients, to whom he provides initial examinations; these examinations provide the physician with information on the duration that is expected for a full evaluation. At any point in time, the physician can decide to stop carrying out initial examinations and to send the other (unexamined) patients to other departments as "floating patients". The physician then schedules full evaluations for the examined patients (he can still decide to send some of them as "floating patients"). After the initial examination, the physician knows/forecasts the patient's evaluation (processing) time (t_i), and the patient's condition uncertainty (p_i). If patient's uncertainty (p_i) is high, the physician will prefer to keep the patient in the ED for full evaluation. The patient's evaluation deadline (D_i) is calculated from the patient's arrival time, r_i (different times $\{r_1,r_2,\ldots,r_n\}$ and the maximal LOS that was decided by the ED management (D), as follows: $D_i=r_i+D$. The problem is to find a permutation $\pi = (\pi_{(1)},\pi_{(2)},\ldots,\pi_{(n)})$ of $\{1,2,\ldots,n\}$, that is, to schedule patients in such an order as to minimize the total uncertainty:

$$P_\pi = \sum_{i=1\ldots n} p_{\pi_i} x_{\pi_i}$$

Where:

$$x_{\pi_i} = \left\lfloor \text{if } t_{\pi_{(1)}} + t_{\pi_{(2)}} + \ldots + t_{\pi_{(i)}} > D_{\pi_{(i)}} \text{ then 1 else 0} \right\rfloor$$

3 Pseudo-polynomial time algorithm

It is well known [5][6], that we can limit our search to schedules such that: (i) patients which are to be on time are evaluated in increasing order of their deadlines, with the tardy patients following them in arbitrary order. In our case tardy patients are sent as "floating patients" to the other departments; and (ii) there is no idle time between patients. Such schedules contain an optimal one. Next, we order all the patients by their deadlines, earliest deadline first and let

$$x_i = \left[\text{if patient } J_i \text{ is to be late 1 else 0}\right]$$

Then the MIN-JSDP can written as the following integer-programming problem:

MIN-JSDP: minimize $P_{(x)} = \sum_{i=1\ldots j} t_i x_i \geq B_j \ , \ j = 1\ldots n$

Subject to $T_{(x)} = \sum_{i=1\ldots j} t_i x_i \geq B_j \ , \ j = 1\ldots n$

$x_i = 0 \text{ or } 1 \ , \ i = 1\ldots n$

Where $B_j = \sum_{i=1\ldots j} t_i - D_j \ , \ j = 1\ldots n$

x stands for $\left(x_1, \ldots, x_n\right)$

As there exists an optimal and feasible schedule in which the jobs are scheduled in EDD order, we say that one feasible schedule dominates another feasible schedule if for any extension of the schedule there exists an extension of it with the same or a better profit and the same or a shorter evaluation due time. Obviously, in this case this schedule can be removed from further consideration without loss of optimality.

4 FPTAS

Our approach for constructing a new FPTAS (Fully Polynomial Time Approximation Scheme) for the MIN-JSDP follows the computational scheme recently developed by [7]. The algorithm consists of three main stages:

Stage A: Find a preliminary lower bound *LB* and an upper bound *UB* on the optimal solution such that $UB/LB \leq n$.

Stage B: Find improved lower and upper bounds on the optimal solution such that $UB/LB \leq 2$.

Stage C: Perform a partition of the interval [*LB, UB*] into $\lceil n/\varepsilon \rceil$ equal sub-intervals, delete sufficiently close solutions in the sub-intervals, and then find an ε-approximation solution.
The complexity of the entire three-stage FPTAS is $O(n^2/\varepsilon)$

The FPTAS will eliminate sub-schedules in each loop of the algorithm, keeping only the n/ε optimal sub-schedules.

5 Concluding remarks

This paper introduces the emergency department's patients scheduling problem with maximal due-time (LOS) constraints. We propose a DP scheduling algorithm with halting an FPTAS, with complexity $O(n^2/\varepsilon)$. When dealing with a large scale network (i.e. network of areal/frontal emergency departments) and\or an on-line scheduling in real-life scenarios, the FPTAS reduces the algorithm's running time in practice. We will propose a simulation method that can be practiced in real-life scenarios.

References

1. Kim, K., Carey, K., Burgess, J. *Emergency department visit: The cost of trauma centers*. Health Care Manage Sci 12, 243–251. 2009.
2. Ballard, D.W., Price, M., Fung, V., Brand, R., Reed, M.E., Fireman, B., New- house, J.P., Selby, J.V., Hsu, J. *Validation of an algorithm for categorizing the severity of hospital emergency department visits*. Medical Care 48, 58–63. 2010.
3. Lowe, R.A., Fu, R. *Can the emergency department algorithm detect430 changes in access to care?* Academic Emergency Medicine 15, 506–516. 2008.
4. Yeh, J.Y., shan Lin, W. *Using simulation technique and genetic algorithm to improve the quality care of a hospital emergency department. Expert* Systems with Applications 32, 1073–1073. 2007.
5. W.E. Smith. *Various optimizers for single-stage production*, Naval Research Logistics Quarterly, 3 (1-2), pp. 59-66. 1956.
6. E.L. Lawler, Moore, J.M. *A functional equation and its application to resource allocation and sequencing problems*, Management Science 16 (1), pp. 77-84. 1969.
7. Tang, Huajun, et al. *Efficient computation of evacuation routes on a three-dimensional geometric network*. Computers & Industrial Engineering 76: 231-242. 2014.

MISTA 2015

Fairness in employee scheduling

Erica Stockwell-Alpert · **Christine Chung**

1 Introduction

In commercial, industrial, and retail settings, it can be tedious and difficult to find a schedule for workers that fills every shift, gives every employee the hours they need, and does not exceed the company's budget. There may be a large number of shifts of varying lengths, all of which require a minimum amount of coverage, as well as a large number of employees who each require a different number of hours of work per week (or scheduling period).

We consider the problem of finding a work schedule that satisfies all employee and shift requirements: where each employee has a minimum number of hours they must work, each shift must be covered by a minimum number of employees, each employee has a set of shifts they are available to work, and there is a limit on the total number of hours available for distribution. We show that deciding whether a feasible schedule exists (one where each employee is working at least their required number of hours) is NP-complete. We then consider the corresponding NP-hard optimization goal of finding the most fair schedule, where the least "satisfied" employee is as satisfied as possible. That is, we wish to minimize the maximum $r_i - h_i$, where h_i is the hours assigned to employee i and r_i is the number of hours employee i is required to work.

In game-theoretic terms, rather than the *utilitarian* objective of maximizing the average satisfaction over the employees, we focus on the *egalitarian* objective of maximizing the satisfaction of the least-satisfied employee, i.e., looking for a solution that is as *fair* as possible.

We are intentionally ambiguous about the source of the parameter r_i, since it can be interpreted as the number of hours the employee specifies they wish to work, or might instead be specified by an employer/manager as a minimum number of hours an employee must work based on work regulations, bookkeeping logistics, etc.

Our problem, which we call the Employee Satisfaction Problem (ESP), has similarities to many employee timetabling problems (ETPs) that have been studied. Much of the work on

Erica Stockwell-Alpert
NorthPoint Digital, Boston, MA
E-mail: estockwell-alpert@northpointdigital.com

Christine Chung
Department of Computer Science, Connecticut College, New London, CT
E-mail: cchung@conncoll.edu

ETPs has been in the artificial intelligence and operations research communities; numerous experimental studies have been conducted using heuristic methods such as local search, branch-and-bound, genetic algorithms, constraint programming and ILP solvers, e.g., see [1–3, 10, 18, 20, 21]. However the variants of timetabling problems in previous works have different constraints and objectives from ours. In particular, they do not focus on fairness to employees. Another point of contrast is that we provide a formal worst-case guarantee on the quality of our algorithm's solution relative to the optimal solution.

Another important distinction between our problem and many other timetabling problems is that our employee requirements are defined in terms of hours, rather than shifts. The difference in the shift lengths is a very real and practical issue, e.g., an employee who is given a series of 3- or 4-hour shifts rather than 8-hour shifts may not actually be working enough hours to support themselves. And indeed, the differing shift durations are the crucial factor in the intractability of ESP.

1.1 Related work

There is a prodigious amount of work done in timetabling, shift assignment, personnel scheduling, and the like. We mention here some of the works that have more significant similarities to ours.

The ESP is similar in nature to the timetable problem studied early on by Gotlieb [16, 15], which Even et al. later show to be NP-complete [13]. They consider the problem of scheduling teachers in a school to class periods, and their problem has many parallels to ours. But while classes may only be taught by one teacher, the work shifts ("classes") in our problem each have a positive integer parameter specifying the minimum number of employees ("teachers") that must be assigned to it. Each teacher in Gotlieb's problem also has a required number of hours that they must teach each class, while our employees simply have a minimum total number of required hours they must work.

Cooper and Kingston [9] demonstrated intractability of timetabling problems in assorted ways, the most similar to ours of which is referred to as "intractibility owing to meeting size," which roughly translates to intractability owing to shift length in our problem. But again the parameters and details of the problems they study have meaningful differences from ours. Their timetabling problem is more complicated, requiring multiple sets to be assigned to the "shifts," with the requirement that certain members of set A be placed on the same shift as certain members of set B, in addition to the basic availability constraints. Aloul et al. proposed a SAT-based approach to solve a variant of employee timetabling [1]. In their formulation, employee requirements are defined in terms of minimum days rather than minimum hours, all shifts are considered equal, and they seek to minimize the number of idle workers.

A wide variety of techniques have been used to solve timetabling problems. To name a few, Aloul et al. [1, 2] examined Boolean satisfiability and ILP-solvers; Boyer et al. [4] used a branch-and-price algorithm to ensure that employees are only assigned to tasks they are capable of; Elahipanah et al. (2013) use a branch-and-bound search tree [1, 2, 4, 12]. Robinson et al. [22] studied a personnel scheduling problem where a set of tasks must be assigned to a set of employees during specific task intervals with the objective of minimizing labor costs. They, along with a later work [5], propose a network flow solution, but in this setting the employees have already been assigned their shifts, and the flow network is for assigning tasks within that schedule. In our work, we also use a network flow-based algorithm, but our

algorithm is used to assign employees to shifts, the problem setting is quite different, and we provide a formal gaurantee on how closely our algorithm approximates an optimal solution.

The scheduling problem most similar to our our own is probably the nurse rostering problem, as it is concerned with fairness to employees [6–8, 17, 19]. Approaches to solving the nurse rostering problem include tabu heuristic search [6]; variable depth search [7]; and heuristic ordering hybridized with a variable neighborhood search [8]. The nurse rostering problem has similar constraints as ESP: there is a set of shifts available for each day (typically "day," "night," and "late night"); an employee has a set availability, and cannot be scheduled when they are unavailable; an employee can only be given one shift per day (this restriction is not necessary for ESP, but we address how to handle this restriction) [19]. The key difference between ESP and the nurse rostering problem is that in ESP, shifts may vary in length, so employee satisfaction is determined based on total hours rather than total number of shifts assigned, and one set of shifts may satisfy an employee while an equal number of shorter shifts may not.

1.2 Contributions

We show that the decision version of ESP is NP-complete. We then present an algorithm that solves a special case of ESP where shifts are of the same length. We further show that for instances that admit a feasible employee schedule (one where $\min_i h_i/r_i \geq 1$), the same algorithm gives approximation guarantees for two variants of our problem: (1) when the "required hours" r_i for each employee i are interpreted to be a minimum number of hours that the employee must work, and additional hours above r_i add to employee satisfaction, and (2) when the "required hours" r_i for each employee i are in fact interpreted to be the *desired hours* of the employee, and hence any additional hours do not add to employee satisfaction, so we cap $\text{OPT} = \min_i h_i/r_i$ at 1.

In the first case, we learn that the further the budget of total available hours k surpasses the total required hours of all employees R, the better the fairness guarantee. And in both cases, keeping all shifts similar in length also improves the fairness guarantee. In particular, in the second case, our algorithm approximates the value of the optimal solution to within an additive

$$\delta \leq \frac{t_{max} - t_{min}}{t_{max}}$$

where t_{min} and t_{max} are the lengths of the shortest and longest shifts, respectively.

However, these results are stipulated by the fact that we allow for a (reasonably bounded amount of) budget overflow. Note that in real world settings t_{min} and t_{max} may not be dramatically different, and the closer they are, the better the approximation guarantee and the lower the budget overflow. Furthermore, in any instance where $k/R \geq t_{max}/t_{min}$, our algorithm is guaranteed to satisfy all employee requirements, i.e., for every employee e_i, $h_i \geq r_i$.

2 Model and preliminaries

The Employee Satisfaction Problem (ESP) can be formalized as follows. We are given as input:

1. A total number of hours available for distribution, k
2. A set of shifts $S = \{s_1, s_2, \ldots, s_n\}$, where for each shift $j = 1...n$, we have:
   - a positive integer t_j, the length of shift j

   - a positive integer m_j, the minimum number of employees needed to cover shift s_j
3. A set of employees $E = \{e_1, e_2, \ldots, e_m\}$, where for each employee $i = 1...m$, we have:
   - $S_i \subseteq S$, the subset of shifts that employee i is available to work
   - a positive integer r_j, the minimum number of hours that employee e_i must be scheduled to work

We make the following basic assumptions on the input, without which the instance would be trivially infeasible.

1. $k \geq \sum_{s_j \in S} m_j t_j$, i.e., there are at least as many hours in the budget as the shifts require
2. $k \geq \sum_{e_i \in E} r_i$, i.e., there are at least as many hours in the budget as the employees require

For convenience and without loss of generality, we also make the assumption that $r_i \geq t_{min}$ for $i = 1...m$. (If $r_i < t_{min}$ for an employee e_i, we can round r_i up to t_{min} because, since $r_i > 0$, any employee must work at least 1 shift, and it is not possible to assign any employee less than 1 t_{min}-hour shift.)

A *solution* to the problem (or *schedule* or *assignment*) is a mapping $\sigma : E \to 2^S$ of employees to sets of shifts they are scheduled to work such that the following constraints are met.

1. for any employee e_i, $\sigma(e_i) \subseteq S_i$, [employee availability constraints]
2. for any shift s_j, $|\{e_i : s_j \in \sigma(e_i)\}| \geq m_j$, and [shift requirements]
3. $\sum_{e_i \in E} \sum_{s_j \in \sigma(e_i)} t_j \leq k$ [budget constraint]

We note that we have not yet addressed the issue of overlapping shifts. As currently stated, the problem allows the same employee to be assigned to two shifts that overlap in time. Indeed, the input as specified above does not even include information about which shifts overlap. For the sake of simplifying presentation, we defer our solution to this issue to Section 5.

The total number of hours assigned by schedule σ for employee e_i is denoted

$$h_i = \sum_{s_j \in \sigma(e_i)} t_j$$

(so the third constraint above can be rewritten $\sum_{e_i \in E} h_i \leq k$).

The decision problem ESP-D is to decide whether a schedule exists where all employees work at least their required number of hours, i.e.,

$$\max_{e_i \in E} r_i - h_i \leq 0,$$

or, alternatively,

$$\min_{e_i \in E} h_i / r_i \geq 1.$$

We refer to such schedules as *feasible*.

We highlight both formulations of the objective here because $\max_i r_i - h_i$ may in fact be 0 or negative. Hence, rather than a standard multiplicative approximation to the corresponding optimization problem, we use the second formulation, and give an additive approximation.[1]

Thus, our corresponding optimization objective for ESP will be to assign shifts to employees so as to

[1] A multiplicative approximation of the second objective would be awkward and perhaps misleading since this objective is effectively formulated as a percentage.

$$\text{maximize} \quad \min_{e_i \in E} h_i/r_i.$$

As previously mentioned, we also allow for two interpretations of the input parameters r_i: (1) r_i represents the minimum required hours an employee must work, and satisfaction level h_i/r_i may exceed 1, or (2) r_i represents the maximum number of hours the employee *wishes* to work, and hence satisfaction level h_i/r_i is capped at 1. Formally, the second interpretation yields the following objective, and we refer to this variant of the problem as ESPW.

$$\text{maximize} \quad \min\left\{\left(\min_{e_i \in E} h_i/r_i\right), 1\right\}.$$

We provide results for both ESP and ESPW. For the remainder of this paper, we denote $T = \sum_{j=1}^{n} t_j$, $R = \sum_{i=1}^{m} r_i$, $t_{min} = \min_j t_j$, $t_{max} = \max_j t_j$, $r_{min} = \min_i r_i$, and $r_{max} = \max_i r_i$. We use OPT or σ^* to denote the optimal solution, and h_i^* will refer to the total hours assigned to employee i in σ^*. We use $|\sigma|$ to denote the objective function value of the solution σ. We sometimes abuse notation and use an algorithm's name to also refer to the solution it returns. An algorithm A is an *additive δ-approximation* for ESP if $|A| \geq |OPT| - \delta$ for all possible instances of ESP.

2.1 Intractability

We show that the decision version of the ESP problem is NP-complete by reduction from PARTITION.

Theorem 1 ESP-D *is NP-complete*

Proof Given as assignment of employees to shifts it can easily be determined in polynomial time whether $h_i/r_i \geq 1$ for all $e_i \in E$. It remains to show that ESP-D is NP-hard. Recall that the problem PARTITION is defined as follows. Decide whether a set of integers $\{x_1, x_2, \ldots, x_n\}$ can be partitioned into two subsets of equal sum. Given an instance of partition, reduce it to ESP as follows. For each integer x_j in the set of integers, we create a shift of duration t_j, with $m_j = 1$. There are two employees with $r_1 = r_2 = k/2$. Both employees are available to work every shift, and we set $k = T$, the sum of the shift durations. Note that if a feasible schedule of shift assignments exists, then all of the shifts have been assigned exactly once, and so the set of integers has been divided into two subsets of equal sum. If a feasible assignment does not exist, there must be no way to partition the integers. □

3 The algorithm

In this section we look at a special case of ESP-D to provide context and build intuition for our proposed algorithm. Specifically, we demonstrate that if all shifts are the same length, the problem can be solved efficiently. In this case, the problem can be solved in polynomial time by reducing to the circulation problem, which can be reduced to the classic max-flow problem [14].

An instance of the circulation problem is comprised of a flow network $G = (V, \mathcal{E})$, a flow demand value d_v for each node $v \in V$, and a capacity specification $[\ell_e, c_e]$ for each edge $e \in \mathcal{E}$, where ℓ_e is the minimum amount of flow required on edge e and c_e is the maximum

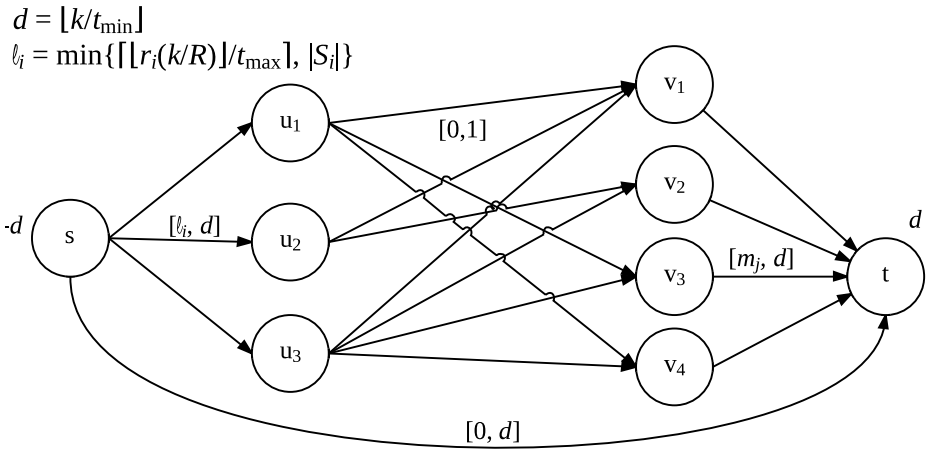


Figure 1 An example of the graph G with 3 employees and 4 shifts. Note that the special case of $t_{min} = t_{max}$ is under consideration in this section.

capacity of edge e. Flow must pass along the edges such that the demands specified at each node are satisfied, and capacity constraints on the edges are observed. Demand on the node v is satisfied if (flow into v) $-$ (flow out of v) $= d_v$. A *feasible circulation* is a flow where all the edge capacity bounds are observed and the demands on each node are satisfied.

Our network (see Figure 1) has an underlying bipartite graph structure with "employee nodes" on one side and "shift nodes" on the other. A unit of flow from an employee node u_i to a shift node v_j means the employee e_i is assigned to shift s_j.

We now present the algorithm that we will be analyzing for the general case where shifts may be differing lengths, keeping in mind that in this section, we assume all shift lengths are equal, hence $t_{min} = t_{max}$. We let $d = \lfloor k/t_{min} \rfloor$, as this is the maximum number of shifts that can be assigned without exceeding k when $t_{min} = t_{max}$. For each employee e_i, the minimum number of shifts that could satisfy their requirement r_i is $\lceil r_i/t_{max} \rceil$. Therefore, we use this as the basis for the lowerbound on the edge incident to the employee node. Finally, we scale each lowerbound by k/R to ensure that any excess hours will be distributed. Formally, the graph G is constructed as stated in Algorithm 1.

We round down $r_i k/R$ in the lowerbound expression to make sure that we do not overestimate our demand and preclude a feasible solution. Finally, we define the algorithm CIRC-D here as Algorithm 2.

The circulation problem can be efficiently solved by reducing it to the max-flow problem and using, for example, the classic Edmonds-Karp algorithm [11] which has a runtime of $O(|V|^2|\mathcal{E}|)$, or, in the context of our problem, $O((n+m)^2(nm))$. (Better run-times can of course be gained by using any of the series of successive improvements to the run time of solving this classic problem.)

The proof of the following theorem may be found in the full version of this paper.

Theorem 2 *If $t_{max} = t_{min}$, the algorithm* CIRC-D *correctly solves the problem* ESP-D.

Algorithm 1: Reducing ESP to the Circulation Problem

Data: a set of employees E, a set of shifts S, the shift availability S_i of each employee i, and a total number of hours k
Result: A circulation flow network G

Add a "source" node s and a "sink" node t;
for *each shift* s_j *in* S **do**
  Add a node v_j to the "right" side of G;
  Add an edge (v_j, t) with capacity bounds $[m_j, d]$;
end
for *each employee* e_i *in* E **do**
  Add a node u_i to the "left" side of G;
  Add an edge (s, u_i) with capacity bounds [min $\{\lceil \lfloor r_i k/R \rfloor / t_{max} \rceil, |S_i|\}, d$] ;
  for *each shift* $s_j \in S_i$ **do**
    Add an edge (u_i, v_j) with capacity bounds [0,1];
  end
end
Add an edge (s,t) with capacity [0, d];
Give s a demand value of $-d$ and t a demand value of d;
Give all other nodes a demand value of 0;

Algorithm 2: Circ-D

Data: ESP inputs
Result: Whether or not there is a feasible employee schedule

Follow the procedure in Algorithm 1 to construct the network G using the relevant inputs to the ESP-D instance;
Run a circulation solver on the graph G;
if *there is a feasible circulation* **then**
  output YES;
else
  output NO;
end

4 Additive approximation guarantee

We use Algorithm 2 to approximate an optimal solution to the general ESP (where t_{min} and t_{max} are not necessarily equal), save for the following modifications, and we refer to the resulting algorithm as CIRC:

- In step 2 we return the circulation itself
- In step 3 we construct the assignment of employees to shifts by adding a shift s_j to the set $\sigma(e_i)$ for each edge (u_i, v_j) that has one unit of flow in the circulation. We then return the assignment σ.

The circulation is the same as in Section 3; however, in the general case that $t_{min} < t_{max}$, the lowerbound produced by $\lceil \lfloor r_i k/R \rfloor / t_{max} \rceil$ on each edge (s, u_i), $i = 1...m$, may be fewer than the minimum number of shifts that could satisfy the requirement r_i, and thus no longer guarantees that $h_i \geq r_i$. The budget restrictions are also effectively relaxed with $d = \lfloor k/t_{min} \rfloor$, which is now a potentially loose upperbound on the number of shifts the budget can afford, and can allow more than k hours of shifts to be assigned. These effectively loosened restrictions ensure that, if no feasible circulation is found, then no feasible assignment of employees exists.

The proof of the following lemma may be found in the full version of this paper.

Lemma 1 *If a feasible solution exists for an instance of ESP-D, a feasible circulation can be found (by* CIRC*) in G.*

Of course, the algorithm may return a feasible circulation when there is no feasible assignment of employees to shifts, and it may indeed return an assignment that exceeds the budget of k, which we will also provide worst-case bounds on. But Lemma 1 ensures that if the algorithm does not return a solution, no feasible solution exists, which perhaps indicates to the employer that the input values are unreasonable and must be reconsidered.

We note that the flow demand value $d = \lfloor k/t_{min} \rfloor$ is the minimum possible that still ensures there will be a feasible circulation when there is a feasible assignment. Indeed, if $d < \lfloor k/t_{min} \rfloor$, there are instances with a feasible assignment where the circulation is infeasible.

As an interesting sidenote, we now show that if $\lfloor k/R \rfloor \geq t_{max}/t_{min}$, all employee requirements are guaranteed to be satisfied by CIRC.

Proposition 1 *If $\lfloor k/R \rfloor \geq t_{max}/t_{min}$, then the solution returned by* CIRC *ensures that $h_i \geq r_i$ for all $i = 1...m$.*

Proof Due to the lowerbounds on edges out of s, each employee e_i is guaranteed to be assigned at least $\lceil \lfloor r_i k/R \rfloor / t_{max} \rceil \geq \lceil \lfloor k/R \rfloor r_i / t_{max} \rceil$ shifts. Thus, we have $h_i \geq \lceil \lfloor k/R \rfloor r_i / t_{max} \rceil \cdot t_{min}$. And with $\lfloor k/R \rfloor \geq t_{max}/t_{min}$, then $h_i \geq \lceil r_i t_{min} \rceil / t_{min} \geq r_i$. □

This simple fact may imply a practical rule of thumb for employers: they should have a budget of at least $k >= (t_{max}/t_{min})R$ total hours for distribution if they wish to guarantee that employees can all work the number of hours they are required to.

Before proceeding with proving our guarantee on minimum employee satisfaction, we first show that the budget overflow of CIRC can be reasonably small when (1) shift lengths are all close in size (i.e., $t_{max} - t_{min}$ is small), or (2) there is a large number of t_{min}-length shifts in S.

Let n_{min} be the number of shifts of length t_{min}. The proof of the following lemma may be found in the full version of this work.

Lemma 2 *In the solution returned by* CIRC*, the budget k will not be exceeded by more than* $b = t_{max} \cdot (\lfloor k/t_{min} \rfloor - n_{min}) + t_{min} \cdot n_{min} - k$

$$\approx k(t_{max}/t_{min} - 1) - n_{min}(t_{max} - t_{min}).$$

As a possible rule of thumb for employers: the budget overflow is lower when the number and duration of maximum-length shifts is lower.

We now move onto the approximation guarantee for minimum employee satisfaction. Let $|\text{CIRC}|$ denote the objective function value of our algorithm's solution. We start by giving a lowerbound on the quality of our algorithm's solution (the proof of which may be found in the full version of this work).

Lemma 3 *For any instance of ESP,*

$$|\text{CIRC}| \geq \frac{\lceil r_{max} \lfloor k/R \rfloor / t_{max} \rceil \cdot t_{min}}{r_{max}}$$

Theorem 3 *Assuming there is a feasible solution to* ESP-D*, and allowing for a budget overflow of b, the algorithm* CIRC *provides an additive δ-approximation to* ESP*, where*

$$\delta \leq \frac{T}{r_{max}} - \frac{\lfloor k/R \rfloor t_{min}}{t_{max}}$$

Proof In any instance, the most hours that any employee can have is T: in the case where they are assigned to every existing shift. Therefore, for any employee e_i we have $h_i/r_i \leq T/r_i$; and hence the minimum satisfaction over all employees in OPT is at most T/r_{max}.

Combining this with Lemma 3:

$$\begin{aligned} |\text{OPT}| - |\text{CIRC}| &\leq \frac{T}{r_{max}} - \frac{\lceil r_{max}\lfloor k/R \rfloor / t_{max} \rceil \cdot t_{min}}{r_{max}} \\ &\leq \frac{T}{r_{max}} - \frac{\lfloor k/R \rfloor \cdot t_{min}}{t_{max}} \end{aligned}$$

□

We defer to the full version of this work the proof of the following theorem, which shows that the above guarantee on the performance of CIRC for ESP (Theorem 3) is essentially tight.

Theorem 4 *There is an instance of ESP where*

$$|\text{OPT}| - |\text{CIRC}| = \delta \geq \frac{T}{r_{max}} - \frac{\lceil r_{max}\lfloor k/R \rfloor / t_{max} \rceil t_{min}}{r_{max}}$$

In the case of the problem ESPW (where satisfaction levels h_i/r_i are always capped at 1, which would be the case when the r_i inputs represent employees' maximum desired hours rather than minimum required hours), we immediately arrive at the following simple characterization of the guarantee of CIRC.

Theorem 5 *Assuming there is a feasible solution to ESP-D, and allowing for a budget overflow of b, the algorithm* CIRC *provides an additive δ-approximation to* ESPW*, where*

$$|\text{OPT}| - |\text{CIRC}| = \delta \leq \frac{t_{max} - t_{min}}{t_{max}}$$

Proof By Lemma 3, and since $\lfloor k/R \rfloor \geq 1$, we have

$$|\text{CIRC}| \geq \frac{\lfloor k/R \rfloor t_{min}}{t_{max}} \geq t_{min}/t_{max}.$$

By definition of ESPW we know that $|OPT| \leq 1$, hence $|\text{OPT}| - |\text{CIRC}| \leq 1 - t_{min}/t_{max}$. □

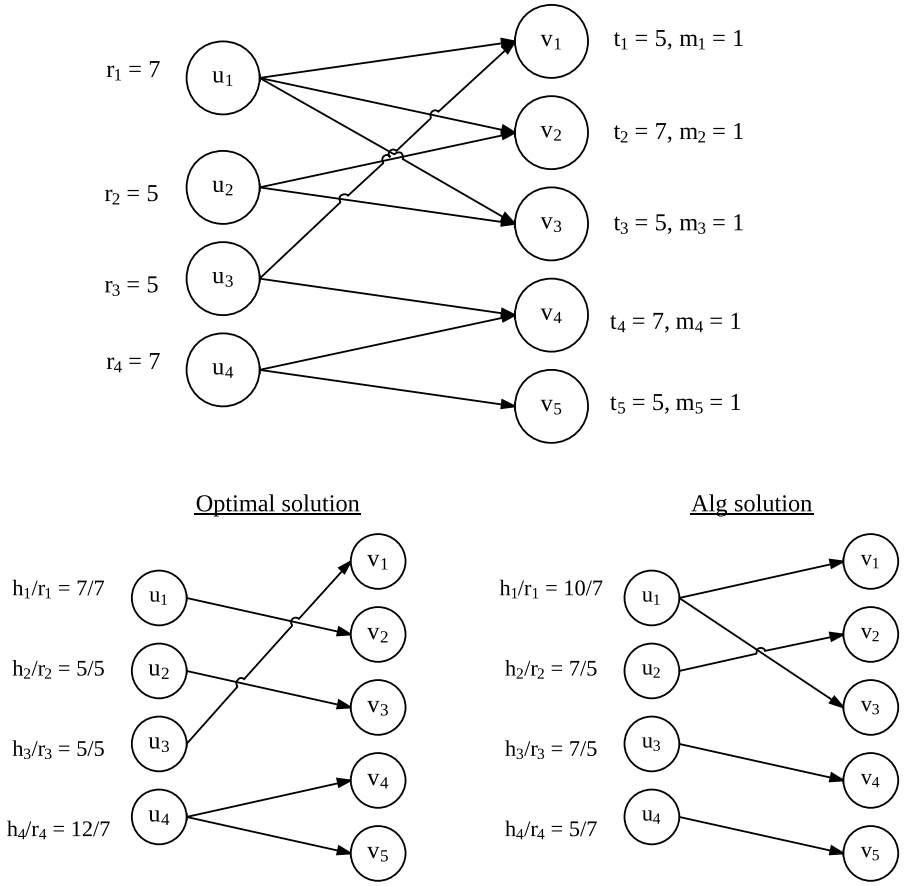
We now demonstrate that Theorem 5 is tight.

Theorem 6 *There is an instance of* ESPW *where*

$$\delta \geq \frac{t_{max} - t_{min}}{t_{max}}$$

Proof The lowerbound on our algorithm for ESPW is demonstrated by a worst-case instance described as follows (also see Figure 2). The instance has $m = 4$ employees, $n = m + 1 = 5$ shifts, and $k = T = 29$. Employee requirements and availability are: $r_1 = 7, S_1 = \{s_1, s_2, s_3\}$; $r_2 = 5, S_2 = \{s_2, s_3\}$; $r_3 = 5, S_3 = \{s_1, s_4\}$; $r_4 = 7, S_4 = \{s_4, s_5\}$.

Shift lengths and are alternating: $t_1 = 5$, $t_2 = 7$, $t_3 = 5$, $t_4 = 7$, and $t_5 = 5$, and shift requirements are $m_j = 1$ for $j = 1 \ldots 5$. The lowerbound for each employee edge (s, u_i), $i = 1 \ldots m$, is hence 1. $d = \lfloor k/t_{min} \rfloor = 5$, which means there are 5 shifts available for distribution among the employees. In the optimal solution, the shifts are assigned as illustrated in Figure 2 for an objective function value of $|OPT| = 1$. However, another feasible circulation exists (as illustrated) that does not satisfy all employees' required hours, giving a minimum satisfaction of t_{min}/t_{max}. Hence $|OPT| - |\text{CIRC}| = \delta \geq 1 - t_{min}/t_{max}$. □



The edge capacities into each node u_i (not pictured here) are [1,d]; d = 5.

Figure 2 An instance that demonstrates the lowerbound of δ for ESPW.

5 Overlapping shifts

Our algorithm as presented thus far assumes that shifts do not overlap. Any two shifts assigned to an employee must not share any hours or else the assignment is invalidated. In order to resolve this and ensure that overlapping shifts are not assigned to the same employee, the following change can be applied to the circulation design.

For each employee e_i, for every pair of overlapping shifts in $\{s_x, s_y\} \in S_i$:

1. Remove the two edges (u_i, v_x) and (u_i, v_y).
2. Add a "median" node w_i "between" the corresponding shift nodes v_x and v_y as follows.
   Add an edge from u_i to the median node w_i.
   Add edges from the median node w_i to each of the two shift nodes v_x and v_y.
   Set all edges to and from w_i to have capacity [0,1].
3. If either of the shift nodes v_x or v_y now has two or more of these median nodes adjacent to it (emanating from u_i), further modify the graph as follows. For each such shift node v_j, $j \in \{x, y\}$, with adjacent median nodes $(w_{i_1} \ldots w_{i_\mu})$:

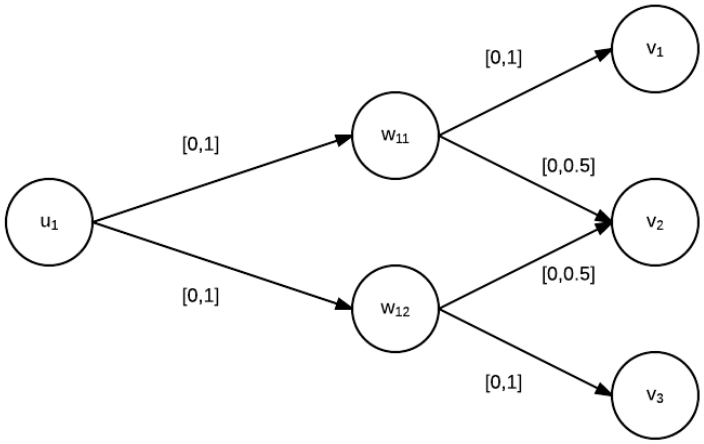


Figure 3 Shifts s_1 and s_2 overlap, and shifts s_2 and s_3 overlap, but shifts s_1 and s_3 are not in conflict

For each edge (w_{i_k}, v_j), $k = 1 \ldots \mu$:
Set its capacity to $[0, 1/\mu]$

An example output of this adjusted procedure is illustrated in Figure 3. In this example, shifts s_1 and s_2 overlap, and shifts s_2 and s_3 overlap, but shifts s_1 and s_3 are not in conflict.

The algorithm must further be modified to prefer whole flows to fractional ones in its tie-breaking; an available edge with capacity of 1 should be preferred over an available edge with a fractional capacity. For example, in Figure 3, there are many different flows that will saturate both edges leaving u_1, but the one that sends whole units of flow over the edges (w_{11}, v_1) and (w_{12}, v_3) is preferred.

The assignments of employees to shifts is determined as before: an employee i is assigned to a shift j if and only if there is one unit of flow from node u_i to node v_j. In particular, if there is less than 1 unit of flow from an employee node to a shift node, that employee is not assigned to that shift.

With this additional procedure, the guarantees of the algorithm remain the same, but now it is certain that no employee will be scheduled for overlapping shifts.

6 Conclusion

Our work shifts the focus of employee timetabling problems onto employee satisfaction. We present an approximation algorithm for the egalitarian objective of maximizing minimum employee satisfaction. ESP can be applied to many types of work environments where varying weekly schedules are used. It addresses the concerns of both the management and the employees: while we allow some budget overflow, we provide a bound for the overflow amount, which the employer can make use of in setting their initial budget (k) value; employees are guaranteed a lowerbound on how many hours they will be given relative to what they need, which promises that the schedule will be relatively fair; and all shift requirements are satisfied, ensuring that every shift will have adequate coverage.

The quality of the guarantees are dependent on the input, specifically on the size difference between the shortest and longest shift in the set, the maximum employee requirement, and the size of k.

Some important future directions include: (1) finding an algorithm with a better approximation guarantee, (2) considering the more complex problem of allowing both a minimum and maximum amount hours to be specified for each employee, and (3) considering the employee's r_i values to be private information that must be extracted from the employee truthfully (making it a mechanism design problem).

References

1. Aloul, F., Al-Rawi, B., Al-Farra, A., & Al-Roh, B. "Solving the employee timetabling problem using boolean satisfiability." in *2006 Innovations in Information Technology*, November 19-21, 2006, Dubai, 4085403 (2006).
2. Aloul, F., Zahidi, S., Al-Farra, A., & Al-Roh, B. "Solving the employee timetabling problem using advanced SAT & ILP techniques." *Journal of Computers*, Vol. 8(4), pp. 851-858, 2013.
3. Artigues, C., Gendreau, M., Rousseau, L.M., & Vergnaud, A. "Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound." *Computers and Operations Research*, Vol. 36(8), pp. 2330-2340, 2009.
4. Boyer, V., Gendron, V., & Rousseau, L. "A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem." *Journal of Scheduling*, pp. 1-13, 2013.
5. Brucker, P., & Qu, R. "Network flow models for intraday personnel scheduling problems." *Annals of Operations Research*, pp. 1-8, 2012.
6. Burke, E., Cowling, P., De Causmaecker, P., Vanden Berghe, G. "A Memetic Approach to the Nurse Rostering Problem." *Applied Intelligence*, Vol. 15(3), pp. 199-214, 2001.
7. Burke, E., Curtois, T., Qu, R., & Vanden Berghe, G. "A Time Pre-defined Variable Depth Search for Nurse Rostering." Technical Report, University of Nottingham, 2007.
8. Burke, E., Curtois, T., De Causmaecker, P., Post, G., Qu, R., Vanden Berghe, G. & Veltman, B. "A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem." *European Journal of Operational Research*, Vol. 188(2), pp. 330-341, 2008.
9. Cooper, T. B., & Kingston, J. H. (1996, October). The Complexity of Timetable Construction Problems. In *Practice and Theory of Automated Timetabling*, Edinburgh, UK, August 1995.
10. Dowsland, K. "Nurse scheduling with tabu search and strategic oscillation." *European Journal of Operational Research*, Vol. 106, pp. 393-407, 1998.
11. Edmonds, J., & Karp, M. "Theoretical improvements in algorithmic efficiency for network flow problems." *Journal of the Association for Computing Machinery*, pp. 248-264, 1972.
12. Elahipanah, M., Dulniers, G., & Lacasse-Guay, E. "A two-phase mathematical- programming heuristic for flexible assignment of activities and tasks to work shifts." *Journal of Scheduling*, pp. 1-18, 2013.
13. Even, S., Itai, A., & Shamir, A. "On the complexity of timetable and multicommodity flow problems." *SIAM J. Comput.*, Vol. 5(4), pp. 691-703, 1976.
14. Ford, L. R., and Delbert Ray Fulkerson. Flows in networks. Vol. 1962. Princeton University Press: Princeton, 1962.
15. Gotlieb, C. C. (1963, January). The construction of class-teacher time-tables. In *IFIP congress* (Vol. 62, pp. 73-77).
16. Gotlieb, C. C. (1962, January). The construction of class-teacher time-tables. In COMMUNICATIONS OF THE ACM (Vol. 5, No. 6, pp. 312-313).
17. Holmes, H., Pierskalla, W., & Rath, G. "Nurse Scheduling Using Mathematical Programming." *Operations Research*, Vol. 24(5), 1976.
18. Kragelund, L. "Solving a timetabling problem using hybrid genetic algorithms." *Software - Practice and Experience*, Vol. 27, pp. 1121-1134, 1997
19. Maenhout, B. & Vanhoucke, M. "Comparison and hybridization of crossover operators for the nurse scheduling problem." *Annals of Operations Research*, Vol 159, pp.333-353, 2007.
20. Meisels, A., Gudes, E., & Soloterevsky, G. "Combining rules and constraints for employee timetabling." *Int'l Journal of Intelligent Systems*, Vol.12, pp.419-439, 1997.
21. Meisels, A., & Shaerf, A. "Modelling and solving employee timetabling problems." *Annals of Mathematics and Artificial Intelligence*, Vol. 39(1-2), pp. 41-59, 2003.
22. Robinson, R., Sorli, R., Zinder, Y. (2005). Personnel scheduling with time windows and preemptive tasks. *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, Pittsburgh, August 2004.

MISTA 2015

Model for planning of distributed data production

Dzmitry Makatun · Jérôme Lauret · Hana Rudová · Michal Šumbera

1 Introduction

The STAR experiment at the Relativistic Heavy Ion Collider (RHIC) studies a primordial form of matter that existed in the universe shortly after the Big Bang. Collisions of heavy ions occur millions of times per second inside the detector, producing tens of petabytes of raw data each year. All the raw data has to be processed in order to reconstruct physical events which are further analyzed by scientists. This process is called data production. Like any other modern experiment in High Energy and Nuclear Physics (HENP), STAR intends to rely on distributed data processing, making use of several remote computational sites (for some experiments this number can scale up to several hundreds).

When running data intensive applications on distributed computational resources long I/O overheads may be observed as access to remotely stored data is performed. Latency and bandwidth can become the major limiting factors for the overall computation performance and can reduce the CPU time / wall time ratio due to excessive I/O wait. Widely used data management systems in the HENP community (Xrootd, DPM) are focused on providing heterogeneous access to distributed storage and do not consider data pre-placement with respect to available CPUs, job duration or network performance. At the same time job scheduling systems (PBS, Condor) do not reason about transfer overheads when accessing data at distributed storage. For this reason,

Dzmitry Makatun
Faculty of Nuclear Physics and Physical Engineering, Czech Technical University in Prague
E-mail: dzmitry.makatun@fjfi.cvut.cz

Jérôme Lauret
STAR, Brookhaven National Laboratory (BNL), USA
E-mail: jlauret@bnl.gov

Hana Rudová
Faculty of Informatics, Masaryk University, Brno, Czech Republic
E-mail: hanka@fi.muni.cz

Michal Šumbera
Nuclear Physics Institute (NPI), Academy of Sciences (ASCR), Czech Republic
E-mail: sumbera@ujf.cas.cz

an optimization of data transferring and distribution across multiple sites is often done manually, using a custom setup for each particular infrastructure [2].

In previous collaboration between BNL and NPI/ASCR, the problem of efficient data transferring in a Grid environment was addressed [6]. Data transfers between n computational sites and m data locations were considered but job scheduling was not covered by that work. In [4] we proposed a constraint programming planner that schedules computational jobs and data transfers in a distributed environment in order to optimize resource utilization and reduce the overall completion time. Since such global scheduling is computationally demanding it should be divided into several stages in order to improve scheduler performance and scalability. A planning of resource load can be completed in the first stage before scheduling file transfers and jobs. In this work we address the problem of data production planning, answering the question how the data should be transferred given the network structure, bandwidth, storage and CPU slots available. This will allow local schedulers to process jobs and have CPUs busy all the time while not exceeding disk and network capacities.

Optimization of data intensive applications in Grid was studied in [5]. In this work an optimization was achieved by replication of highly used files to more sites while the jobs were executed where their input data is located. However, this is not the case for data production, when each file has to be processed once. Explicit model distributing jobs over a Grid with respect to the network bandwidth was proposed in [3]. The network structure of the Grid was modeled as a tree and all the files were assumed to be of the same size and processing time. In our study we do not limit the network topology to trees, and assume fluctuations of job parameters.

2 Modeling

Due to a data level of parallelism a typical workflow of HENP computation consists of independent jobs using one CPU, one input and one output file. We assume there is a local scheduler running at each site, which picks a new input file to process from the local storage of that site each time when a CPU becomes free. Input data must be transferred from the central storage to each site in such a manner that at the every moment of time there is enough input files at each site to keep all the available CPUs busy while not exceeding the local storage and network throughput. Another task is to transfer the output files back to central storage, cleaning each local storage for the new input.

Let us consider a scheduling time interval ΔT. We assume that at the starting moment all the CPUs in the Grid are busy, and there is some amount of input data already placed at each site. We need to transfer the next portion of data to each site during time interval ΔT in order to avoid draining of the local queue by the end of this interval.

The computational Grid is represented by a directed weighted graph where vertexes $c_i \in C$ are computational nodes and edges $l_j \in L$ are network links. The weight of each link b_j is the amount of data that can be transferred over the link per unit of time (i.e. bandwidth). One of the nodes c_0 is the central storage where all the input files for the further processing are initially placed. All the output files has to be transferred back to c_0 from the computational nodes. We will give two separate problem formulations: for an input and output transfer planning.

In order to formulate a network flow maximization problem [1] for input/output file transferring we have to define a capacitated $\{s,t\}$ network, which is a set of vertexes V including a source s and a sink t; and a set of edges $e \in E$ with their capacities $cap(e)$. A solution that assigns a nonnegative integer number $f(e)$ to each edge $e \in E$ can be found in polynomial time with known algorithms.

In order to transform a given graph of a Grid into a capacitated $\{s,t\}$ network for an input transfer problem we add two dummy vertexes: a source s and a sink t. Next we add dummy edges $d_i \in D$ from each computational node i to the sink, and a dummy edge q_0 from the source s to the central storage c_0. These dummy edges allow us to introduce constraints on the storage capacity of the nodes. The set of vertexes V consists of computational nodes C and dummy vertexes: $V = C \cup \{s,t\}$. The final set of edges consists of real network links L, dummy edges D from computational nodes to the sink and from the source to the central storage q_0: $E = L \cup D \cup \{q_0\}$. Capacity of each edge defines the maximal amount of data that can be transferred over an edge within time interval ΔT:

$$cap(e) = \begin{cases} b_j \cdot \Delta T & \text{if } e = l_j \in L \\ w_i & \text{if } e = d_i \in D \\ k_0 & \text{if } e = q_0 \end{cases} \quad (1)$$

where w_i is the maximal amount of data that can be transferred to the node i without exceeding its storage capacity $Disk_i$ and k_0 is the total size of available input files at c_0. We denote the solution for the input transfer problem as $f^{in}(e)$.

For transfer of output files we use a similar transformation, but swap the source s and the sink t, change the direction of dummy edges and redefine capacities of dummy edges. In this case the capacity $\overline{k}_0$ of the dummy edge $\overline{q}_0$ leading from the central storage c_0 to the sink s is equal to the amount of data which can be transferred to c_0 within time interval ΔT (it is limited by the available space at the central storage). The capacity $\overline{w}_i$ of dummy edges $\overline{d}_i$ leading from the source t to computational nodes c_i is equal to the maximum amount of output data which can be transferred from the node c_i.

$$cap(e) = \begin{cases} b_j \cdot \Delta T & \text{if } e = l_j \in L \\ \overline{w}_i & \text{if } e = \overline{d}_i \in \overline{D} \\ \overline{k}_0 & \text{if } e = \overline{q}_0 \end{cases} \quad (2)$$

We denote the solution for the output transfer problem as $f^{out}(e)$.

Let us consider data production jobs which perform the same type of processing on the same type of files. Duration p_j of a job j processing an input file of size $InSize_j$ at a node i is $p_j = \alpha_i \cdot InSize_j$ where α_i is constant for each node i. The ratio of size of input $InSize_j$ and output $OutSize_j$ files of each job j is considered to be constant for the same type of data processing, i.e., $OutSize_j = \beta \cdot InSize_j$. During the time interval ΔT a node i with $NCPU_i$ of CPUs will process $\frac{1}{\alpha_i} \cdot NCPU_i \cdot \Delta T$ of input data and will produce $\frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T$ of output data. Using constraints on storage space, we can define the maximal amount of input w_i and output $\overline{w}_i$ data which can be transferred to/from a node i:

$$w_i = Disk_i - I_i^{in} - I_i^{out} + \frac{1-\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T + Del_i^{out} \quad (3)$$

$$\overline{w}_i = I_i^{out} + \frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T - Min_i^{out} \quad (4)$$

where $Disk_i$ is available disk space at the node i, I_i^{in} and I_i^{out} are the initial size of input and output data at a local storage respectively, Del_i^{out} is the amount of output data that will be transferred out of the node and deleted from its storage during ΔT; Min_i^{out} is the total size of output files which cannot be transferred because the jobs which produce them are not finished (output files of running jobs).

In the Eqn. 3, Del_i^{out} is equal to the amount of data which will be transferred from a node c_i, i.e., the solution to the output transfer problem $f^{out}(\overline{d}_i)$. In the Eqn. 4, ΔT and Min_i^{out} are parameters of the scheduler. The other values used in Eqns. 3–4 can be obtained from monitoring data right before each planning iteration.

3 Solving Procedure

It can be proven that the maximum flow problems for input and output transfers can be solved independently under assumptions: (a) all the real network links in the considered Grid are full-duplex, i.e., a network throughput between two nodes is the same in both directions (b) in a steady state the size of the output transferred from each node is proportional to the size of the input transferred to that node in each scheduling interval, i.e., $f^{out}(\overline{d}_i) = \beta \cdot f^{in}(d_i)$, where $\beta \leq 1$.

Since in real environment the assumption (b) will not strongly hold due to resource performance fluctuations we propose the following approach to solve the problem:

1. Calculate values for $\overline{w}_i$ using Eqn. 4.
2. Solve the problem for output data flows to obtain $f^{out}(e)$.
3. Using Eqn. 3 and $Del_i^{out} = f^{out}(\overline{d}_i)$ calculate w_i.
4. For real links $l \in L$ reduce the capacity by the amount which is used by output transfers: $cap(l_j) = b_j \cdot \Delta T - f^{out}(l_j)$.
5. Solve the problem for input transfers with w_i and $cap(l_j)$ defined in previous steps. Find input data flows $f^{in}(e)$.

To conclude, this procedure is expected to compute feasible data transfers such that CPUs in Grid are busy with computational jobs while not exceeding local disk capacities. An evaluation of the proposed heuristic with the help of Grid simulations is planned as the next part of the research.

4 Conclusion

In this paper we proposed a model of distributed data production, where all the files from a single source has to be processed once and transferred back. This model allows planning of WAN, storage and CPU loads using the network flow maximization approach. The proposed model will be used in a distributed data production planner which is being developed. The planner will enable automated and scalable planning and optimization of distributed computations which are highly required in data intensive computational fields such as High Energy and Nuclear Physics.

Acknowledgements This work has been supported by the Czech Science Foundation (13-20841S, P202/12/0306), the MEYS grant CZ.1.07/2.3.00/20.0207 of the European Social Fund (ESF) in the Czech Republic: Education for Competitiveness Operational Programme (ECOP) and the Office of Nuclear Physics within the U.S. Department of Energy.

References

1. Ahuja, R.K., Magnati, T.L., Orlin, J.B.: Network flows : theory, algorithms, and applications. Prentice Hall (1993)
2. Balewski, J., Lauret, J., Olson, D., Sakrejda, I., Arkhipkin, D., et al.: Offloading peak processing to virtual farm by STAR experiment at RHIC. J. Phys.: Conf. Ser. **368**(012011) (2012)
3. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Bandwidth-centric allocation of independent tasks on heterogeneous platforms. IEEE International Parallel and Distributed Processing Symposium (2002)
4. Makatun, D., Lauret, J., Rudová, H., Šumbera, M.: Planning for distributed workflows: constraint-based coscheduling of computational jobs and data placement in distributed environments. Journal of Physics: Conference Series **608**(1), 012,028 (2015)
5. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. 11th IEEE International Symposium on High Performance Distributed Computing pp. 352–358 (2002)
6. Zerola, M., Lauret, J., Barták, R., Šumbera, M.: One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs. J. Phys.: Conf. Ser. **368**(012022) (2012)

MISTA 2015

A Decomposition Heuristic for a Bicriteria Evacuation Scheduling Problem

Kaouthar Deghdak · Vincent T'kindt

1 Introduction

Evacuation after a major disaster is a very complicated task which needs an effective scheduling to transport resident from endangered areas to safe destinations. Whenever a disaster take place, many sub-problems arise: when should people be evacuated? What are the best shelters' locations to accommodate evacuees? How to transport the evacuees there?. Evacuation problems have been of a great interest for researchers due to an increasing number of natural and man-made catastrophes, where developing evacuation plans for urban area are necessary. Frequently, several operation research approaches have been used to solve these problems separately or partially. Important reviews about evacuation modeling and disaster management can be found in [1] and [3].

One interesting problem tackled as an OR problem is the shelter location decision and the evacuation routing problem. It has been firstly adressed in [5] and later are followed by others publications mainly introducing heuristics. Recently, Bish [2] has introduced the bus evacuation problem, in which a mathematical model and heuristics are proposed. Bretschneider [7] has introduced the multiple commodity evacuation problem using buses and vehicles. The aim is to define routes and timetables, in such a way a weighted linear combination of the flows of the commodities arriving at their corresponding destinations and the total number of emergency lanes is minimized. Furthermore, an heuristic based on mathematical formulation is proposed to solve this problem. This heuristic is able to solve small instances in a reasonable time.

The purpose of this work is to study the problem of evacuating an urban area after a major disaster. We consider the real-world instances of Nice (France) and Kaiserslauter (Germany). Also, we consider simultaneously the location choice of shelters, bus routing for public transport, and routing for individual traffic. The objective is to minimize both evacuation time and evacuation risk. This problem was introduced in [6] and they solved this problem by using a genetic algorithm. In contrast to the integrated approach developed in [6], we propose an heuristic based on a decomposition of our

Kaouthar Deghdak, Vincent T'kindt
Université François-Rabelais de Tours, CNRS, LI EA 6300, OC ERL CNRS 6305, Tours, France
E-mail: {deghdak,tkindt @univ-tours.fr}

problem into sub-problems. The solution of these sub-problems is achieved by exact or heuristic algorithms.

2 Problem definition

Let $\mathcal{C}$ be the set of collection points: any $c_j \in \mathcal{C}$ is defined by its geographic coordinates and a maximum capacity $c_j^{\mathcal{C}}$. Let $\mathcal{BS}$ be the set of bus stops: any $bs_j \in \mathcal{BS}$ is defined by its geographic coordinates. Let $\mathcal{S}$ be the set of shelters: any $s_j \in \mathcal{S}$ is defined by its geographic coordinates and a maximum capacities $c_j^{\mathcal{S}}$ and $c_j^{\mathcal{P}}$ representing the number of people who can be hosted at that shelter and the parking space available near this shelter, respectively. Let $\mathcal{B}$ be the set of buses used to evacuate people from the bus stops and collections points towards shelters: any $b_j \in \mathcal{B}$ is defined by its initial geographic coordinates (depot) and a capacity in terms of number of people who can be transported at the same time. We make the assumption that all the buses have the same capacity.

Consider a transportation network within the study area represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs between two nodes. A node $n_i \in \mathcal{N}$ can model either a crossroad, a shelter location $s \in \mathcal{S}$, a collection point $c \in \mathcal{C}$, or a bus stop $bs \in \mathcal{BS}$. An arc $a_{i,j} = (n_i, n_j)$ indicates that a flow of buses/cars can go from node n_i towards node n_j. Besides, it is associated with two capacities $c_{i,j}^b$ (the maximum number of buses using the arc at a given time t) and $c_{i,j}^v$ (the maximum number of cars using the arc at a given time t), a traversing time $p_{i,j}$, and a risk value $r_{i,j}$ indicating the likelihood of a building collapse.

The aim is then at computing a set of solutions, which minimize the evacuation duration T_{evac} and the total risk R. A solution is defined by: (I) the list of shelters that will be opened and accomodate evacuees, (II) the routing of cars towards the shelters, and (III) the routing of buses to evacuate all people located at the bus stops and the collection points towards the shelters.

In the field of multicriteria optimization, many methods to compute the Pareto front are known [8]. In this work we use the ε-constraint approach as follows: the total risk R is minimized under the constraint that the maximum evacuation time T_{evac} is lower or equal to a given value ε. From practical point of view, solving one ε-constraint problem makes sense: the ε value represents a threshold which guarantee that the evacuation is performed in no more than ε time units. Then the aim becomes at minimizing the total risk within that time limit. Additionally, while the evacuation time is a very descriptive value, the total risk is a more abstract value, and fixing a desired quality is hardly possible in practice.

3 Decomposition heuristic

While the Genetic algorithm[6] already solves the problem it is not expected to be capable to solve large instances like the case of the city of Nice. Consequently, our goal in this communication is to propose an efficient and a fast heuristic capable of handling such large instances.
We describe below the main steps of our decomposition heuristic.
(**1**) **Bicriteria paths**: This step is fundamental and aims at reducing the size of the

transportation network. For each couple (collection point/bus stop ; shelter), we calculate bictreteria paths, where the two criteria considered are the maximum evacuation time and the total risk. For this, we use the algorithm proposed in [4].
(**2**) **Shelters' locations:** The maximum number of shelters which can be opened during the evacuation is fixed by the decision maker. In this step, we choose the set of the best shelters' locations among all available. In term of optimization models, this problem can be modeled as a multi-dimensional knapsack problem. To this end, we propose a dynamic program to enumerate sets of shelters' locations. For each set, we solve the bus-routing and the car-routing problems. These steps are explained in detail below.
(**3**) **Car-routing:** The choice of routes for cars is performed using a *min-max flow* algorithm on a time expanded network, while only the paths that respect the ϵ-constraint in the T_{evac} criterion are considered.
(**4**) **Bus-routing:** After the evacuation of all cars, we evacuate all people by buses. The bus-routing problem is more elaborated and complicated than the car-routing problem, as a single bus may perform several trips from collection points toward shelters. Initially all buses are located in a depot. Then, the buses are routed to the nearest collection points and bus stops. We apply a *min cost flow* algorithm to transport evacuees from collection points or buses stops to shelters, where the risk R is minimized under the ϵ-constraint. We uses a shortest path algorithm to route empty buses from shelters to collection points or bus stops.
(**5**) **Local search:** Finally, solutions are post-optimized using a local search based on permutation of the paths used by buses and cars. To reduce computation times, the local search is performed until a given number of iterations has been performed, or a local optimum is reached.

4 Results

We have tested the decomposition heuristic and the genetic algorithm on the real instances of Nice city (France). Preliminary results show that the decomposition heuristic complements the Pareto front obtained by the genetic algorithm. In other words, the decomposition heuristic is capable for finding a non dominated solutions that are not calculated by the genetic algorithm. Notice that, apparently, the convergence of the decomposition heuristic towards an approximation of the Pareto front seems to be slower than for the genetic algorithm.

During the conference, we will present detailed comparisons between the two methods in terms of running times and solutions' quality for several instances of Nice city and Kaiserslautern city.

Acknowledgements This research has been supported by ANR-11-SECU-002-01, project DSS_EVAC_LOGISTIQUE (CSOSG 2011).

References

1. N. Altay, W. G. Green III, OR/MS research in disaster operations management, Eur. J. Oper. Res, 175, 475493 (2006)
2. D R. Bish, Planning for a bus-based evacuation, OR Spectrum, 33(3), 629-654(2011)

3. H. Hamacher, S. Heller, W. Klein, G. Kster, S. Ruzika, A sandwich approach for evacuation time bounds. In: Peacock, R.D., Kuligowski, E.D., Averill, J.D. (Eds.), Pedestrian and Evacuation Dynamics,503513. Springer, USA (2011)
4. G. Sauvanet, E. Néron, Search for the best compromise solution on Multiobjective shortest path problem, Electronic Notes in Discrete Mathematics, 36, 615-622 (2010)
5. H.D. Sherali, T.B. Carter, A.G. Hobeika, A location-allocation model and algorithm for evacuation planning under hurricane/flood conditions, Transp. Res. Part B: Methodol, 25, 439-452 (1991)
6. M. Goerigk and K. Deghdak and P. Hessler, A comprehensive evacuation planning model and genetic solution algorithm, Transportation Research Part E: Logistics and Transportation Review, 71, 82 - 97 (2014)
7. S. Bretschneider, Mathematical Models for Evacuation Planning in Urban Areas, Springer (2010)
8. V. T'kindt and J-C. Billaut, Multicriteria scheduling: theory, models and algorithms, Springer-Verlag Berlin Heidelberg, 2nd edition, (2006)

MISTA 2015

Iterated Local Search for the Generator Maintenance Scheduling Problem

Ahmad Almakhlafi · Joshua Knowles

Abstract We consider the task of devising an effective metaheuristic for a variant of the preventive maintenance scheduling problem (PMSP) — the (power) generator maintenance scheduling problem (GMSP). Recent research on metaheuristics for this problem has made progress on it, but the potential economic benefits of effective methods is significant in this area, and warrants further focused work. We propose here a solution method based on *Iterated Local Search* (ILS) following an earlier study by us on neighbourhood search for the same task. Several extensions to a basic ILS design are developed and analysed, including specialised operators and delta-evaluation, as well as restart and portfolio strategies. With these methods, we obtain a significant improvement in performance (in terms of solution quality, runtime and function evaluations) over recent techniques for real-world derived instances of the GMSP. We also provide a benchmark (and results on additional benchmark instances) for future studies of this problem.

Keywords Iterated Local Search, Algorithm Portfolio, Maintenance Scheduling Problem, Generators, Delta Function, Variable Neighbourhood Descent, Hybrid, Evaluation Function, Restart Strategy, Run Length Distribution, Benchmarks.

1 Introduction

Preventive maintenance (PM) is a series of tasks, such as regular inspections or the replacement of aged parts, carried out to extend the life of a machine. PM is done preemptively before the machine has reached a point of critical wear and is about to fail [53], and hence it is a planned activity, dependent on scheduling. Due to the

First Author
School of Computer Science, University of Manchester, Kilburn Building, Oxford Road, Manchester, M13 9PL, United Kingdom
E-mail: almakhla@cs.man.ac.uk

Second Author
School of Computer Science, University of Manchester, Kilburn Building, Oxford Road, Manchester, M13 9PL, United Kingdom
E-mail: j.knowles@manchester.ac.uk

widespread use of PM in production and service industries, and its economic benefits in preventing damage to (often very expensive) equipment or infrastructure, some commentators have suggested that improving PM scheduling is one of the most significant problems faced by industry today [74]. Its economic importance is likely to easily rival more better-known planning problems such as production scheduling [43], nurse rostering [13] and timetabling [60].

One important form of the PMSP is the Generator Maintenance Scheduling Problem (GMSP) in the power generation industry, which concerns the maintenance of expensive infrastructure that can also be critical to the reliability of national power grids. This problem has been studied for a number of years in the OR literature (often using rather small benchmark instances) and several optimization methods, from exact methods to metaheuristics, have been developed and applied (e.g., see [74,24,41, 68]). We here tackle the GMSP using local search as our basis, following some earlier investigative work where we found it a promising approach to the problem [2]. Here we use the framework of Iterated Local Search (ILS), a simple but powerful metaheuristic framework for improving the performance of basic local search, which has been applied successfully to a number of problems including the travelling salesman problem [65], the permutation flowshop [22] and the quadratic assignment problem [64]. We aim here to achieve similarly good results with the GMSP.

To pursue our goal, we use a staged development process. We first analyse the main components of the ILS algorithm using a set of in-house developed GMSP instances for the testing. Then we suggest several extensions of the proposed ILS: an ILS with restart strategy, an ILS with delta evaluation implementation, an ILS hybrid with Variable Neighbourhood Descent (VND) algorithm and a Portfolio of ILSs. These algorithms are developed carefully by looking at run-length distributions and other analysis tools. The performance of the proposed ILS and its variants are also tested on two GMSP instances from the existing literature, allowing us to compare directly with some previously proposed algorithms.

The remainder of the paper is organized as follows: a description of the GMSP and its mathematical formulation is presented in Section 2. Section 3 details the different ILS operators proposed and the function of the run-length distributions used for analysis in this work. In Section 4 we suggest several extensions to the initial proposed ILS algorithm. In section 5, we present an analysis of the run-length behaviour of different configurations of the ILS. The proposed ILS algorithm and its variants are tested and compared with some results from the literature in Section 6. Finally, some concluding remarks are given in Section 7.

2 PMSP in power plants

In this section, we introduce the generator maintenance scheduling problem, give the formulation we will consider in this paper, and briefly survey related work on the development of solution methods.

2.1 Overview

Maintenance plays a crucial role in the power-generating industry's planning and operation. Modern power plants utilize large capacity units making a single outage cause

Table 1: List of parameters used to formulate the GMSP.

i	Index of generating units	I	Total number of units
t	Index of intervals	T	Total number of intervals
C_{it}	Generating capacity of unit i in interval t	C_s	Generating capacity of the system
		M_t	Available manpower at interval t
m_{it}	Manpower needed to maintain unit i in interval t	R_t	Nett reserve for interval t
		d_i	Duration of maintenance for unit i
R_m	Reserve safety margin	P_t	Predicted load for interval t

both a possible large loss of generation capacity and increased cost of corrective maintenance [51]. Hence, the effective scheduling of preventive maintenance activities, where units need to be taken off-line for maintenance, allows the power system to perform its function reliably and achieve considerable savings. The preventive maintenance scheduling problem of power plants is dealt with as a long-term planning problem in which the power and energy resources are utilized and maintained during a time horizon from several weeks to several months or years into the future, discretized into intervals [51,39]. Usually, the maintenance outages are planned on a yearly time horizon with scheduling intervals being set at one week [40,30]. The starting times of these outages have to be determined taking into account organizational objectives, system constraints and consumers' power demands [51,39]. The GMSP is a constrained optimization problem [38,73], and is NP-hard in most forms [53,12,11,6].

2.2 Problem formulation

Consider a collection of $I \in \mathbb{Z}^+$ generating units which can generate a total power of C_s over a scheduling horizon of $T \in \mathbb{Z}^+$ intervals. The scheduling horizon is organized in weeks and remains constant for all units. Each $i = 1, \dots I$ has a generating capacity C_i and must be maintained exactly once for d_i consecutive weeks during the period T. The maintenance tasks are performed by a workforce of size m_{it}, where i indexes the unit under maintenance and $t \in 1, \dots, T$ is the maintenance interval. However, the total size of the workforce maintaining generators at an interval t cannot exceed the maximum manpower available M_t at that interval. The nett reserve of the system R_t at any interval t is the power remaining after subtracting the generation loss due to meeting demand and scheduled maintenance outages for that interval. At any interval t, nett reserve must be greater than or equal to the minimum reserve, which is the total of the predicted load P_t for interval t and any reserve safety margin R_m. The problem parameters are summarized in Table 1.

The objective function aims to level the reserve load over the planning horizon by minimizing the sum of the squares of the reserve loads (SSR). In real-world terms, the SSR value measures the reliability of the power system. The constraints present in this problem consist of the system meeting the minimum reserve, the availability of the maintenance workforce, the maintenance window and the duration for each unit to be offline for maintenance, as well as any exclusion constraints to prevent the simultaneous maintenance of certain combinations of generators. In our formulation, the pseudoboolean solution vector X is two-dimensional, and its element X_{it} represents the status of unit i in interval t as follows:

$$X_{it} = \begin{cases} 1 \text{ if unit } i \text{ is under maintenance during interval } t \\ 0 \text{ otherwise.} \end{cases} \quad (1)$$

The optimization problem, which has the goal to minimize the sum of squares of the reserve (SSR) generation, can be written as

$$\text{Minimize} \quad \left\{ SSR = \sum_{t=1}^{T} \left(C_s - P_t - \sum_{i=1}^{I} C_{it} X_{it} \right)^2 \right\} \quad (2)$$

subject to the minimum reserve constraints

$$\sum_{i=1}^{I} C_{it}(1 - X_{it}) \geq P_t + R_m \quad \text{for all } t = 1, \ldots, T, \quad (3)$$

the manpower constraints

$$\sum_{i=1}^{I} X_{it} m_{it} \leq M_t \quad \text{for all } t = 1, \ldots, T, \quad (4)$$

and the duration constraints

$$\sum_{t=s_i}^{s_i+d_i-1} X_{it} = d_i \quad \text{for all } i = 1, \ldots, I \quad (5)$$

where s_i represents the interval in which the maintenance of unit i starts. In addition, this expression ensures the continuous maintenance requirements of units. In general, a GMSP may include alternative or additional constraints. Although this formulation seems to be pure and simple, it actually reflects the problem settings in the power company described in [2].

2.3 GMSP Optimization Methods

The initial formulation of the GMSP was presented in [16,34,33]. Later, the cost function of the problem formulation was improved by Yamayee and Sidenblad [75]. In literature, GMSP has been optimized by many different techniques and each technique has its own difficulties. They can be classified into two groups: exact optimization methods, and metaheuristic methods.

The exact methods, such as integer programming [41,44], dynamic programming [78, 70] and the branch and bound method [23,24,15,69], have generally been applied to small GMSPs. Although these methods have the capability of finding the optimal solution, they cannot generally be applied to large-scale problems, because the size of the solution space increases exponentially with the increasing number of generator units, increasing the computational time of these algorithms accordingly.

To overcome these disadvantages, metaheuristics have been applied to solve the GMSP. These methods include the application of GAs [73,8,2], Simulated Annealing [56,58] and Tabu Search [14,25]. Although metaheuristics can find (near-) optimal

solutions for large problem sizes better than the exact methods with a reasonable computational time, finding such solutions is not guaranteed.

By exploiting individual advantages of the exact and metaheuristic techniques, they can be combined to form hybridized algorithms with good overall performance. Variants of such algorithms which have been proposed in recent years are reported to be superior to their pure counterparts in terms of solution quality and computational time, especially when optimizing real-world *NP*-hard problems [54]. These approaches are commonly referred to as *hybrid algorithms*.

An emerging optimization method is *algorithm portfolios* [32]. This method uses collections of multiple algorithms, either different copies of the same algorithm or different algorithms running in a parallel or interleaved in time [59], to optimize a problem . Compared to a single algorithm, a portfolio can offer better performance when optimizing many instances or even on a single instance [31,32]. An application of this method to optimize GMSP can be found in [3]. Other approaches to solve the maintenance scheduling problem in the power-generating industry include Knowledge-based [5], fuzzy logic [26] and expert systems [46].

Algorithm 1: Iterated Local Search

```
1 s_0 ← GenerateInitialSolution ;
2 s ← LocalSearch(s_0) ;
3 repeat
4 |   s' ← Perturbation(s, history) ;
5 |   s'' ← LocalSearch(s', history) ;
6 |   s ← AcceptanceCriterion(s, s'', history) ;
7 until termination condition met;
8 return s
```

3 ILS for GMSP

Iterated local search [47,48] is a simple yet powerful framework which can be implemented in any type of local search algorithm to improve its performance. The algorithm walks randomly in the space of the local optimum performing a stochastic greedy search. The search strategy of ILS consists of utilizing the local search algorithm to find the local optimum in the defined neighbourhood and then applying small perturbations on the local optimum to escape the basin of attraction of the current local optimum. The local search is then restarted from the perturbed solution.

Despite its simplicity, ILS has been applied successfully to several combinatorial optimization problems like the travelling salesman problem (TSP) [65,49] and various scheduling problems [17,27,42]. Its performance is comparable with several state-of-the-art metaheuristics, such as Simulated Annealing, Tabu Search, Genetic Algorithm and Ant-Colony Optimization [22]. In order to apply ILS to the GMSP, its building components have to be defined: *initial solution generation*, to generate an initial solution; *local search*, to find an improved solution; *perturbation*, to perturb a solution; and *acceptance criterion*, to decide from which solution to continue the search. The architecture of ILS is given by Algorithm 1 [48].

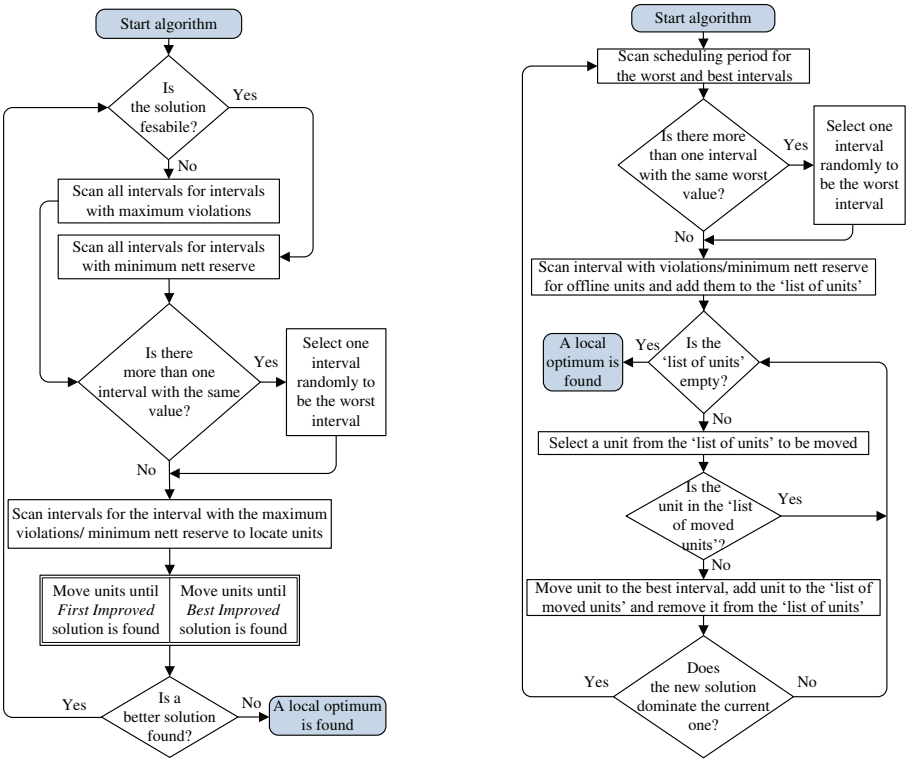


Fig. 1: Flowcharts of the proposed local search operators. Flowchart (a) is similar for BILS and FILS algorithms except for the moving units part. The WBLS algorithm is presented in Flowchart (b).

Making the best possible choice of ILS components is essential to achieve the best overall performance of the algorithm for a particular problem. In the following subsections we provide further details of the components used in this work.

3.1 Initial solution generation

We use a heuristic to generate the initial solution. It is encoded using a two-dimensional binary string representation of $(T \times I)$ bits such that '1' means that the unit is under maintenance (offline), whilst '0' means that it is operational (online). The heuristic assigns each unit exactly one job in the scheduling horizon and determines the position of this job uniformly at random. The schedule is regenerated if a maintenance duration collides with the end of the scheduling period.

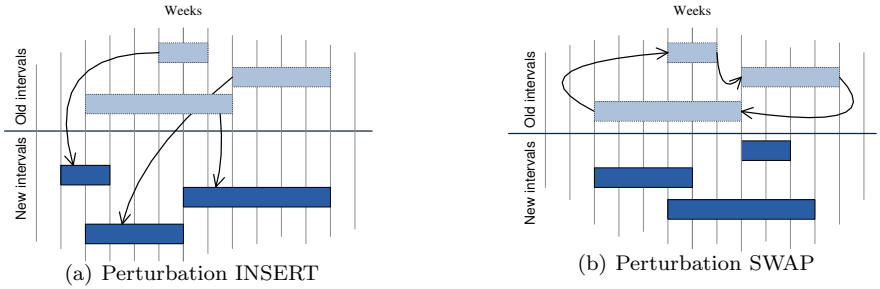
3.2 Local search

A local search algorithm searches for an improved solution by exploring the neighbourhood of a given initial one. If a better solution is found, it replaces the current solution and the search is continued until a local optimum is found. One of the main ingredients of a local search algorithm is defining the neighbourhood structure. A proper neighbourhood structure definition allows for an efficient move from one solution to another. For the local search component of the ILS algorithm we considered three algorithms (FILS, BILS and WBLS, described next) with different neighbourhood definitions. These algorithms were designed to guide the search to the feasible region of the search space and maintain feasibility afterwards. Hence, they would not move to an infeasible solution when the current one was feasible.

First Improvement Local Search (FILS): This algorithm has an advanced version of the intelligent mutation operator described in [2] and is shown in Figure 1(a). The algorithm starts by checking the feasibility of the solution. When the solution is infeasible, it scans all intervals in the scheduling horizon to identify the interval(s) with the maximum violations. If a solution is feasible, it scans all intervals to find one(s) with the minimum nett reserve. In a case where there is more than one interval with the same maximum violations / minimum nett reserve, one interval is selected uniformly at random as the worst interval. The worst interval is then scanned to locate the offline units (under maintenance). Each of these is tried in each of the intervals in sequence, starting from the first week of the scheduling horizon, until an improvement is found and this is moved to. If the end of the scheduling period is reached without finding a fitter solution, the next unit is considered until a better neighbourhood is found. The algorithm continues until no improved solution is found and terminates with the current best when there are no more units to be moved.

Best Improvement Local Search (BILS): BILS has the same mutation operator as that used in FILS, except that it evaluates all possible solutions resulting from moving units from the targeted interval. The best solution among the possible solutions is then chosen. The FILS and BILS algorithms are enhanced versions of the hill climbers (FIHC and BIHC) we presented previously in [2]. The mutation operator in FIHC and BIHC scans all intervals in the scheduling horizon to identify the one with minimum nett reserve, while in FILS and BILS it scans first for the intervals with high violations. When the solution is feasible, it scans for the intervals with minimum reserve. Figure 1(a) illustrates a flowchart of the FILS and BILS algorithms.

Worst Best Local Search (WBLS): WBLS incorporates a tracking heuristic and a memory structure. The heuristic tracks changes in the constraint violations and in the objective function value and locates the worst and best intervals. When a solution is infeasible, the worst interval is the interval with the highest number of violations. If these violations are caused by different constraints, the heuristic checks for which of them contributes more of the violations. The best interval is then determined such that moving the units from the worst interval will reduce the total number of violations for the solution. When the solution becomes feasible, the worst and best intervals are the intervals with minimum and maximum nett reserve respectively. The heuristic makes sure that the search is restricted to the feasible region.



(a) Perturbation INSERT (b) Perturbation SWAP

Fig. 2: Two perturbation operators are illustrated. Figure (a) shows the principle of the INSERT operator where units are relocated randomly in other intervals. The SWAP operator shown in Figure (b) where units exchange their maintenance starting intervals.

The memory structure tracks the units moved during the search. Units which are scheduled to be maintained on the worst interval are added to the 'unitsToBeMoved' list, from where they are selected one by one and moved to the best interval. When a unit is moved, it is added to the 'movedUnits' list to prevent it from moving again. An attempt is also made at this juncture to move a different unit in order to bias the search towards promising areas of the search space. Solutions are compared on the basis of a constraint violation test and objective function value. In cases where there is more than one worst interval, the algorithm utilizes a similar mechanism to that described for the FILS and BILS algorithms. When there are no more units to be moved, the search is considered to have reached equilibrium and the current solution is reported as a local optimum. Figure 1(b) is a flowchart of the WBLS algorithm.

3.3 Perturbation

With a local search algorithm, progress is made by moving only to better solutions and the search terminates in a local optimum. The aim of the perturbation phase is to modify a current local optimum so that it can be effectively escaped and to provide a good starting solution for the local search. The perturbation should be strong enough to allow the local search to escape the local optimum, but also weak enough to maintain some features of the current solution. In this context, we identify two concepts related to perturbation: *perturbation strength* and *perturbation nature* [48].

Perturbation Strength: In this work, perturbation strength refers to the number of units that are moved from their current intervals to some other ones. We examine two types of perturbation strengths, deterministic and adaptive. The deterministic perturbation strengths are fixed for all of the instances where 2 and 3 units are considered for perturbation for any instance. The adaptive perturbation strengths are determined by the size of the instance as a percentage of the number of units in an instance (25%, 50% and 75%).

Type of Perturbation: We considered two different types of perturbation for the GMSP described previously: INSERT and SWAP. The INSERT perturbation modifies a solution by moving a uniformly randomly selected unit from its interval(s) to another

interval or intervals selected uniformly at random, while the SWAP perturbation modifies a solution by exchanging the starting intervals of two uniformly random selected units (without replacement), as Figure 2 shows.

3.4 Acceptance criterion

The acceptance criterion defines the conditions of transition from the current solution to a newly generated one. If the new solution is accepted, it will be perturbed and serves as an initial solution for the next iteration of the search. We consider two acceptance criteria: BETTER and PROBABILITY. BETTER accepts the new solution if it is better than the current solution. This criterion, which is similar to one described in [64], is defined as:

$$BETTER(s^*, s^{*\prime}) = \begin{cases} s^{*\prime} & if \quad f(s^{*\prime}) < f(s^*) \\ s^* & otherwise \end{cases} \tag{6}$$

where $f(s^{*\prime})$ and $f(s^*)$ are the objective functions for the new and current solutions respectively. Such a criterion may result in a very strong intensification of the search. To introduce some diversification of the search, PROBABILITY can accept a worse solution with a probability p. We used three values for p: 1%, 3% and 5%. PROBABILITY can be defined as:

$$PROBABILITY(s^*, s^{*\prime}) = \begin{cases} s^{*\prime} & if \quad f(s^{*\prime}) < f(s^*) \\ s^{*\prime} & if \quad r_p < p \\ s^* & otherwise \end{cases} \tag{7}$$

where r_p is a randomly generated number within the range $[0, 1]$. This criterion can be used to control the balance between intensification and diversification of the search. Note that PROBABILITY tends to make the search more intensive when p is smaller and to make the search more diverse if it is larger.

3.5 Run-length distributions for evaluating ILS

We follow visualization methods introduced in [62] and [36] to support the analysis of our ILS algorithms. These methods, called run-length distributions (RLDs), are useful to see how the solution quality grows with time (or, here, function evaluations) while properly accounting for the stochastic nature of an ILS or other metaheuristic. The RLDs in Figure 3 (a) show the best solution found and various bounds on the solution quality given as the percentage deviation from the best found solution; these are plotted against the run length (number of function evaluations) for a large number of independent algorithm runs.

As well as basic visualization of an algorithm's performance, RLDs can be used to compare algorithms in detail [36,62], which can facilitate the design of algorithm portfolios that perform better than any of the constituent algorithms (see Section 4.4). Figure 3 (b) shows the RLDs for two ILS algorithms, A and B, and the crossing over of their RLD curves indicates that a portfolio based on these two could be beneficial.

Finally, RLDs may also be used to consider restart strategies. By fitting an exponential distribution to an RLD, one can detect stagnation of an optimizer. An example of this method is given in Figure 3 (c), where the run-length distributions of an ILS

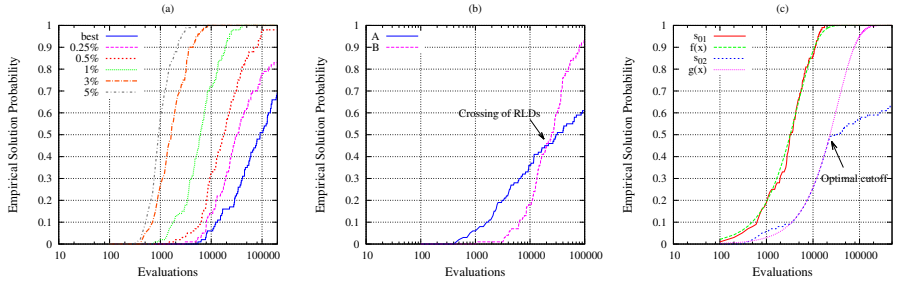


Fig. 3: The empirical run-length distributions for GMSP instances across 100 independent runs of different ILS algorithms in semi-log plots. The x-axis gives the number of evaluations used and the y-axis represents the empirical probability to find a solution. (a) The RLD for an ILS algorithm that is allowed to run for 200000 evaluations. The RLDs are for the best solution found and various bounds on the solution quality given as the percentage deviation from the best found solution. (b) RLDs for two ILS algorithms, A and B. The algorithm allowed to run for 100000 evaluations per each run. (c) The RLDs of an ILS algorithm measured on two GMSP instances, s_{01} and s_{02}. The exponential approximations of the ILS are indicted by $f(x)$ for instance s_{01} and $g(x)$ for instance s_{02}.

algorithm measured on two GMSP instances are plotted. For instance s_{01}, the run length distribution is well approximated by the exponential distribution $f(x)$. This indicates that the restart strategy would not improve the algorithm performance on this instance. The instance s_{02}, on the hand, develops significantly below the exponential curve $g(x)$ from the tangential point on the empirical RLD. Hence, the algorithm suffers from a stagnation behaviour when optimizing this instance and the tangential point (here at about 18000 evaluations) is hypothesised to be the optimum cut-off value [64]. However, an optimum value of a cut-off may depends on the particular GMSP instance. Hence, Stützle suggested using soft restarts instead of applying fixed cut-offs for restarting an ILS algorithm [63,65,64].

We refer the interested reader to the extensive discussion provided by Hoos [36] and Stützle [62] in their PhD thesis where they discussed the RLDs theoretically and practically.

4 ILS-extended algorithms

In the previous section, we have presented different basic ILS operators for the GMSP. In the following, we discuss four possible extensions to the base method: using a restart strategy, use an objective function with delta evaluation, a hybrid with a VND algorithm, and an algorithm portfolio of ILSs.

4.1 ILS with restart strategy

Stagnation, where an algorithm fails to find a better solution for an extensive number of evaluations, can afflict metaheuristics such as ILS, reducing their effectiveness. The

simplest solution to this problem is to restart the algorithm from a new initial solution after a predefined number of evaluations (cut-off value). As stated in Section 3.5, an empirical RLD can be approximated by an exponential distribution, while departure from that distribution can serve to detect stagnation and help to identify the most appropriate cut-off value for the restart. Based on a well-known result from probability theory, if a given algorithm has an exponential RLD and it is allowed to run k runs for e evaluations, the probability of finding a target solution using such configurations is the very same as when running the algorithms once for evaluations $k.e$. Hence, using a restart strategy for an exponentially distributed ILS algorithm will not affect the probability of obtaining the target solution [64,65]. Restarting an ILS algorithm will enhance its performance only if the empirical run-length distribution falls below the plotted exponential.

4.2 ILS with delta evaluation

Delta evaluation is an important technique to reduce the amount of computational work performed by local search algorithms [9,10]. In this approach, any solution is partially evaluated by computing only the cost difference between that solution and its neighbouring solution. When scheduling the generator maintenance tasks, it considers only the fitness function contribution of intervals that are not common between the two schedules, which reduces the repeated evaluations. Let us demonstrate the usefulness of this method using an example of GMSP. Consider the case where a unit i with a duration of d_i needs to be moved from its current position within a range of total intervals of T. Usually, the solution is evaluated by computing the change of the nett reserve on each interval, which is equal to T times. Using delta evaluation, only the intervals that the unit i has moved from and to are computed and these will be in the range between $d+1$ and $d \times 2$ intervals. Hence, the cost of moving the unit i evaluated using the delta function compared to using a full evaluation would be in the range between $(d+1)/T$ and $(d \times 2)/T$ to 1. For instance, for a unit with a duration of $d_i = 6$ and scheduling horizon of $T = 52$, the delta evaluation would be between 0.135 and 0.231 evaluations. This method is more effective for schedules with many units having a small number of durations.

In this ILS variant, the algorithm has the same components as a standard ILS, but a different fitness function. Delta evaluation is utilized and the new fitness is calculated using the function: $SSR_{new} = F_{delta}(solution_{old}, solution_{new}, SSR_{old})$. This function uses the information from the fitness and content of an existing solution in order to calculate a new neighbouring solution's fitness much more rapidly.

Algorithm 2: HYBRID ILS/VND

```
1 s_0 ← GenerateInitialSolution;
2 s ← VND(s_0);
3 repeat
4 |   s' ← Perturbation(s, history);
5 |   s'' ← VND(s');
6 |   s ← AcceptanceCriterion(s, s'', history);
7 until termination condition met;
8 return s
```

4.3 ILS/VND hybrid

Each of the various exact methods and metaheuristics has its assets and shortcomings. By exploiting individual advantages, algorithms can be combined to form hybridized algorithms with better overall performance. Hybrid metaheuristics tend to be superior to their pure counterparts in terms of solution quality and computational time, especially when optimizing larger-scale instances [7]. Several ILS and VND [35] hybrid algorithms are proposed for several problems, such as the Vehicle Routing Problem [67], the attribute reduction in rough set theory [4] and the TSP [55]. In this section, we propose a new hybrid ILS and VND algorithm for solving the GMSP. This hybrid integrates a VND procedure into the framework of an ILS algorithm.

Algorithm 3: VND Procedure

```
1  N: Set of neighborhood structures ;
2  k: Index of the current neighbourhood structure ;
3  Set k = 1 ;
4  while k ≤ |N| do
5  |   s'' ← LocalSearch(s', N_k) ;
6  |   if s'' dominates s' then
7  |   |   s'' ← s' ;
8  |   end
9  |   k = k + 1 ;
10 end
11 return s''
```

Variable neighbourhood descent is a variant of Variable Neighbourhood Search (VNS) [35], where there is no shaking step and the change of neighbourhood is performed in a deterministic way, in the descent to local optimum. More precisely, the idea behind VND is to alternate between different neighbourhood structures where the local minimum found by a local search algorithm on one neighbourhood is the starting point of the algorithm within the next neighbourhood. The framework of the proposed algorithm is presented in Algorithm 2 and explained as follows:

Initial solution generation: The initial solution is generated using the same heuristic described in section 3. Each unit is assigned exactly one job in the scheduling horizon and the outage period of this job is determined uniformly at random.

Base ILS algorithm: The ILS considered here utilizes an INSERT(3) perturbation operator, BILS as its local search and the BETTER acceptance criterion. These ILS components have been described earlier in Section 3.

Variable Neighbourhood Descent: In GMSP, an incumbent solution can often be improved by a valid movement of the units from their current intervals to others within the scheduling horizon. How many units should move determines a neighbourhood's structure and size. This moving process is essential to the definition of the neighbourhood structures used in the proposed algorithm, which defines two neighbourhoods:

N_1: This is a large neighbourhood structure where the effective search space can be as large as T^I. An INSERT operator is used to remove a unit from interval i and reinsert it in interval j, anywhere in the scheduling horizon. If the maintenance period has multiple intervals, i and j represent the first interval of the maintenance period.

N_2: This is a small neighbourhood search structure where the effective search space can be approximatly $2I$. An INSERT operator is also used to remove a unit, but the movement is limited to one interval earlier and one interval later than the current first interval of the maintenance period.

The VND procedure, described by Algorithm 3, is used to search the neighbourhood defined by the operator N_k of the current solution using the local search component. The procedure starts by exploring in the large neighbourhood N_1 until a local optimum is reached, then it performs a small neighbourhood search for a better solution than the current one in N_2. The best solution found by the procedure is returned.

Stopping criterion: Similar to the other ILS variants, the hybrid algorithm stops when it reaches the maximum number of evaluations allowed.

The performance of the hybrid algorithm is tested and compared with other ILS algorithms presented here and some metaheuristic results from the literature in Section 6.

4.4 A portfolio of ILS algorithms

The term *algorithm portfolio* was introduced originally by Huberman et al [37] and was also studied by Gomes and Selman [32]. *Algorithm portfolio* can be defined as "*A collection of different algorithms (heterogeneous portfolio) and/or different copies of the same algorithm (homogeneous portfolio) running on different processors*" [59]. The motivation behind studying algorithm portfolios arose from the fact that there is not a single algorithm which can guarantee to be able to find an optimum or near-optimum solution for a specific type of problem with varying dimensions and complexities where its performance varies based on the problem instance. Portfolios comprising different algorithms, or differently configured instances of an algorithm, can offer better performance even on a single instance, since switching between algorithms is likely to match each phase of the search to a good algorithm, or search can be terminated earlier when one constituent algorithm finds a good solution. Such portfolios can probably be further enhanced by exchanging of information between constituent algorithms. For instance, Peng et. al. [52] have used different strategies to communicate adaptively between consistent algorithms and reported an improvement in performance. In a well-regarded technique called AMALGAM [71,72], multiple different search algorithms are run simultaneously and learn from each other through information exchange using a common population of points. The computational effort of each one is adapted continuously during the course of the optimization, in order to favour individual algorithms that exhibit the highest reproductive success during the search. Results from experiments show that AMALGAM significantly improves the efficiency of evolutionary search.

Algorithm 4: ILS Portfolio

Define:
A : The total number of constituent algorithms;
P : The main population;
φ_a : Algorithm a contribution to overall improvements in objective function where $a = 1, \ldots, A$;
N_a: Number of individuals in Algorithm a to be migrated from its sub-population P_a to P;
P_s : Size of population and sub-populations;
e_{pm} : Maximum evaluations for the portfolio algorithm;
e_{im} : Maximum evaluations for each iteration of optimizing the main population;
ν_a : Minimum contribution of solution to be migrated for each algorithm a;
Set $e_p = 0$ `/* Portfolio evaluations counter */`
Generate initial solutions in population P;
Calculate fitness and constraints violations for all individuals in P;
while $e_p \leq e_{pm}$ **do**
  Set $e_i = 0$ `/* Iteration evaluations counter */`
  while $e_i \leq e_{im}$ **do**
    for $a = 1$ **to** A **do**
      Algorithm a uses each individual in P as s_0 to generate a sub-population P_a of size P_s;
      Calculate fitness and constraints violations for all individuals in sub-population P_a;
    end
  end
  Clear population P;
  Calculate φ_a where $a = 1, \ldots, A$;
  Update $N = N_a, ..., N_A$ according to equation 8;
  for $a = 1$ **to** A **do**
    for $n = 1$ **to** N_A **do**
      while *solution does not exist in* P **do**
        Move N_a solution from sub-population P_a to population P;
      end
    end
  end
  Update e_p;
end
return Best solution

For the ILS algorithms studied here, several observed phenomena suggest that the portfolio approach would be beneficial for optimizing the GMSP. First, no single algorithm has dominated the others and we occasionally observed the crossing of the RLDs of different algorithms. Secondly, we noticed that for different instances, different ILS algorithms showed the best performance. Thus, to improve performance over a broad range of instances, it would be advantageous to combine several of the best-performing ILS algorithms mentioned previously into a portfolio. The approach has proved ef-

fective for solving many problems such as satisfiability, planning and scheduling [61]. Although algorithm portfolios have been applied to different scheduling problems [59, 50], they appear to have limited application to the GMSP. In fact, to the best of our knowledge, it has been applied only as reported in [3], where constituent algorithms in a portfolio were run in parallel (all algorithms being executed concurrently), with no communication between algorithms. A similar application was used here, except that the constituent algorithms were able to exchange information, based on the AMALGAM approach.

The pseudocode of the portfolio algorithm is given in Algorithm 4. It employs a population-based elitism search strategy to find the best solution of a GMSP using the predefined objective function. The algorithm is initiated by creating a main population P of randomly generated solutions using the procedure *GenerateInitialSolution* described in section 3. The fitness value and number of constraint violations are then computed for each solution in the population. Each algorithm a creates a sub-population P_a of the size of the main population, where each individual in P is optimized and the solution achieved is added to P_a. The solutions in all sub-populations are evaluated and their constraint violations are recorded. A defined number of solutions $N = \{N_1, ..., N_A\}$ is migrated from each sub-population P_a for $a = 1, \ldots, A$. This number is different for each algorithm and depends on its performance. A two-step procedure is used to update the N in each iteration. First, we measure for each algorithm a its contribution to the overall improvement in objective function, denoted as φ_a. The second step is to update N for each algorithm a according to this equation:

$$N_a = \left\lfloor (P_s - (\nu_a \times A)) \frac{\varphi_a}{\sum_i^A \varphi_a} \right\rfloor . \tag{8}$$

Based on the calculated values of N, the best solutions are migrated from the sub-populations to the main population. Whatever the values of N, there is always a minimum number of solutions that each algorithm needs to contribute to the main population to avoid the possibility of disabling an algorithm which may provide better solutions in the future with higher evaluations. This learning mechanism ensured that the algorithms with the best performance were rewarded in the next iteration by allowing them to contribute more solutions. As a result, a faster convergence may have been achieved. However, to preserve population diversity and prevent too early convergence of the portfolio algorithm, we took three precautions:

1. A diverse set of ILS algorithms was considered as constituent algorithms for the portfolio. The best performing ILSs using BILS and WBLS were selected; these were A05 (BILS, INSERT(3), BETTER) and A69 (WBLS, INSERT(3), BETTER) respectively, see section 5.2. In the same section, it is shown that ILSs with these two local searches may achieve better performance when optimizing a GMSP cooperatively. In addition, the ILS variants mentioned in this section were considered, as they have different approaches to performing optimization. Two algorithms of each variant were added, one with BILS and the other with WBLS. Choosing constituent algorithms that exhibited different behaviours on the GMSP problem to be optimized may have led to better performance.
2. When migrating a solution from a sub-population to the main population, it was compared to the solutions already in the population, based on the Hamming distance. If the Hamming distance of the solution to be migrated was equal to zero

Table 2: List of the components of the best ILS algorithms.

ILS	Local Search	Perturbation	Acceptance Criterion	ILS	Local Search	Perturbation	Acceptance Criterion
A05	BILS	INSERT(3)	BETTER	***A65***	WBLS	INSERT(2)	BETTER
A69	WBLS	INSERT(3)	BETTER	***A13***	BILS	INSERT(0.5)	BETTER
A01	BILS	INSERT(2)	BETTER	***A33***	FILS	INSERT(2)	BETTER
A09	BILS	INSERT(0.25)	BETTER	***A37***	FILS	INSERT(3)	BETTER
A73	WBLS	INSERT(0.25)	BETTER				

compared with any solution in P, the next better solution was selected, even if its quality was lower than that of the first chosen solution.

3. To introduce new solutions during the search process, we used two ILS algorithms which both had INSERT perturbation and the PROBABILITY acceptance criterion, but different local search algorithms: BILS and WBLS. For both algorithms, the perturbation strength and probability of accepting worsening moves were set high (0.75). New solutions which improved upon the current best solution could then be found during the search.

The portfolio algorithm is stopped when it reached the maximum evaluations allowed. The portfolio seems to be a promising approach to optimize different large, complex and dynamic problems such as the GMSP. Just as for the other ILS variants suggested, the performance of the proposed ILS-based portfolio was investigated on two GMSPs from the literature, as reported in section 6.

5 Selecting ILS best components

In this section, we investigate the effectiveness of different ILS components (in all, 96 ILS algorithm with different components and parameters) using run-length distributions over a set of 11 problem instances. The more advanced variants of the ILS, using delta-evaluation, restarts or a portfolio, were formed from the best of the ILS components tested here; these are evaluated in Section 6.

5.1 Experimental Settings

A set of 96 ILS algorithms was constructed by combining the components described in section 3. To test the performance of these algorithms, we ran them on a set of 11 instances from our own GMSP instance generator, based on the problem model described in section 2. The instances were hand-crafted to represent different sizes and characteristics of generator scheduling problems, with the number of units from 4 to 29. The scheduling horizon for all instances was 52 weeks.

Each algorithm optimized each instance for 100 runs and the mean Percentage Deviation (PD) from the best known solution for that instance was calculated using the formula $PD = (f(s) - f(s_{best}))/f(s_{best}) \times 100$. For each algorithm we also measured the Infeasibility Ratio (IR) (IR = *number of infeasible solutions/ number of runs* $\times$ *100*). All algorithms were allowed to run to their maximum evaluations, calculated by the formula $maxEvaluation = \alpha I + \beta$, where $\alpha = \beta = 20000$ and I is the total number of units for an instance (Table 1). A list of the ILSs tested, showing the their components and the instance details, is provided at [1], including instances and results.

Timing information is given below; these results are from an implementation in Java run on a server with 12 cores (2x6 XEON-5690 cores @ 3.6 GHz) using 32 GB of RAM.

5.2 Results and findings

For the sake of making this paper short, a detailed discussion of the results is provided in the supplementary section S1 while in this section we wish to identify some 'overall best' ILS algorithms. To do so, we ranked the PD and IR values obtained by all algorithms for each instance. In both cases, these rankings start from zero and the lower the rank, the better the performance of the algorithm. Each algorithm's ranks were then summed for all instances and plotted in Figure 4. The left plot of the figure shows the sum of the ranks for all the algorithms divided into three groups, based on the local search algorithm used. It can be observed from the plot that the ILS algorithms using BILS tended in general to perform better than the other ILSs and to have a spread of data points towards the bottom left. On the other hand, the ILS algorithms with FILS seem to have varied in performance, indicating that they struggled to find feasible and good quality solutions for all instances. The best performing ILS algorithm with FILS was A33. The ILSs with WBLS showed less variation in performance than those with FILS, but more variation than the ILSs with BILS. The seven best ranked ILSs are shown in the right plot of the figure and listed in Table S1, while Table 2 lists their components and those of algorithms A33 and A37. All nine of these ILSs managed to obtain feasible solutions for all runs on all instances; IR = 0. The best ranked ILS algorithm was A05, with the components BILS, INSERT(3) perturbation and BETTER acceptance criterion. This choice of configurations seems to have allowed the ILS algorithm to achieve good solutions for all GMSP instances. Indeed, it was the best solver for many of them. Algorithm A69 also showed solid performance over all instances.

As an outcome of this analysis, we propose the ILS algorithms A05 and A69 as effective algorithms for GMSPs such as the one modelled here. The implementation and behaviour analysis of these algorithms showed them to be easily adaptable alternatives to other more complex metaheuristics, as they showed excellent performance after some straightforward optimizations. The performance of the proposed ILSs were tested on two benchmark GMSPs from the literature and compared to the results for other metaheuristics on the same instances.

Finally, there seems to have been a strong positive correlation between the IR and PD sums of ranks of the ILS algorithms, as can be seen from the scatter plot in Figure 4 (Spearman rank correlation coefficient $\rho = 0.976$). This suggests that the ability to obtain feasible solutions was strongly linked to the ability to find high quality solutions for these instances.

5.3 Overall benchmark results

Results for 100 runs of all 96 algorithms across the full set of 11 instances are presented in Table 3. It should be noted that every one of the algorithms managed to find a feasible solution for instance s_{10} during all runs. On the other hand, a feasible solution for instance s_{09} proved difficult to find on more than half of the optimization runs.

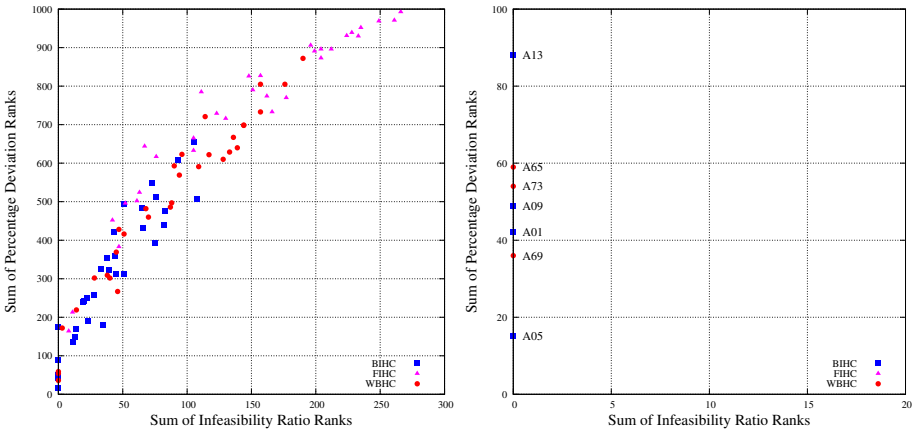


Fig. 4: The performance of different ILS algorithms represented as the sum of the ranks of the algorithms percentage deviation against the sum of the ranks of their infeasibility ratio. For each algorithm optimizes an instance, the percentage deviation from the best know solution for that instances is calculated for all runs and then averaged over them while the infeasibility ratio is calculated according to the equation in Section 5.1. The percentage deviation and infeasibility ratio ranking starts from zero. The lower the x-axis and y-axis values the better.

The last column shows the number of times the best solution was found for an instance among all optimizations. The highest number of runs resulting in a best solution was obtained on instance s_{01}, yet it represented only about 29% of the optimization runs. For the instances s_{07} to s_{11}, which were the largest instances in the set, only once was the best solution achieved, and in general they seem to have been the hardest among the set. As we belive that this set of instances are likely to be valuable as benchamrks for the GMSP, due to the different sizes and characteristics of the instances, we have made it available to the optimization community and it can be downloaded from [1].

6 Comparative study

This section reports the experimental evaluation of the performance of two standard ILS algorithms and the ILS variant algorithms proposed in section 4 on the test systems described in section 6.1. The standard ILS algorithms are those proposed in section 5.2 based on their overall performance: A05 and A69, referred to here as `ILS-BILS` and `ILS-WBLS` respectively. The ILS with the restart strategy is referred to as `Restart` and that with delta evaluation as `Delta`. The ILS/VND hybrid is referred to as `Hybrid`. Finally, we refer to the portfolio of different ILS-based algorithms simply as `Portfolio`. For each problem, all algorithms were run for 100 independent trials and were given the same number of evaluations per optimization run. In order to facilitate fair comparisons for the 21-unit system, we used the same number of evaluations (30000) as were used in previous studies which investigated this problem. As the 32-unit system had more units, the algorithms were given 41000 evaluations per run to optimize the problem. All experiments were implemented in Java and run on the same hardware configuration

Table 3: Computational times required to find the best solution (averaged over 100 runs and algorithms), percentage of feasible solutions (of all runs and algorithms) and the number where the best fitness was achieved (among 9600 optimizations).

Instance	*Number of units*	*Best Solution*	*Avg Best Solution*	*Min Time (s)*	*Avg Time (s)*	*Max Time (s)*	*% Feasible Solutions*	*NBSA*
s_{01}	4	15995312	16417081.42	0.49	0.57	1.31	82.48	2789
s_{02}	8	49680	54918.41	1.26	1.35	2.32	78.9	721
s_{03}	9	5075	5252.88	1.48	1.82	2.51	96.99	667
s_{04}	12	11025	12415.38	2.4	2.92	3.31	80.23	21
s_{05}	14	11799791	12256051.82	3.24	3.87	6.47	98.82	542
s_{06}	16	110349	125927.05	4.06	4.34	6.35	72.74	12
s_{07}	17	80975	82373.58	4.46	4.46	4.46	99.96	1
s_{08}	21	480832	502959.79	6.67	6.67	6.67	86.65	1
s_{09}	24	639884	652454.35	7.93	7.93	7.93	40.33	1
s_{10}	25	18320744	18385367.47	8.44	8.44	8.44	100	1
s_{11}	29	980211	1034613.82	13.7	13.7	13.7	97.14	1

mentioned in section 5.1. The resultant objective functions (SSR) for all runs of all algorithms are available to be downloaded at [1].

6.1 Test systems

The case studies considered in this research as benchmarks to investigate the effectiveness of the proposed algorithms are two GMSP test systems obtained from the literature in which reliability is considered as the optimality criterion. The objective of both problems is to minimise the sum of the squares of the reserve (SSR) over a 52-week scheduling horizon while meeting the problem constraints.

The first test system comprised 21 units. It was loosely derived from the problem in [76] and presented with some simplifications and additional constraints in [20]. Each unit must be maintained continuously for a given duration within a specified window, either in the first or second half of a year's scheduling horizon, while meeting problem constraints such as the load demand and the availability of maintenance crew. The details of the problem including the unit capacities, maintenance allowed periods, duration of maintenance and manpower needed are available in [20, 19].

Several metaheuristics have been applied to this test system. They are GAs in [20, 18, 21], Simulated Annealing (SA) in [21] and hybrid algorithms (GA,SA and a heuristic) in [19]. The best results were obtained by Foong [29] using the Ant Colony Optimisation (ACO) algorithm. Recently, Schlünz and Vuuren [57, 58] managed to match the best known objective function found by Foong using an SA algorithm and an SA/Heuristic hybrid. However, they were unable to match or improve the best known average incumbent solution quality.

The second test system was the 32-unit scheduling problem first introduced in [58]. This is a modified version of the system presented in the 1979 IEEE Reliability Test System [66], with parameter values and constraints added. It has additional exclusion constraints compared to the 21-unit problem, to prevent certain units from being in a state of simultaneous maintenance. The problem has a safety reserve to be considered during scheduling and is assumed to be highly constrained by both its maintenance crew and power demand constraint sets. The details of the test system are available in [57, 58].

6.2 Algorithm parameters

All ILS variants were extensions to the standard `ILS-BILS` algorithm. The `Restart` algorithm required a single parameter (∂) to be set in order to calculate the cut-off evaluations. If the objective function of a solution has not improved for $e_r = I.\partial$ evaluations, then the algorithm will start with a new random solution where $\partial = 210$. The `Delta` and `Hybrid` algorithms have no parameters to be set. Finally, the `Portfolio` algorithm starts by creating a P of size 50. The same size was chosen for the sub-population of the algorithms. The e_{im} was set according to the formula ($e_{im} = \alpha I + \beta$, where $\alpha = 100$ and $\beta = 900$). The minimum number of solutions that an algorithm a needs to contribute was set to $\nu_a = 2$. All the parameter settings were based on preliminary experiments performed to check the effects of different settings on the performance of the algorithms.

6.3 Performance of ILS variants

Similar to section 5.2, we provide an extensive results and detailed discussion of them in the supplementary section S2. Here, we can summarize the above observations from Figures S6 and S7 concerning the performance of the ILS standard and variant algorithms as follows:

- Keeping in mind that `ILS-BILS` and `ILS-WBLS` were standard ILS algorithms that had not been modified or tuned on the test problems, they showed good performance, especially `ILS-WBLS`. This difference can be credited to the advanced local search in WBLS, as the two algorithms were similar in terms of their other components.
- The restart strategy was not effective for the ILS algorithms, indicating that they did not suffer from stagnation behaviour when optimizing the GMSP modelled in this work.
- Implementing the VND within the ILS framework as a hybrid algorithm significantly improved the performance of the ILS algorithm on both objective values achieved and computation evaluations required.
- As shown by the plots, the principle of using a portfolio of ILS algorithms that can communicate proved to be efficient and effective in optimizing the test instances in a short time and on finding excellent solutions. These findings encourage the use of such a method for larger and harder problems, such as those ocurring in the industry.

We validated these outcomes by a further exploration of the results using applicable statistical tests (see supplementary section S3). These tests confirmed the conclusions mentioned earlier, as follows:

- There were statistically significant differences in performance between the ILS algorithms with BILS and WBLS for both test cases. Both median and mean values showed better performance for ILS with the WBLS algorithm.
- The t-test (21-unit problem) and Mann-Whitney test (32-unit problem) revealed no significant difference in the SSR results of the standard ILS algorithm (`ILS-BILS`) and its variant with the restart strategy (`Restart`). This indicates that the ILS algorithms did not suffer stagnation behaviour on the GMSP instances studied here, making the restart strategy superfluous.

Table 4: Comparison of ILS algorithms with other algorithms from previous studies. Column three represents the best fitness obtained in all runs, while columns four and five list the average (over 100 runs) of the run output and standard deviation respectively. Column six lists the median and the worst fitness achieved during the runs is shown in column seven. Column eight shows the number of evaluations required by an algorithm to find the best solution in a single run, averaged over 100 runs.

System	Algorithm	Best $\times 10^7$	Mean $\times 10^7$	StD $\times 10^5$	Median $\times 10^7$	Worst $\times 10^7$	Average Evaluations
21-Unit System	ACO [28]	**1.3665**	1.3682	0.11	1.3681	1.3722	13593
	GA [20]	1.3791	1.4671	-	-	-	-
	SA [19]	1.4049	1.4606	-	-	-	-
	PSO [77]	1.3749	1.3871	0.11	-	1.4015	-
	SA [58]	**1.3665**	1.3988	0.19	-	-	-
	MIQP [57,58]	1.3973	-	-	-	-	-
	Global Solver [57,58]	1.3884	-	-	-	-	-
	ILS-BILS	1.3845	1.4488	3.53	1.4502	1.5575	21926.5
	ILS-WBLS	1.3675	1.4234	3.82	1.4252	1.5449	15854.4
	GA/SA hybrid [19]	1.3812	1.4578	-	-	-	-
	SA/Heuristic hybrid [58]	**1.3665**	1.3732	-	-	-	-
	GA/SA/Heuristic hybrid [19]	1.3910	1.4171	-	-	-	-
	Restart	1.3812	1.4547	3.32	1.4561	1.5312	21139.5
	Hybrid	**1.3665**	1.3721	1	1.3687	1.4193	10518.7
	Delta	1.3681	1.3884	1.94	1.384	1.4486	18079.9
	Portfolio	**1.3665**	**1.3669**	**0.05**	**1.3665**	1.3681	**4675.7**
32-Unit System	SA [57,58]	3.3639	3.3709	7.3	-	-	-
	MIQP [57,58]	3.3904	-	-	-	-	-
	Global Solver [57,58]	3.5463	-	-	-	-	-
	ILS-BILS	3.3847	3.4084	1.47	3.4066	3.4445	36522.4
	ILS-WBLS	3.3726	3.3913	1.06	3.3903	3.4166	36041.2
	SA hybrid [58]	**3.3627**	3.37	-	-	-	-
	Restart	3.3822	3.4082	1.71	3.4041	3.4867	36799.8
	Hybrid	3.3644	3.377	0.69	3.3754	3.3957	24137.2
	Delta	3.3682	3.3868	0.92	3.3852	3.4111	36578.9
	Portfolio	3.3628	**3.3667**	**0.17**	**3.3668**	3.3698	**24096.8**

- The tests revealed a statistically highly significant difference in the results of the standard ILS algorithm (`ILS-BILS`) and the versions with the delta evaluation (`Delta`) and VND (`Hybrid`) implementations. The median and mean values demonstrate that the hybrid ILS variants performed much better than the standard version.
- When comparing the performance of the `Portfolio` and `Hybrid` algorithms which achieved the best results among all algorithms, as shown in Figures S7 and S8, the tests show a superiority of performance for `Portfolio` over `Hybrid`, particularly on the 21-unit system.

6.4 Comparison with other methods

As described in section 6.1, the 21- and 32-unit systems had been studied previously using different optimization methods, as reported in the literature. The results obtained for the ILS algorithms are compared with those from the literature in Table 4, which shows the objective function values (MW^2) as the best SSR achieved, the mean averaged over 100 runs, the standard deviation, the median and the worst fitness value. In addition, it lists the number of evaluations required to reach the best solution in a single run averaged over the runs. It should be mentioned that the previous studies

mainly reported the best solution and average only to measure the performance of their proposed methods. For the results of the ACO, we used the raw data available in Foong's dissertation [28] and performed the required statistical analysis.

To make a fair judgement of the performance of the ILS algorithm and its variants, we divided the optimization algorithms in Table 4 into two groups: the standard optimization methods and the non-standard methods, such as the hybridized algorithms and the portfolio. Looking at the performance of the standard algorithms on the 21-unit system, we notice that the ACO and SA algorithms provided by [58] achieved the best results, followed very closely by ILS-WBLS. On the 32-unit system, the same SA performed the best among the standard algorithms, followed by ILS-WBLS and ILS-BILS. Considering the non-standard algorithms, Table 4 shows that the `Portfolio` algorithm was able to match the best known objective function value. Compared to the ACO results and a hybridized version of the SA mentioned above, `Portfolio` performed better in terms of the average, standard deviation and median values.

Furthermore, the average number of evaluations on all runs that `Portfolio` took to find the best solution was less than 35% of the number that ACO needed. The performance of `Portfolio` looks better than the ACO according to these figures; in order to confirm this outcome, we used statistical procedures similar to those which we used before. First, we checked the normality of the results (number of runs = 50) of the ACO algorithm. The K-S test of normality after Lilliefors significance correction indicated that the results were not normally distributed, as the *p-value*= 7.140E-08, which is less than the significance level $\alpha = 0.05$. Hence, the non-parametric test (Mann-Whitney) was used, as the `Portfolio` data were not normally distributed, as explained previously. The test revealed a statistically significant difference in the performance of the two algorithms, as the probability value ($p = 3.421$E-14) was much less than the significance level of $\alpha = 0.05$. The `Hybrid` algorithm also managed to match the best objective function value, but could not improve on the ACO algorithm in terms of the other statistical values.

On the 32-unit system, none of the ILS algorithms was able to match the best objective function value obtained by [58]. However, `Portfolio` achieved a value very close to it and improved the average value. Unfortunately, no performance metrics other than the best value and the average are available from this previous study. The second best algorithm in terms of performance was `Hybrid`, which managed to find good solutions in almost the same average number of evaluations as the `Portfolio` algorithm.

Finally, it should be mentioned that the SA [57,58] algorithm was heavily tuned on both test instances, while no tuning was applied to the ILS algorithm and its variants; yet the standard ILS algorithm (ILS-WBLS) performed well compared to SA and to the methods used previously. In addition, results show that the ILS variants `Hybrid` and `Portfolio` were able to find the best solutions in a much smaller number of evaluations. Generally, `Portfolio` seems to be the most robust optimization method among all those tested and compared to the methods reported in the literature.

The results for all algorithms, and the best schedules found for the 21 and 32 units system, are available at [1].

7 Conclusion

Iterated Local Search (ILS) is an efficient and effective metaheuristics yet it is simple to implement. It has been applied to many combinatorial optimization problems. In this paper, we explored the application of ILS to the Generator Maintenance Scheduling Problem (GMSP). Several ILS operators are proposed and used to form many ILS algorithms. The ILSs have been tested on in-house developed GMSP instances. Their Run-Length Distributions (RLD) have empirically analysed where they revealed that there is no single ILS algorithm that dominates the others and the ILS algorithms did not show stagnation behaviour. Moreover, crossover of some algorithms RLD have been noticed which indicates that these algorithms can perform better if they work cooperatively. Two ILS algorithms were proposed to optimize the GMSP. We also provide a benchmark for future studies of this problem.

Based on the observations from the RLD, several extensions to a standard ILS design are developed and analysed, including specialised operators and delta-evaluation, as well as restart and portfolio strategies and an ILS/VND hybrid. The ILS algorithms have been tested on two benchmark problems obtained from the literature. The experimental and statistical analysis we carried out showed a superior performance of the portfolio and hybrid ILS variants. However, the restart strategy has not been successful to improve the final solution which confirms the previous finding. In addition, the standard ILS showed a good performance and managed to find good quality solutions.

Despite the simplicity of the ILS framework, the experimental results show that algorithm can obtain very high solution quality. In fact, some ILS variants are new state of-the-art algorithms for one of the GMSP benchmark problems.

Supplementary Materials

S1. Selecting ILS best components discussion

Figure S1 shows the empirically observed RLDs for three ILS algorithms with INSERT(3) perturbation operator and BETTER acceptance criterion but different local search algorithms on the GMSP instances s_{01}, s_{02}, s_{03} and s_{05}. Figure S2 is similar but shows the effects of different choices of perturbation operators and strengths, whilst Figure S3 and S4 show the effect of acceptance criteria choices. Finally, Figure S5 illustrates the empirical run-length distributions for the two GMSP instances s_{01} and s_{03} with ILS algorithms that uses INSERT(3) perturbation, BETTER acceptance criterion and BILS and WBLS as their local search operators. The figure shows the RLDs of the best solution found at several levels of required solution quality. Additionally, exponential distributions which may indicate stagnation behaviour are fitted to approximate the RLDs for the best quality solutions. It can be noticed that on both instances using the BILS and WBLS algorithms, it is rather easy to get within 1% or even 0.5% of the best solution. Yet, if a higher solution quality is required, the performance of the ILS algorithms is less good. In particular, when the WBLS is the local search, the plots indicate that a higher number of evaluations are needed to reach the target solution quality. An important observation from the plots is that the run length distribution is well approximated by the exponential distribution $f(x)$. This indicates that the restart

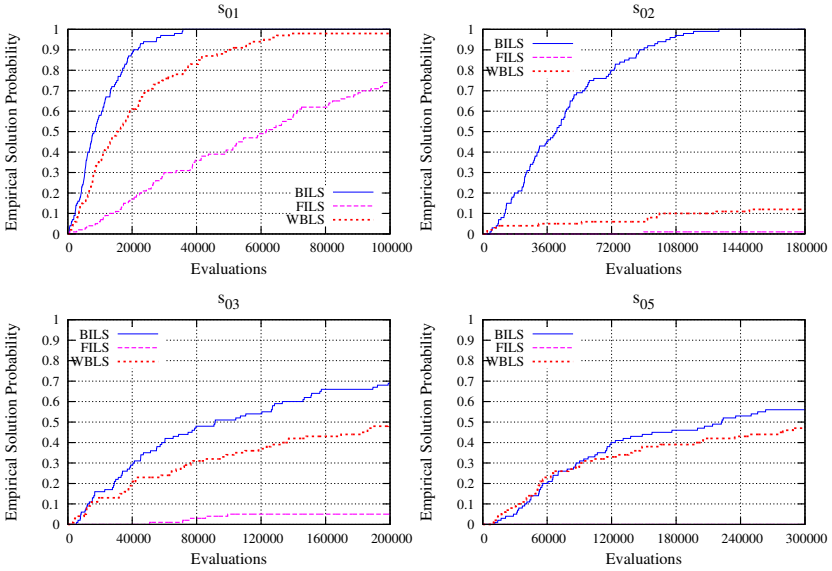


Fig. S1: A comparison of empirical run-length distributions of ILS algorithms for the different choices of local search algorithms. The ILS algorithms are with BETTER acceptance criterion, INSERT(3) perturbation and different local search algorithms (BILS, FILS and WBLS).

strategy would not improve the algorithm performance on these instances. Later, we will examine this observation experimentally and statistically.

A review of all these RLDs allows some useful observations to be made. In general, no single individual ILS algorithm dominated all the others. Hence, there was no single 'winner' among the components tested for the local search, perturbation and acceptance criterion operators. For each operator configuration, we found a mix of performance, depending on the instance being optimized. Table S1 shows the seven members of the set of 96 ILS algorithms that achieved the best percentage deviations (the lower the better) for all instances averaged over all runs. All of these algorithms had a 0 infeasibility ratio, which means that they managed to find feasible solutions for all the runs. Based on the PD values, instance s_{01} appears to have been the easiest problem to solve, as all the best algorithms found the best solution to it on all runs. Conversely, instances s_{04} and s_{06} had the largest PD values, indicating that they were the hardest of the eleven problems.

Regarding the local search algorithms, the performance of BILS seems to have improved as the number of evaluations increased, while the WBLS algorithm seems to have performed better at the beginning of the search. This may be explained by the ability of WBLS to guide the search faster than BILS to the feasible area of the search space. Figure S6 compares the number of evaluations needed by three ILS algorithms with INSERT(3) perturbation, the BETTER acceptance criterion and the local search algorithms used in this work (BILS, FILS and WBLS) to move the search to the feasible region of the search space for all instances. In addition, the figure shows the lowest number of evaluations required that was achieved by any algorithm, aver-

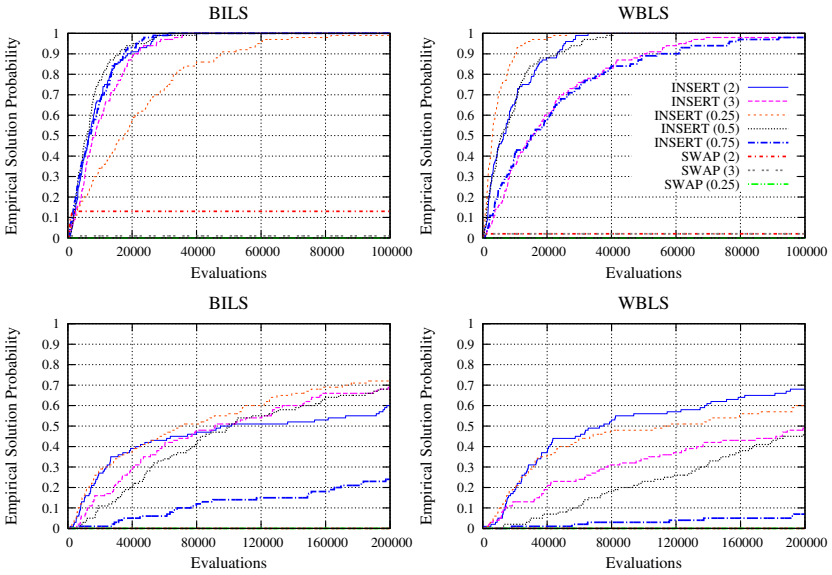


Fig. S2: A comparison of different choices for the perturbation operators. The ILS algorithms are with two local search algorithms (BILS, WBLS) and BETTER acceptance criterion but different perturbation operators and strengths for instance s_{01} (top) and s_{03} (bottom). The legend in the top right plot is applicable to all plots in the figure.

aged over all the runs. On all instances, the WBLS algorithm always needed a lower number of evaluations than BILS and FILS (note that the y-axis is a log scale). By examining the algorithms that achieved the lowest number of evaluations, we found that they all implemented WBLS as their local search operator but had different perturbation operators and acceptance criteria. However, the results for ILS with WBLS presented here are relatively close to the lowest ones achieved by other algorithms. For the ILS algorithms, we observed that domination between different algorithms and configurations seems to be the rule. In many cases, a crossing of RLDs was observed for instances within our benchmark suite. Such an observation can generally be exploited to enhance the overall performance on optimizing an instance by combining a suitable configuration of ILS algorithms—such as an ILS with BILS and an ILS with WBLS—into algorithm portfolios. This behaviour was also noticed when testing the acceptance criteria, where accepting worse solutions by PROBABILITY yielded a better performance, leading to the suggestion that modifying PROBABILITY to reflect the quality of a solution would improve the performance. In other words, solutions only marginally worse than the current solution would have a higher probability of acceptance than a plainly awful solution (as is the case for simulated annealing and threshold accepting, for example).

S2. Performance of ILS variants dicussion

Figure S7 and S8 present the respective results of optimizing the 21- and 32-unit systems by ILS algorithms. Each figure consists of three plots. The top one is a scatter

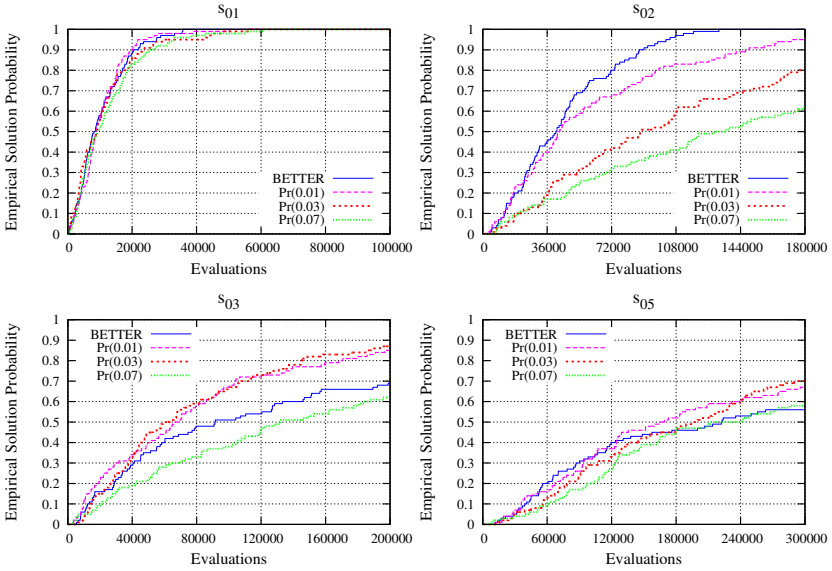


Fig. S3: A comparison of different choices for the acceptance criteria when BILS is the local search algorithm with INSERT(3) perturbation. The x-axis represents the number of evaluations and the y-axis is the cumulative empirical solution probability.

Table S1: The best algorithms for all instances based on the percentage deviation (PD) achieved, averaged over 100 runs.

s_{01}		s_{02}		s_{03}		s_{04}		s_{05}		s_{06}	
Algorithm	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*
A01	0	***A05***	0	***A13***	0.084	***A73***	2.891	***A05***	0.15	***A05***	3.416
A05	0	***A09***	0	***A09***	0.097	***A05***	3.17	***A13***	0.152	***A09***	3.437
A13	0	***A13***	0.054	***A05***	0.104	***A09***	3.341	***A09***	0.157	***A01***	4.138
A17	0	***A01***	0.067	***A69***	0.113	***A13***	3.349	***A29***	0.2	***A73***	4.245
A65	0	***A17***	0.481	***A65***	0.121	***A69***	3.669	***A25***	0.201	***A69***	4.292
A73	0	***A33***	2.231	***A73***	0.123	***A65***	3.765	***A17***	0.229	***A13***	4.776
A77	0	***A41***	2.439	***A01***	0.143	***A01***	3.925	***A01***	0.229	***A29***	4.908

s_{07}		s_{08}		s_{09}		s_{10}		s_{11}		**Sum of Ranks**	
Algorithm	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*	*Algorithm*	*PD*
A05	0.212	***A69***	0.616	***A69***	0.535	***A05***	0.026	***A21***	0.506	**A05**	15
A01	0.214	***A73***	0.698	***A65***	0.565	***A01***	0.029	***A25***	0.51	**A69**	36
A09	0.222	***A65***	0.721	***A05***	0.613	***A25***	0.03	***A69***	0.545	**A01**	42
A65	0.317	***A25***	0.78	***A93***	0.629	***A69***	0.035	***A89***	0.568	**A09**	49
A69	0.343	***A29***	0.782	***A01***	0.711	***A93***	0.036	***A05***	0.641	**A73**	54
A73	0.38	***A05***	0.81	***A73***	0.868	***A65***	0.037	***A01***	0.681	**A65**	59
A29	0.394	***A01***	0.836	***A09***	0.931	***A29***	0.037	***A65***	0.685	**A13**	88

plot of the optimization runs performed by all algorithms, which shows the trade-off between the best objective function found (SSR) and the number of evaluations needed to find it. The bottom left plot illustrates the same trade-off between the SSR and the number of evaluations averaged over all runs. Finally, at the bottom right of the figure is a boxplot which presents a graphical view of the distribution of the best solutions found and compares multiple algorithms. The lower the box in the boxplot, the better the performance of the corresponding algorithm.

In the scatter plot of Figure S7, we can discern two different patterns of behaviour of the algorithms. The `ILS-BILS`, `ILS-WBLS` and `Restart` solutions are spread over all the plot, showing the stochastic nature of these algorithms. It should be remembered

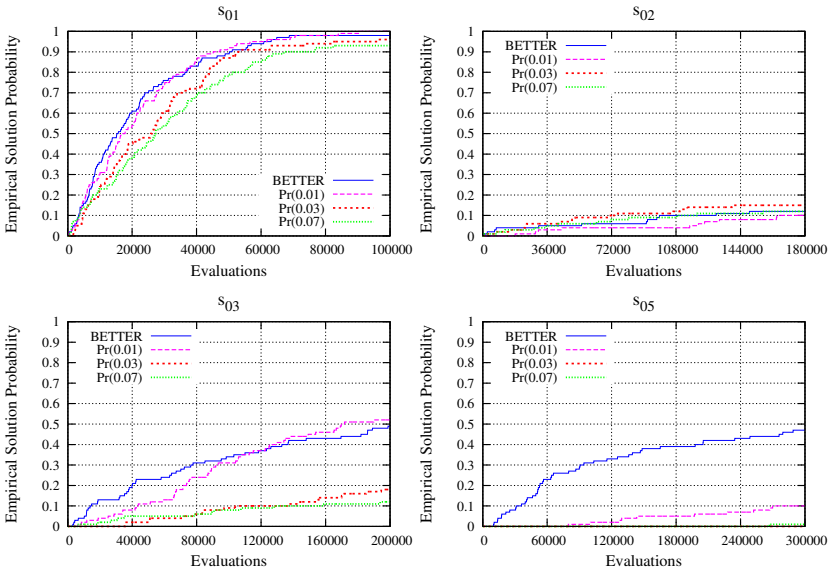


Fig. S4: A comparison of different choices for the acceptance criteria when WBLS is the local search algorithm with INSERT(3) perturbation. The x-axis represents the number of evaluations and the y-axis is the cumulative empirical solution probability.

that they were basic ILS algorithms without modification. However, `ILS-WBLS` found many solutions using a relatively low number of evaluations compared to the other ILSs. As discussed previously, the local search component of `ILS-WBLS` was designed to find feasible solutions quickly and it seems to have performed well on this test system. Almost all of the solutions found by the `Hybrid` and `Portfolio` algorithms were close to the best solution known for the test system. While `Hybrid` and `Delta` found the solutions with different values of evaluations, the portfolio algorithm found all of the solutions in fewer than 12000 evaluations.

Averaging the SSR and evaluation results for the solutions provides an indication of the overall performance of the ILS algorithms. In the bottom left plot, the superior performance of the `Portfolio` is noticeable, as it had the best average SSR and the lowest computational cost of optimization. The `Hybrid` algorithm also appears to have performed well, achieving an average SSR close to that of `Portfolio` at almost double the computational cost. On the other hand, the `ILS-BILS` and `Restart` algorithms demonstrated the worst performance. Their average SSR and evaluations were close; however, `ILS-WBLS` showed a better overall performance compared to them. It was even better in terms of evaluations than the `Delta` algorithm, but worse in terms of the objective function value.

A better overview of the distribution of the solutions is represented by the boxplot. It can be seen that there was substantially more variation in the ranges of `ILS-BILS`, `ILS-WBLS` and `Restart` compared to the `Hybrid`, `Delta` and `Portfolio` algorithms. `ILS-BILS` and `Restart` had similar solution ranges and relatively close median values, while `ILS-WBLS` had a better inter-quartile range and median. It tended to find better solutions, as shown by the fact that its lower whisker was proportionally shorter than

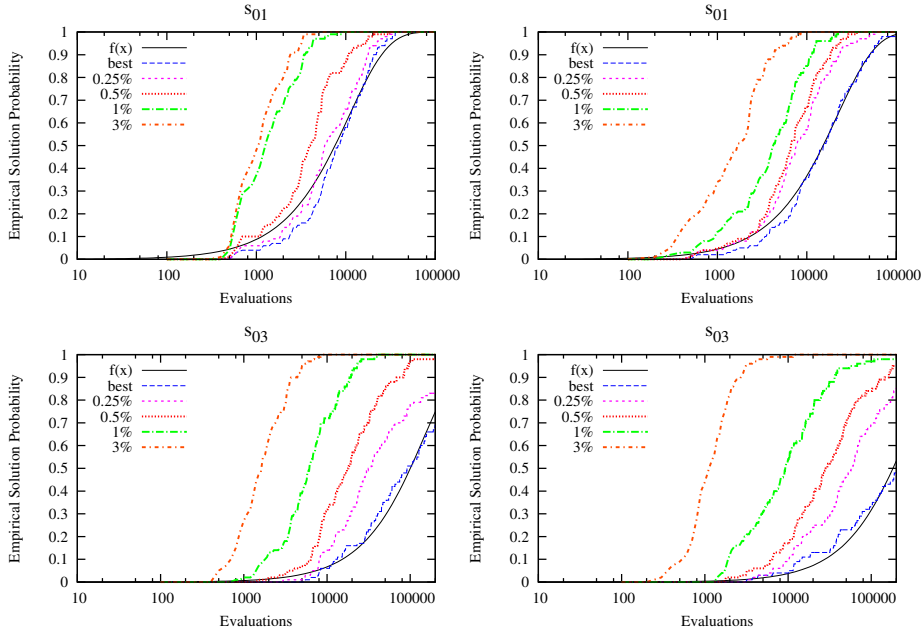


Fig. S5: The plots show empirical run-length distributions across 100 independents runs of an ILS algorithm with BETTER acceptance criterion, INSERT(3) perturbation and BILS local search algorithm for instances s_{01} (top left) and s_{03} (bottom left) and WBLS local search for s_{01} (top right) and s_{03} (bottom left). Several bounds on the best known solution quality are given as percentage deviation. The $f(x)$ curves represent approximations of the empirically measured distributions via exponential distributions.

the upper one compared to `ILS-BILS` and `Restart`. Although there were some minor differences in the median and spread between the `ILS-BILS`, `ILS-WBLS` and `Restart` runs, these differences do not seem significant. The `Hybrid`, `Delta` and `Portfolio` ranges were comparatively short, in particular the `Portfolio`, with several outliers. The boxplot and outliers for `Portfolio` are so close that they cannot be distinguished visually. This indicates a consistently significant performance compared to the other algorithms. The performance of `Hybrid` was good as well, and better than that of `Delta`.

From the scatter plot of the 32 units test S8, it can be noticed that solutions are skewed to the higher evaluations side of the plot except for the `Hybrid` and `Portfolio` algorithms. It seems that the other algorithms can not find good solutions with low number of evaluations compare to the `Hybrid` and `Portfolio`. This is understandable as the 32 units system is larger and more constrained than the 21 units system. The `Hybrid` managed to find several solutions in lower number of evaluation compare to the `Portfolio` algorithm.

The bottom left plot shows that in terms of evaluations, the `Hybrid` and `Portfolio` algorithms have almost the same average of evaluations while the other algorithms average evaluations are close. However, the performance of algorithms in term of SSR obtained are different except for the `ILS-BILS` and `Restart` where the averaged SSR

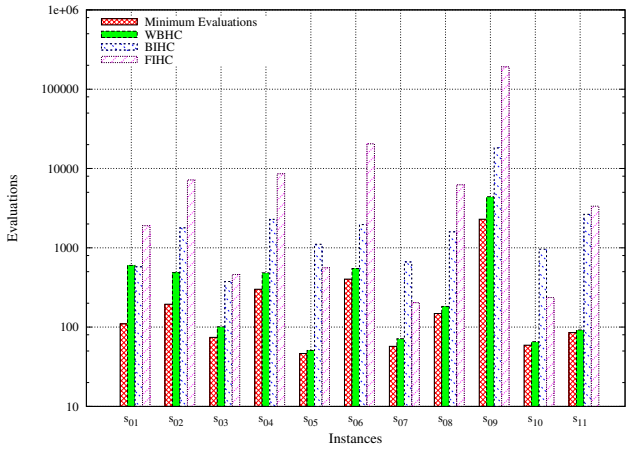


Fig. S6: The number of evaluations (averaged over 100 runs) required ILS algorithms with BETTER acceptance criterion, INSERT(3) perturbation and different local search algorithms to guide the search process to the feasible search space area. The Minimum Evaluations is the lowest evaluations required for all instances optimized by all algorithms.

values are almost identical. The `Portfolio` obtained by far the best solutions followed by `Hybrid` algorithm. Just like its performance on the 21 units system, the `ILS-WBLS` performed better than the `ILS-BILS` algorithm although the computational cost required is similar to `ILS-BILS`.

The boxplots of the `ILS-BILS`, `ILS-WBLS` and `Restart` algorithms show smaller variations than was the case for the 21-unit system, with fewer outliers. The range of solutions found by the `ILS-BILS` and `Restart` algorithms were similar, with close median values. The boxplot of the `ILS-WBLS` algorithm shows a lower range of SSR values and a lower median, indicating better performance. `Delta` performed slightly better than `ILS-WBLS`, with similar IQR and close median values. The performance of the `Hybrid` and `Portfolio` algorithms was more consistent on this test problem than on the 21-unit problem. There were no outliers for `Portfolio` and few for `Hybrid`. `Portfolio` was the algorithm which performed best of all in terms of boxplot range, position and median.

S3. Statistical analysis

Prior to choosing the appropriate test to validated these outcomes, it was necessary to check the normality of distribution of the results. The Kolmogorov-Smirnov test of normality after the Lilliefors significance correction [45] was chosen over other tests because it is commonly applied and is considered conservative. The test rejects the normality hypothesis for most of the algorithm results for the 21 units problem and some algorithms for the other problem. In this case, it was more practical to use a non-parametric test to examine the significance of the statistical results, as the normality assumption had been violated by some of the algorithms.

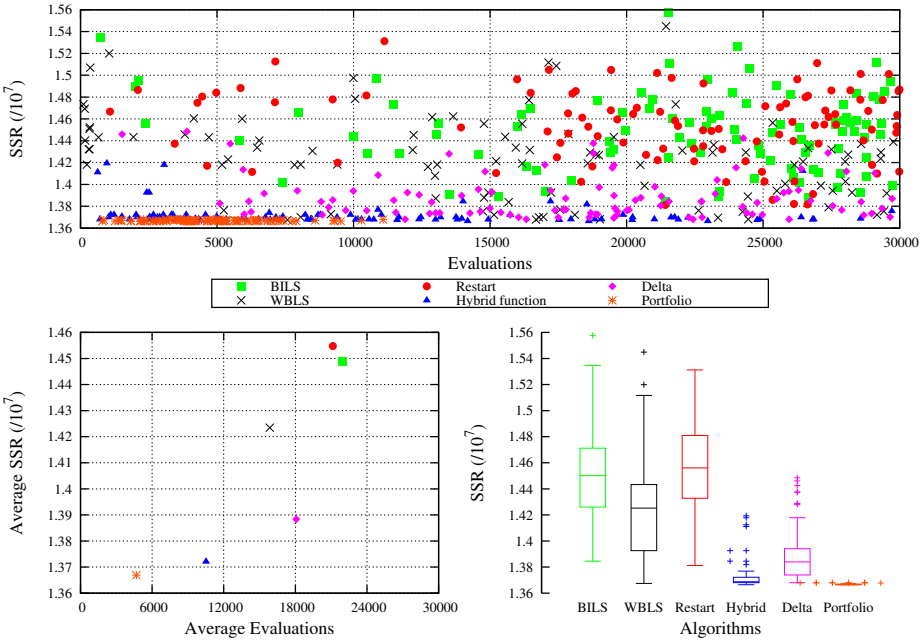


Fig. S7: The performance of the ILS algorithms on the '21 Units System' is presented as a scatter plot (top) of the runs, evaluations against SSR averaged over all runs (bottom left) and the boxplots (bottom right). BILS and WBLS stands for the algorithms `ILS-BILS` and `ILS-WBLS` respectively.

Table S2: The comparison of selected pairs of algorithms where each algorithm has executed 100 runs. Each test was performed at a significance level of $\alpha = 0.01$ after applying the Bonferroni adjustment. The statistics show the direction of the difference as mean or median.

21-Unit System					
Pair of algorithms	**Test**	**Statistic**		***P-value***	**Significance**
BILS and WBLS	Mann-Whitney	Median	14502209/14252268	1.465E-06	Significant
BILS and Restart	T-test	Mean	14487741/14547422	0.612	Not significant
BILS and Delta	Mann-Whitney	Median	14502209/13839697	9.060E-27	Highly significant
BILS and Hybrid	Mann-Whitney	Median	14502209/13687191	3.263E-33	Highly significant
Portfolio and Hybrid	Mann-Whitney	Median	13664879/13687191	2.739E-28	Highly significant
32-Unit System					
Pair of algorithms	**Test**	**Statistic**		***P-value***	**Significance**
BILS and WBLS	T-test	Mean	34083826/33912796	1.210E-23	Highly significant
BILS and Restart	Mann-Whitney	Median	34066316/34041033	0.651	Not significant
BILS and Delta	T-test	Mean	34083826/33867678	9.990E-36	Highly significant
BILS and Hybrid	Mann-Whitney	Median	34066316/33753789	1.303E-32	Highly significant
Portfolio and Hybrid	T-test	Mean	33666498/33769502	2.807E-09	Significant

The Kruskal-Wallis test is a non-parametric method of testing the null hypothesis that all populations are identical against the alternative that there is at least one population which differs from the others. We used the test here to validate the following hypotheses:

H_0: The algorithms have a similar performance,
H_1: At least one of the algorithms tends to achieve better results than the others.

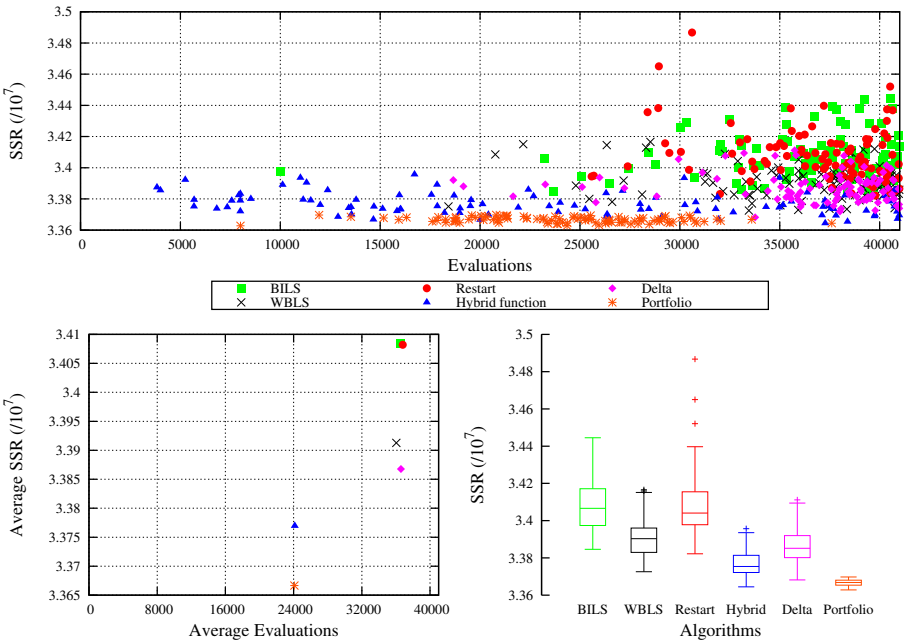


Fig. S8: The performance of the ILS algorithms on the '32 Units System' is presented as a scatter plot (top) of the runs, evaluations against SSR averaged over all runs (bottom left) and the boxplots (bottom right). BILS and WBLS stands for the algorithms `ILS-BILS` and `ILS-WBLS` respectively.

The value of the significance level obtained by the test (p-*value* = 3.757E-98 and 8.277E-95 for the 21- and 32-unit systems respectively) was much less than $\alpha = 0.05$, leading to the clear rejection of H_0 and revealing a statistically highly significant difference in the SSR values across the algorithms for both problems. Therefore, the alternative hypothesis H_1 was accepted. However, this result does not indicate which of the algorithms were statistically significantly different from one another. A multiple comparison method can be used to perform tests between pairs of algorithms. The method required to test a pair of algorithms depends on their normality. The t-test can be used for testing pairs where results for both are normally distributed, while the non-parametric Mann-Whitney test should be used otherwise. Table S2 shows the probability values for different tests and the direction of the difference as mean or median for the t-test and Mann-Whitney tests respectively.

The pair tests were performed at a significance level of $\alpha = 0.01$ after applying the Bonferroni adjustment. Any probability value (p) higher than the α value indicates that the result was not significant and that there was no statistical difference in the performance of the pair of algorithms under study.

Acknowledgements Ahmad Almakhlafi gratefully acknowledges support by the Saudi Arabian Cultural Bureau under grant number SACB-D184.

References

1. A. Almakhlafi. http://www.cs.man.ac.uk/∼almakhla/ILS.html, 2014.
2. A. Almakhlafi and J. Knowles. Benchmarks for maintenance scheduling problems in power generation. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.
3. A. Almakhlafi and J. Knowles. Systematic construction of algorithm portfolios for a maintenance scheduling problem. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 245–252. IEEE, 2013.
4. Y. Z. Arajy and S. Abdullah. Hybrid variable neighbourhood search algorithm for attribute reduction in rough set theory. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 1015–1020. IEEE, 2010.
5. S. Arueti and D. Okrent. A knowledge-based prototype for optimization of preventive maintenance scheduling. *Reliability Engineering & System Safety*, 30(1-3):93–114, 1990.
6. A. Bar-Noy, R. Bhatia, J.S. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete Algorithms*, pages 11–20. Society for Industrial and Applied Mathematics, 1998.
7. H. Bashir and R. Neville. A hybrid evolutionary computation algorithm for global optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2700–2707, 2012.
8. S. Baskar, P. Subbaraj, M. V. C. Rao, and S. Tamilselvi. Genetic algorithms solution to generator maintenance scheduling with modified genetic operators. *Generation, Transmission and Distribution, IEE Proceedings-*, 150(1):56–60, 2003.
9. L. Bianchi, J. Knowles, and N. Bowler. Local search for the probabilistic traveling salesman problem: correction to the 2-p-opt and 1-shift algorithms. *European Journal of Operational Research*, 162(1):206–219, 2005.
10. M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo. Estimation-based local search for stochastic combinatorial optimization using delta evaluations: a case study on the probabilistic traveling salesman problem. *INFORMS Journal on Computing*, 20(4):644–658, 2008.
11. G. Budai, D. Huisman, and R. Dekker. Scheduling preventive railway maintenance activities. *Journal of the Operational Research Society*, 57:1035–1044(10), 2 September 2006.
12. G. Budai, D. Huisman, and R. Dekker. Scheduling preventive railway maintenance activities. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 5, pages 4171 – 4176 vol.5, 10-13 2004.
13. E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
14. E. K. Burke and A. J. Smith. A multi-stage approach for the thermal generator maintenance scheduling problem. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2. IEEE, 1999.
15. D. Chattopadhyay. A game theoretic model for strategic maintenance and dispatch decisions. *Power Systems, IEEE Transactions on*, 19(4):2014–2021, 2004.
16. W. R. Christiaanse and A. H. Palmer. A technique for the automated scheduling of the maintenance of generating facilities. *Power Apparatus and Systems, IEEE Transactions on*, PAS-91(1):137–144, 1972.
17. R. K. Congram, C. N. Potts, and S. L. Van De Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
18. K. P. Dahal, C. J. Aldridge, and J. R. McDonald. Generator maintenance scheduling using a genetic algorithm with a fuzzy evaluation function. *Fuzzy Sets and Systems*, 102(1):21–29, 1999.
19. K. P. Dahal and N. Chakpitak. Generator maintenance scheduling in power systems using metaheuristic-based hybrid approaches. *Electric Power Systems Research*, 77(7):771–779, 2007.
20. K. P. Dahal and J. R. McDonald. Generator maintenance scheduling of electric power systems using genetic algorithms with integer representation. In *IEE conference publication*, pages 456–461. Institution of Electrical Engineers, 1997.
21. K. P. Dahal, J. R. McDonald, and G. M. Burt. Modern heuristic techniques for scheduling generator maintenance in power systems. *Transactions of the Institute of Measurement and Control*, 22(2):179–194, 2000.

22. X. Dong, H. Huang, and P. Chen. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, 36(5):1664–1669, 2009.
23. J. F. Dopazo and H. M. Merrill. Optimal generator maintenance scheduling using integer programming. *Power Apparatus and Systems, IEEE Transactions on*, 94(5):1537–1545, 1975.
24. G. T. Egan, T. S. Dillon, and K. Morsztyn. An experimental method of determination of optimal maintenance schedules in power systems using the branch-and-bound technique. *Systems, Man and Cybernetics, IEEE Transactions on*, 6(8):538–547, 1976.
25. I. El-Amin, S. Duffuaa, and M. Abbas. A tabu search algorithm for maintenance scheduling of generating units. *Electric Power Systems Research*, 54(2):91–99, 2000.
26. M. Y. El-Sharkh and A. A. El-Keib. Maintenance scheduling of generation and transmission systems using fuzzy evolutionary programming. *Power Systems, IEEE Transactions on*, 18(2):862–866, 2003.
27. L. Fanjul-Peyro and R. Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.
28. W. K. Foong. *Ant colony optimisation for power plant maintenance scheduling*. PhD thesis, School of Civil and Environmental Engineering, The University of Adelaide, 2007. http://digital.library.adelaide.edu.au/dspace/handle/2440/47786.
29. W. K. Foong, H. R. Maier, and A. R. Simpson. Ant colony optimization for power plant maintenance scheduling optimization. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 249–256. ACM, 2005.
30. D. Frost and R. Dechter. Maintenance scheduling problems as benchmarks for constraint algorithms. *Annals of Mathematics and Artificial Intelligence*, 26(1):149–170, 1999.
31. C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 190–197. Morgan Kaufmann Publishers Inc., 1997.
32. C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.
33. J. Gruhl. Electric generation production scheduling using a quasi-optimal sequential technique. Technical report, MIT Energy Lab, 1973.
34. J. Gruhl. Electric power unit commitment scheduling using a dynamically evolving mixed integer program. Technical report, MIT Energy Lab, 1973.
35. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.
36. H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, Darmstadt University of Technology, 1999.
37. B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51, 1997.
38. A. Khanlari, K. Mohammadi, and B. Sohrabi. Prioritizing equipments for preventive maintenance (pm) activities using fuzzy rules. *Computers & Industrial Engineering*, 54(2):169–184, 2008.
39. B. Kralj and R. Petrović. Optimal preventive maintenance scheduling of thermal generating units in power systems–a survey of problem formulations and solution methods. *European Journal of Operational Research*, 35(1):1–15, 1988.
40. B. Kralj and R. Petrovic. A multiobjective optimization approach to thermal generating units maintenance scheduling. *European Journal of Operational Research*, 84(2):481–493, 1995.
41. B. Kralj and N. Rajakovic. Multiobjective programming in power system optimization: new approach to generator maintenance scheduling. *International Journal of Electrical Power & Energy Systems*, 16(4):211–220, 1994.
42. B. Laurent and J. K. Hao. Iterated local search for the multiple depot vehicle scheduling problem. *Computers & Industrial Engineering*, 57(1):277–286, 2009.
43. D. Lei. Multi-objective production scheduling: a survey. *The International Journal of Advanced Manufacturing Technology*, 43(9):926–938, 2009.
44. R.-C. Leou and S.-A. Yih. A flexible unit maintenance scheduling using fuzzy 0-1 integer programming. In *Power Engineering Society Summer Meeting, 2000. IEEE*, volume 4, pages 2551–2555. IEEE, 2000.
45. H. W. Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

46. C. E. Lin, C. J. Huang, C. L. Huang, C. C. Liang, and S. Y. Lee. An expert system for generator maintenance scheduling using operation index. *Power Systems, IEEE Transactions on*, 7(3):1141–1148, 1992.
47. H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. *Handbook of Metaheuristics*, page 321, 2003.
48. H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.
49. P. Merz and J. Huhse. An iterated local search approach for finding provably good solutions for very large tsp instances. In *Parallel Problem Solving from Nature–PPSN X*, pages 929–939. Springer, 2008.
50. T. Messelis, S. Haspeslagh, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux Conference on Artificial Intelligence*, volume 21, pages 177–183, 2009.
51. D. K. Mohanta, P. K. Sadhu, and R. Chakrabarti. Deterministic and stochastic approach for safety and reliability optimization of captive power plant maintenance scheduling using ga/sa-based hybrid techniques: A comparison of results. *Reliability Engineering & System Safety*, 92(2):187–199, 2007.
52. F. Peng, K. Tang, G. Chen, and X. Yao. Population-based algorithm portfolios for numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 14(5):782–800, 2010.
53. G. Quan, G. W. Greenwood, D. Liu, and S. Hu. Searching for multiobjective preventive maintenance schedules: Combining preferences with evolutionary algorithms. *European Journal of Operational Research*, 177(3):1969–1984, 2007.
54. G. Raidl, J. Puchinger, and C. Blum. Metaheuristic hybrids. *Handbook of Metaheuristics*, pages 469–496, 2010.
55. L. Ren, C. Duhamel, and A. Quilliot. A hybrid ILS/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. In *International Workshop on Green Supply Chain–GSC 2012*, 2012.
56. T. Satoh and K. Nara. Maintenance scheduling by using simulated annealing method. *Power Systems, IEEE Transactions on*, 6(2):850–857, 1991.
57. E. B. Schlünz. Decision support for generator maintenance scheduling in the energy sector. Master's thesis, Stellenbosch: Stellenbosch University, 2011.
58. E. B. Schlünz and J. H. van Vuuren. An investigation into the effectiveness of simulated annealing as a solution approach for the generator maintenance scheduling problem. *International Journal of Electrical Power & Energy Systems*, 53:166–174, 2013.
59. N. Shukla, Y. Dashora, M. K. Tiwari, F. T. S. Chan, and T. C. Wong. Introducing algorithm portfolios to a class of vehicle routing and scheduling problem. In *Proceedings of The 2nd International Conference on Operations and Supply Chain Management*, pages 1015–1026. Novotel Siam Square Hotel, Bangkok, Thailand., 2007.
60. B. Sigl, M. Golub, and V. Mornar. Solving timetable scheduling problem using genetic algorithms. In *Proc. of the 25th int. conf. on information technology interfaces*, pages 519–524, 2003.
61. B. Silverthorn and R. Miikkulainen. Latent class models for algorithm portfolio methods. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 167–172, 2010.
62. T. Stützle. *Local Search Algorithms for Combinatorial Problems*. PhD thesis, Darmstadt University of Technology, 1998.
63. T. Stützle. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Infix Sankt Augustin, Germany, 1999.
64. T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006.
65. T. Stützle and H. Hoos. Analyzing the run-time behaviour of iterated local search for the tsp. In C.C. Ribeiro, editor, *Proceedings of the Third Metaheuristics International Conference MIC 99*, pages 1–18, Universidade Catolica do Rio de Janeiro, Angra dos Reis, 1999.
66. P. M. Subcommittee. Ieee reliability rest system. *Power Apparatus and Systems, IEEE Transactions on*, 6:2047–2054, 1979.
67. A. Subramanian, L. A. F. Cabral, and L. S. Ochi. An efficient ils heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Relatório Técnico, Universidade Federal Fluminense, disponível em http://www. ic. uff. br/˜ satoru/index. php*, 2008.
68. K. Suresh and N. Kumarappan. Combined genetic algorithm and simulated annealing for preventive unit maintenance scheduling in power system. In *Power Engineering Society General Meeting, 2006*, pages 1–5. IEEE, 2006.

69. N. M. Tabari, M. Pirmoradian, and S. B. Hassanpour. Revenue based maintenance scheduling of a genco in restructured power systems. In *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 International Conference on*, pages 155–158. IEEE, 2008.
70. R. Tonić and M. Rakić. Annual preventive maintenance scheduling for thermal units in an electric power system. *The Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043*, 20(2), 2010.
71. J. A. Vrugt and B. A. Robinson. Improved evolutionary optimization from genetically adaptive multimethod search. *Proceedings of the National Academy of Sciences*, 104(3):708, 2007.
72. J. A. Vrugt, B. A. Robinson, and J. M. Hyman. Self-adaptive multimethod search for global optimization in real-parameter spaces. *Evolutionary Computation, IEEE Transactions on*, 13(2):243–259, 2009.
73. Y. Wang and E. Handschin. A new genetic algorithm for preventive unit maintenance scheduling of power systems. *International Journal of Electrical Power & Energy Systems*, 22(5):343–348, 2000.
74. T. Wireman. *Benchmarking best practices in maintenance management*. Industrial Press Inc., 2nd edition edition, 2010.
75. Z. Yamayee, K. Sidenblad, and M. Yoshimura. A computationally efficient optimal maintenance scheduling method. *IEEE Trans. Power Appar. Syst.;(United States)*, 102(2), 1983.
76. Z. A. Yamayee. Maintenance scheduling: description, literature survey, and interface with overall operations scheduling. *Power Apparatus and Systems, IEEE Transactions on*, 101(8):2770–2779, 1982.
77. Y. Yare and G. K. Venayagamoorthy. Optimal maintenance scheduling of generators using multiple swarms-mdpso framework. *Engineering Applications of Artificial Intelligence*, 23(6):895–910, 2010.
78. H. H. Zurn and V. H. Quintana. Several objective criteria for optimal generator preventive maintenance scheduling. *Power Apparatus and Systems, IEEE Transactions on*, 96(3):984–992, 1977.

MISTA 2015

A general technique for improving the complexity of FPTAS for scheduling problems

Eugene Levner • Amir Elalouf

1 Introduction

An approximation algorithm is called a *fully polynomial time approximation scheme* (FPTAS) if it satisfies two conditions: (i) for any instance of an optimization problem and a parameter $\varepsilon > 0$, it finds an approximate solution Z^A that satisfies $Z^A \le Z^*(1+\varepsilon)$ or $Z^A \ge Z^*(1-\varepsilon)$, respectively, for the minimization and maximization problem, where Z^* is the optimal solution; (ii) its running time is polynomial in problem size and $1/\varepsilon$. An FPTAS is the strongest polynomial approximation method that can be obtained for a NP-hard problems (if $P \neq NP$).

We describe a technique for improving the complexity of known FPTAS for scheduling problems. The suggested approach continues and extends the computational procedure developed by Ergun et al. [3] for the restricted shortest path problem and later extended by the authors of the current paper for several scheduling and routing problems ([1], [2], [9]).

We suggest the improved FPTAS to consist of three stages as follows:

Stage A: Find a preliminary "rough" lower bound *LB* and an upper bounds *UB* on the optimal solution such that $UB/LB \le n$.

Stage B: Using the preliminary, "rough" bounds of Stage 1, find the improved lower and upper bounds on the optimal solution, such that

$$UB/LB \le 2. \quad (1)$$

Stage C: Using the improved bounds of Stage B, run a standard (known) FPTAS for deriving an ε-approximation solution to a considered problem.

In this work we will focus on improving the complexity of stage *B*.

2 Previous work

Over the past decades, there have evolved a number of different approaches for designing fast approximation schemes for scheduling and routing problems; we refer to [6], [10] and [12], for excellent reviews of definitions and recent results.

For various scheduling and routing problems, the rough preliminary bounds, such that $UB/LB \le n$, have long been known to researchers as folklore. In past three decades, for several scheduling/routing problems, improved bounds satisfying (1) have been obtained in polynomial time. Table 1 presents a summary of corresponding complexity results. The second (logarithm-containing) term in the formulas of Table 1 displays the worst-case complexity of constructing the improved bounds, where *n* stands for the number of jobs in scheduling problems. Actually, the improved bounds for the problems in Table 1 have been obtained as follows: If a FPTAS for a problem runs in time bounded by a polynomial $\boldsymbol{p}(L, 1/\varepsilon)$ in problem size *L* and $1/\varepsilon$ then the improved bound is built in $O(\boldsymbol{p}(L) \log (UB/LB))$ time.

Eugene Levner
School of Economics, Ashkelon Academic College, Ashkelon, Israel
E-mail: elevner@acad.ash-college.ac.il

Amir Elalouf
Department of Management, Bar Ilan University, Ramat Gan, Israel
E-mail: amir.elalouf@biu.ac.il

Table 1. Complexity of the FPTAS for scheduling problems

Problem	Complexity	Source
Scheduling to minimize the number of late items	$O(n^2/\varepsilon + n^2 \log UB/LB)$	Gens & Levner [4]
Scheduling to minimize the total tardiness	$O(n^6/\varepsilon + n^6 \log UB/LB)$	Kovalyov [7]
Scheduling to minimize the number of late Items	$O(n^3/\varepsilon + n^3 \log UB/LB)$	Kovalyov [7]
Scheduling with set-ups to minimize the number of late jobs	$O(n^3/\varepsilon + n^3 \log UB/LB)$	Kovalyov [7]
Two-machine problem to maximize the weighted number of early jobs	$O((n^4/\varepsilon + n^4 \log UB/LB)$	Shabtay & Bensoussan [11]

A breakthrough in this line of research came in 2002, when Ergun, Sinha and Zhang [3] have proved that the ratio $UB/LB \leq 2$ for the restricted shortest path (RSP) problem can be obtained in $O(Nm)$ time, where N denotes the number of nodes and m the number of arcs. As a result, Hasin's FPTAS algorithm of $O(Nm/\varepsilon + Nm \log UB/LB)$ time [5] was improved to $O(Nm/\varepsilon)$.

The authors of the current work have continued this line of research for the knapsack and scheduling problems (see [1], [2] and [9]), and, in particular, they have answered positively the question posed in [4], whether a FPTAS can be built for the minimization version of the job-scheduling-problem-with-deadlines running in $O(n^2/\varepsilon)$.

In the next section, we develop an accelerating construction based on the Ergun's et al. procedure which allows to improve the FPTAS complexities for all the scheduling problems displayed in Table 1. Namely, all the expressions for the improved complexity have the form of $O(n^y/\varepsilon)$ (where y is an integer) while the term of the form $n^y \log UB/LB$ is deleted.

3 Improving the complexity of the FPTAS at stage B

For all scheduling problems of Table 1, Stage *B* is constructed with two building blocks, an approximate binary search procedure denoted by Test(w,ε) and a narrowing procedure denoted by NARROW, which uses Test(w, ε) as a sub-procedure. Test(w, ε) is a parametric dynamic programming algorithm that has the following property: Given positive parameters w and ε, and an objective function Z to be minimized, Test(w, ε) reports that for the optimal objective value Z^* either $Z^* \leq w$ holds or $Z^* \geq w(1-\varepsilon)$ holds. This type of algorithm was first suggested by Gens and Levner [4] for the scheduling problem with deadlines and later further studied by Hassin [5], Kovalyov [7], Ergun et al. [3], Levner et al. [8], among others, for improving the UB/LB ratio in various combinatorial problems.

For each problem in Table 1, the complexity of Test(w, ε) depends on ε and is $O(n^y/\varepsilon)$. This fact is proved along the same line as the proof for the scheduling problem with deadlines in [4]. Test(w, ε) with fixed ε values is repeatedly applied as a sub-procedure in the NARROW algorithm to make $UB/LB \leq 2$.

The idea of the NARROW algorithm is based on the dynamic approximate binary search that chooses a larger error value ε when the bounds UB and LB are far from each other and a smaller ε when the UB and LB get closer. The implementation of this idea is given in [3], and, also, can be found in [1], [2] and [9].

The complexity of the algorithm NARROW for the considered scheduling problems is $O(n^y)$ where parameter y is shown in the corresponding expressions in Table 1. The proof proceeds along the same line as that of Lemma 5 in Ergun et al. [3], and establishes that Test(w,ε) should run in the NARROW algorithm at most 7 times. From the latter fact it immediately follows that in all the expressions for the FPTAS complexity for the problems in Table 1, the second term, of the form $n^y \log UB/LB$ (where y is a fixed integer) is excessive, and we derive the main result:

Theorem 1. The FPTAS for all the scheduling problems in Table 1 have complexity $O(n^y/\varepsilon)$.

4 Sufficient conditions for the problems allowing for an improved FPTAS

The following theorem formulates sufficient conditions under which the set of combinatorial problems with the improved FPTAS can be extended. Let *P* be a general scheduling problem with *n* jobs for which the following conditions are assumed to hold:

(i) The optimal solution Z* can be found, e.g., by dynamic programing, in $O(n^y UB)$, where *UB* is the upper bound to Z*, and *y* is a fixed positive integer;

(ii) Initial upper bound UB and lower bound LB such that UB/LB<n can be found in at most $O(n^{y+1})$ time.

Then the following claim takes place:

Theorem 2. Under conditions (i) and (ii), the complexity of the FPTAS for a problem *P* is $O(n^{y+1}/\varepsilon)$.

This is based on a straightforward consideration of the three-stage FPTAS construction where at stage *B* the accelerating algorithm NARROW is being used. At each stage the time complexity is, clearly, at most $O(n^{y+1}/\varepsilon)$.

The proposed general technique can be extended according to Theorem 2 and used for finding the improved FPTAS for other combinatorial problems, such as inventory problems, multi-constrained and multimodal routing problem, and others. Finding more accurate conditions than those of Theorem 2 is a challenge for future research.

a.

References

[1] A. Elalouf, E. Levner, E.Cheng, Routing and dispatching of multiple mobile agents in integrated enterprises, International Journal of Production Economics, 145(1), pp. 96-106 (2013).

[2] A. Elalouf, E. Levner, H. Tang, An improved FPTAS for maximizing the weighted number of just-in-time jobs in a two-machine flow shop problems. Journal of Scheduling, 16(4), pp. 429-435 (2013)

[3] F. Ergun, R. Sinha, L. Zhang, An improved FPTAS for restricted shortest path, Information Processing Letters, 83, pp. 287-291 (2002).

[4] G.V. Gens, E.V. Levner, Fast approximation algorithm for job sequencing with deadlines, Discrete Applied Mathematics, 3, pp. 313-318 (1981).

[5] R. Hassin, Approximation schemes for the restricted shortest path problem, Mathematics of Operations Research, 17, pp. 36-42 (1992).

[6] I. Kacem, H. Kellerer, Y. Lanuel. Approximation algorithms for maximizing the weighted number of early jobs on a single machine with non-availability intervals. Journal of Combinatorial Optimization, doi:10.1007/s10878-013-9643-7 (2013).

[7] M.Y. Kovalyov, Improving the complexities of approximation algorithms for optimization problems, Operations Research Letters 17 85–87 (1995).

[8] E. Levner, A. Elalouf, An improved approximation algorithm for the ancient scheduling problem with deadlines, Control, Decision and Information Technologies, International Conference on, Metz, 3-5 November 2014, IEEE Publ. pp.113-116 (2014).

[9] E. Levner, A. Elalouf, T.C.E. Cheng, An improved FPTAS for mobile agent routing with time constraints, Journal of Universal Computer Science, 17, pp.1854-1862 (2011).

[10] P. Schuurman, G.J. Woeginger, Approximation Schemes - A Tutorial. Manuscript, available at http://www.win.tue.nl/~gwoegi/papers/ptas.pdf , 65pp. (2011).

[11] D. Shabtay, Y. Bensoussan. Maximizing the weighted number of just-in-time jobs in several two-machine scheduling systems. Journal of Scheduling 15 pp.39-47 (2012).

[12] G.J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing 12, pp 57-75 (2000).

MISTA 2015

Family Scheduling in Flow Shop Manufacturing Systems with Batch Availability

Liji Shen · Jatinder N. D. Gupta

1 Introduction

This paper addresses a batch scheduling problem in flow shop production systems, where jobs are grouped into *families* according to their technological similarities. The objective is to minimize makespan. In this context, family setup times (*major setup*) are usually defined as the change-over costs for the processing of jobs from different families [1]. In comparison, setup times between jobs in the same family (*minor setup*) are typically small and are commonly included in the processing time of a job. Moreover, we take into account *sequence dependency*. That is, the magnitude of setup times depends on the family of the current job as well as the family of its previous job.

Due to the time-consuming and costly setups, production efficiency can be improved by choosing a long run-length for each family, which leads to the batching consideration. *Batching* refers to the decision of whether or not to schedule similar jobs contiguously [5]. Therefore, a *batch* is a maximal subset of jobs which share a single setup and must be processed jointly. If a batch must contain an entire family, it is referred to as *group technology assumption* [2,6] (GTA). As a result of integrating batching decisions into the family scheduling model, advantages occur due to the reduced number of setups and therefore, higher machine utilization.

Furthermore, there are two variants depending on when jobs are dispatched. *Job availability*, as a traditional type of processing, assumes that a job becomes available once it is completed on a certain machine. *Batch availability*, on the other hand, requires that all jobs of a batch are not available for processing on a downstream stage until the entire batch completes. In such cases, jobs in a batch are processed sequentially so that the processing time of a batch is equal to the sum of the processing times of its jobs or operations. Applications arise when, for example, the jobs in a batch are placed on pallets, containers or boxes which can only be removed upon the completion

Liji Shen
Faculty of Business and Economics, Technische Universitaet Dresden, 01062 Dresden, Germany
E-mail: liji.shen@tu-dresden.de

Jatinder N. D. Gupta
College of Business Administration, University of Alabama in Huntsville, Huntsville, AL 35763, USA

of the last job [4]. Strongly motivated by the practical relevance, we consider batch availability in this study. In addition, *inconsistent batches* are allowed, which indicate different batch formations on different machines.

To examine problems with batch availability closely, Figure 1 presents three optimal solutions to the same flow shop problem subject to different assumptions. Given that the first family consists of jobs 1, 2 and 3, the first Gantt-diagram depicts a permutation schedule with GTA imposed; the next one is a permutation schedule without GTA. In comparison, the last solution drops GTA and adopts inconsistent batches, where different batches are formed for the first family on different machines. This schedule differs from traditional permutation flow shops, and is essentially a non-permutation schedule. Apparently, makespan is remarkably improved by applying inconsistent batches and violating GTA. In general, permutation schedule is not optimal for flow shop problem, if sequence dependent setup times are present [3]. More importantly, explicitly considering batching decisions is necessary when batch availability is required. The problem

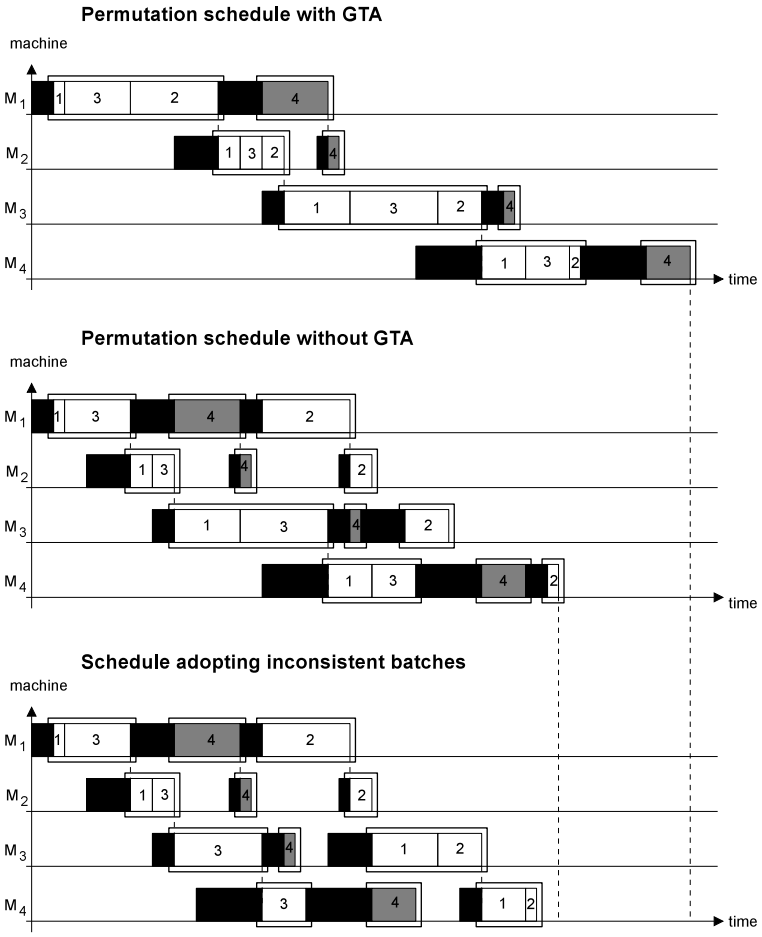


Fig. 1 Benefits of Batching with inconsistent batches

now is to find both batch composition and batch sequence for the flow shop system so that makespan is optimized. We focus on determining schedules with inconsistent batches on different machines. This, however, is complicated by the fact that batching and scheduling are inter-dependent. Although production efficiency is maximized by selecting large batches, this may cause delays to other families. Thus, these two decisions must be integrated in order to achieve a satisfactory compromise.

2 The tabu search algorithm

For solving the problem under study, we develop a tabu search algorithm with multiple neighbourhood functions, which are well adjusted to problem structures. Our neighbourhood functions primarily employ insertion-based moves. Note that they concern more operations and thus, provide thorough changes to the current schedule.

Besides utilizing specifically defined moves, particular attention is paid to setup times. In comparison to traditional scheduling problems, setup times play a crucial role in the family scheduling model since they represent a significant portion in the resulting makespan. From this point of view, it is essential to adjust the neighbourhood structure emphasizing family setup times.

First of all, we focus on critical operations as they define the resulting makespan. Moves involving these operations thus have the potential of immediately improving makespan. Furthermore, we propose neighbourhood functions using batch-based as well as extraction-based moves.

3 Computational results

The purpose of the experiments is to verify the advantages of batching first. In addition, we also conduct experiments examining various neighbourhood functions. Problem instances are randomly generated by using the common structure of benchmark instances for flow shop problems.

To the best of our knowledge, this specific scheduling problem has not been addressed in the literature so far. Since no comparable heuristic is currently available, we use modified CDS and NEH heuristics, and the state-of-the-art metaheuristic of [7] as references for comparison. Computational results not only confirm the remarkable benefits provided by batching, they also suggest improving production efficiency by dropping the group technology assumption.

For future research, more sophisticated algorithms for solving this problem are desirable, which would also represent a challenge to our approach.

References

1. Cheng, T.C.E., Lin, B.M.T., Toker, A.: Makespan minimization in the two-machine flowshop batch scheduling problem. Naval Research Logistics **47**, 128–144 (2000)
2. Gupta, J.N.D., Stafford, E.F.: Flowshop scheduling research after five decades. European Journal of Operational Research **169**, 699–711 (2006)
3. Gupta, J.N.D., Tunc, E.A.: Scheduling a two-stage hybrid flowshop with separable setup and removal times. European Journal of Operational Research **77**(3), 415–428 (1994)
4. Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: A review. European Journal of Operational Research **120**, 228–249 (2000)

5. Potts, C.N., Van Wassenhove, L.N.: Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. The Journal of the Operational Research Society **43**, 395–406 (1992)
6. Schaller, J.E., Gupta, J.N.D., Vakharia, A.J.: Scheduling a flowline manufacturing cell with sequence dependent family setup times. European Journal of Operational Research **125**, 324–339 (2000)
7. Shen, L., Gupta, J.N.D., Buscher, U.: Flow shop batching and scheduling with sequence-dependent setup times. Journal of Scheduling **17**, 353–370 (2014)

MISTA 2015

Three approaches to solving the problem of cosmonauts' training planning

S. Bronnikov · V. Gushchina · A. Lazarev · N. Morozov · A. Sologub · D. Yadrentsev

1 Introduction

We consider the problem of distribution operation's qualifications among cosmonauts. The proper preparation of cosmonauts is a long, expensive and sophisticated process. In order to maintain reliability of a flight, the members of a crew are obligated to be trained for different types of situations and operations, obtain required skills and

Sergei Bronnikov
Rocket and Space Corporation Energia after S.P. Korolev, 4A Lenin Street, Korolev, Moscow Region, 141070, Russian Federation;
E-mail: sbronnik@mail.ru

Varvara Gushchina
Lomonosov Moscow State University, Russian Federation;
V.A. Trapeznikov Institute of Control Science of Russian Academy of Sciences, Moscow, Russian Federation;
E-mail: vg@kvartasoft.ru

Alexander Lazarev
V.A. Trapeznikov Institute of Control Science of Russian Academy of Sciences, Moscow, Russian Federation;
Lomonosov Moscow State University, Moscow, Russian Federation;
Moscow Institute of Physics and Technology, Moscow Region, Russian Federation;
National Research University Higher School of Economics, Moscow, Russian Federation
E-mail: jobmath@mail.ru

Nikolai Morozov
Lomonosov Moscow State University, Russian Federation;
V.A. Trapeznikov Institute of Control Science of Russian Academy of Sciences, Moscow, Russian Federation;
E-mail: morozov.nikolay@physics.msu.ru

Alexander Sologub
Lomonosov Moscow State University, Russian Federation;
V.A. Trapeznikov Institute of Control Science of Russian Academy of Sciences, Moscow, Russian Federation;
E-mail: sologub10@gmail.com

Denis Yadrentsev
YU. A. Gagarin Research & Test Cosmonaut Training Center, Star City, 141160 Moscow Region, Russian Federation;
E-mail: D.Yadrentsev@gctc.ru

knowledge before launch. Hence YU. A. Gagarin Research & Test Cosmonaut Training Center (CTC) must plan and schedule list of trainings for every cosmonaut.

In this paper we consider crew of three cosmonauts. The cosmonauts are divided into two groups: experienced and inexperienced. In general three crew qualification levels are defined: User, Operator and Specialist. For every onboard complex a set of minimum qualifications is needed to operate safely. Consequently, the training program for each crewmember is assigned individually tailored to his or her set of tasks and predefined qualification levels.

Amount of time needed to train cosmonaut to certain qualification level on certain onboard complex are known.

Main goal is to distribute qualification levels between cosmonauts achieving minimal differences between the total time of the preparation of all team members.

In [1] is presented algorithms for the "exact" solution of the problem when there is only one qualification level. In [6] was shown that these algorithm have pseudo-polinimial time complexity. In the same article was also proved that problem is strongly $\mathcal{NP}$-hard for general m. Thus the algorithm in [1] cannot guarantee the optimal solution unless $\mathcal{P} \neq \mathcal{NP}$, although it may be used as good heuristic ([2]).

From the statement of the problem can be seen that its possible to use one of the multi-way partition problem's objective function. In [4] was shown that there were at least three of them: minimizing the largest subset sub, maximizing the smallest subset sum and minimizing the difference between the largest and smallest subset sums. In present work we will use the third of them.

2 Mathematical Problem

- The crew of K cosmonauts $\mathcal{K} = \{1, \ldots, K\}$. Each cosmonaut $k \in \mathcal{K}$ can be either experienced ($e_k = 1$) or inexperienced ($e_k = 0$) ;
- 3 qualification levels are known: Specialist ($q = 1$), Operator ($q = 1$) and User ($q = 1$);
- $\mathcal{J} = \{1, \ldots, J\}$ — set of onboard complexes;
- p_{jqe_k} — amount of time needed to train experienced ($e_k = 1$) or inexperienced ($e_k = 0$) cosmonaut $k \in \mathcal{K}$ to qualification level q on onboard complex $j \in \mathcal{J}$;
- n_{jq} — required amount of cosmonauts with qualification level q on onboard complex $j \in \mathcal{J}$;
- $x_{kjq} \in \{0, 1\}$ — boolean variables. $x_{kjq} = 1$ iff cosmonaut $k \in \mathcal{K}$ should has qualification level q on onboard complex $j \in \mathcal{J}$;
- $\tau_k = \sum\limits_{j=1}^{J} \sum\limits_{q=\{1,2,3\}} p_{jqe_k} x_{kjq}$ — total time of training of cosmonaut $k \in \mathcal{K}$.

The objective function:

$$(\max_{k \in \mathcal{K}} \tau_k - \min_{k \in \mathcal{K}} \tau_k) \to \min, \tag{1}$$

subject to:
for every qualification level required exist cosmonaut which should be trained for it

$$\sum_{k \in \mathcal{K}} x_{kjq} = n_{jq}, \tag{2}$$

cosmonaut can't have two different qualification levels on the same onboard complex

$$\sum_{q=\{1,2,3\}} x_{kjq} \leq 1. \tag{3}$$

By the local replacement ([3]) its possible to show that the problem is $\mathcal{NP}$-complete. Suppose that all cosmonauts have an equal training time for every job and $n_j = [1\,0\,0]$, $\forall j$. Then the problem reduces to multiway-partition problem that is actually $\mathcal{NP}$-complete. So the cosmonauts assignment problem is $\mathcal{NP}$-complete too.

3 Heuristic "Greedy" algorithm

This algorithm comprises two parts: at the first step we find any feasible solution, and at the second one we search for single qualification permutations that lead to a lower objective value. Initially all x_{kjq} are zero. At each step we are fixing an operation and assign a minimal qualification level to a cosmonaut with maximal training duration. Each qualification to be appointed until the constraint (2) is met. So this algorithm has a pseudopolynomial complexity, but however it has an arbitrary error. Accuracy may be improved if all qualifications are sorted with key $\max_l t_{jl}$ in nonincreasing order. As a result we obtain the first approximation of the problem and denote the objective value as $\delta_0 = \max_i \tau_i - \min_i \tau_i$

The next step is to find such a swap of qualification assignments that will lower an objective value. Without loss of generality we assume that the first cosmonaut has the maximal training time and the third one has the minimal training time. Let's denote $p_{ii'j} = |t_{jl}\{l : x_{ijl\neq 0}\} - t_{jl'}\{l' : x_{i'jl'\neq 0}\}|$ as a lag between assigned qualifications. So we search if there exists a lag that satisfies one of the conditions :

$$0 < p_{13j} < 2\delta_0 - (\tau_2 - \tau_3) \quad \text{or} \quad 0 < p_{13j} < \delta_0 + (\tau_2 - \tau_3) \quad \text{or} \quad 0 < p_{13j} < \delta_0,$$

$$0 < p_{12j} < \delta_0 - (\tau_2 - \tau_3) \quad \text{or} \quad 0 < p_{12j} < \delta_0 - (\tau_2 - \tau_3)/2,$$

$$0 < p_{23j} < (\tau_2 - \tau_3) \quad \text{or} \quad 0 < p_{23j} < \frac{\delta_0 + (\tau_2 - \tau_3)}{2}.$$

If one of those inequalities holds, we make a swap of corresponding qualifications.

4 Integer programming relaxation

Here we've used the branch&bound method, where at each brunch the relaxed linear problem was solved. More specifically we've replaced the boolean variables x_{kjq} by the set of variables that belong to the interval [0,1]. So $0 \leq x_{kql} \leq 1$, $\forall k, q, l$. Then the linear programming problem was solved by the interior point method, and then the new solution was searched with the brunch and bound methodology [5].

Data Sample	Experience	"Greedy" algorithm			Integer Programming relaxation		
		max	min	δ	max	min	δ
1	3 Inexperienced	887.8	886.75	1.05	888.05	887.75	0.3
	3 Experienced	570.5	569	1.5	570	569.5	0.5
	1 Exp 2 Inexp				697.25	695.25	2
	2 Exp 1 Inexp				616.5	612.75	3.75
2	3 Inexperienced	266.25	265	1.25	265.75	265.2	0.55
	3 Experienced	234.2	233	1.2	233.75	233.25	0.5
	1 Exp 2 Inexp				244.45	244	0.45
	2 Exp 1 Inexp				233.75	233.25	0.5
3	3 Inexperienced	661.25	657.5	3.75	659.85	659.75	0.1
	3 Experienced	353.5	353.05	0.45	353.5	353	0.5
	1 Exp 2 Inexp				484.05	481.75	2.3
	2 Exp 1 Inexp				393.5	392.5	1
4	3 Inexperienced	925.75	922.25	3.5	925	924.8	0.2
	3 Experienced	587	586.5	0.5	587	586.5	0.5
	1 Exp 2 Inexp				731.5	730.75	0.75
	2 Exp 1 Inexp				628.75	628	0.75

5 Experiments

The numerical experiments for two algorithms were carried out. All the initial data were given by YU. A. Gagarin Research & Test Cosmonaut Training Center. We have crew of three cosmonauts. Four possible cases of crew experience were considered. All obtained results are presented in the table. Here we fix the maximal and minimal total training times and also its difference (the unit of time equals to an hour).

The obtained results show that the second algorithm has the best accuracy, but even in the worst case, the error of each of the algorithms does not exceed 1 percent of the maximal training time.

Time of working of the first "Greedy" algorithm was less then 0.001 seconds in all cases. Time of working of the second algorithm was limited to 5 seconds: if the algorithm worked for 5 seconds, branching stopped. But for overwhelming majority of experiments processing time was less then 1 second.

6 Conclusion

In this paper the problem of volume planning was presented. The next step will be to set and to solve the calendar planning problem. Only after numerical experiments with calendar problem we could understand what algorithm is better: more accurate relaxation algorithm or quicker "Greedy".

References

1. Aggarwal, V., Tikekar, V., Hsu, L.: Bottleneck assignment problems under categorization. Computers & Oper. Res. **13**, 11–26 (1986)
2. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104) (2009). URL http://books.google.ru/books?id=nHIzbApLOr0C
3. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of books in the mathematical sciences. W. H. Freeman (1979). URL http://books.google.ru/books?id=fjxGAQAAIAAJ

4. Korf, R.E.: Objective functions for multi-way number partitioning. In: Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010 (2010). URL http://aaai.org/ocs/index.php/SOCS/SOCS10/paper/view/2098
5. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience, New York, NY, USA (1988)
6. Punnen, A.: On bottleneck assignment problems under categorization. Computers & Oper. Res. **31**, 151–154 (2004)

MISTA 2015

An exponential dynamic programming algorithm for the 3-machine flowshop scheduling problem to minimize the makespan

L. Shang · C. Lenté · M. Liedloff · V. T'Kindt

1 Introduction

Scheduling theory is a common interest for numerous researchers and it is concerned in many different areas, such as operations research, combinatorial optimization and industrial engineering. Most intriguing scheduling problems are $\mathcal{NP}-hard$, *i.e.* the best algorithm that we can expect, to find an optimal solution, is super polynomial (unless $\mathcal{P} = \mathcal{NP}$). To deal with $\mathcal{NP}-hard$ problems, the design of exponential algorithms has become a hot topic during the last decades. However, for $\mathcal{NP}-hard$ scheduling problems, few results are yet known. Some results for basic scheduling problems can be found in the survey of Lenté et al. [3], Cygan et al. [1] and Lenté et al. [4].

In this paper, we deal with the 3-machine flowshop scheduling problem, which has been proved as $\mathcal{NP}-hard$. The problem, denoted by $F3\|C_{max}$, is described as follows. Consider n jobs to be scheduled, each of which must be processed first on machine one, then on machine two and machine three in this order. Each machine can only have one job being processed at a time. The objective is to find an optimal job sequence that will minimize the makespan of all jobs. This problem can be trivially solved by enumerating all possible job permutations, which yields a $\mathcal{O}(n!)$ time algorithm.

The aim in this paper is to design a good exponential algorithm, that is, an algorithm in $\mathcal{O}^*(c^n)$ time with the constant c as small as possible. We make use of the notation $\mathcal{O}^*$ to express the worst-case complexity of an algorithm. An algorithm is in $\mathcal{O}^*(\alpha^n)$ time iff there exists a polynomial p such that the algorithm is in $\mathcal{O}(p(n) \cdot \alpha^n)$. With a *Dynamic Programming* approach, we have established an algorithm that solves the problem in $\mathcal{O}^*(3^n)$ time and space.

L. Shang, C. Lenté, V. T'Kindt
Université François-Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, 64 avenue Jean Portalis, 37200 Tours, France
E-mail: {shang,lente,tkindt}@univ-tours.fr

M. Liedloff
LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France
E-mail: mathieu.liedloff@univ-orleans.fr

2 Dynamic Programming

We note C_j the completion time on machine j of the last scheduled job, $j \in \{1, 2, 3\}$. Given a job permutation π, $C_j(\pi)$ returns the corresponding completion time of the machine j for this scheduling. We also define a binary operator "." which concatenates two permutations.

The proposed *Dynamic Programming* algorithm is based on the following theorem (Theorem 1).

Theorem 1 *Let S be a set of n jobs to be processed, $S' \subset S$. Given π and π' as two permutations of jobs from S', σ as a permutation of jobs from $S \setminus S'$, then we have :*

$$If \begin{cases} C_2(\pi) \leq C_2(\pi') \\ C_3(\pi) \leq C_3(\pi') \end{cases} \quad then \begin{cases} C_2(\pi.\sigma) \leq C_2(\pi'.\sigma) \\ C_3(\pi.\sigma) \leq C_3(\pi'.\sigma) \end{cases}$$

that is, the partial permutation π dominates π'.

The proof of Theorem 1 is straightforward, note that $C_1(\pi) = C_1(\pi')$ always holds.

The idea of *Dynamic Programming* is that, when trying to construct an optimal solution with partial solutions, only the partial solutions that are not dominated need to be considered. If we represent partial solutions as 2D points in the criteria space $< C_2, C_3 >$, then the non-dominated partial solutions can be seen as the *Pareto Front* of points (Fig. 1).

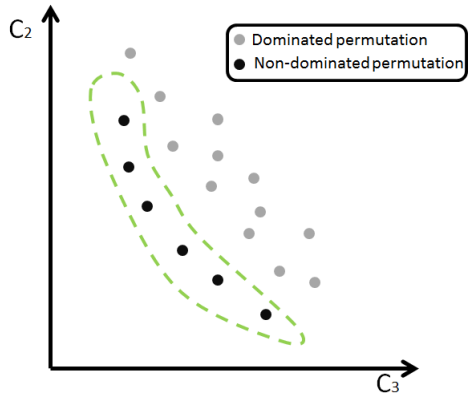


Figure 1 Partial solutions and *Pareto Front* from a given subset S' of jobs in criteria space

Now let us formulate the algorithm, starting with some definitions.

Definition 1 Given a set of permutations of a job set S, *Pareto Permutations* define a subset of permutations whose resulting criteria vector $< C_2, C_3 >$ is not dominated by the criteria vector of another permutation in the set. Let $MinPerm$ be a function that takes a set of job permutations as input and returns its *Pareto Permutations* set. $OptPerm(S)$ is the *Pareto Permutations* of all jobs from S and $OptPerm(S, m)$ $m \in \{1, ..., |OptPerm(S)|\}$ is the m-th permutation (the numeration is arbitrary).

Lemma 1 *For $|S'| = t$, the number of its Pareto Permutations $|OptPerm(S')|$ is $\mathcal{O}^*(2^t)$*

Lemma 2 *We note P_{ij}, a non-negative integer, the processing time of job i on machine j. If $\forall i, j\ P_{ij} \leq M$, then for a given job set S' of t jobs, the number of non-dominated criteria vectors $< C_2, C_3 >$, in Pareto Permutations, is $\mathcal{O}((t+1)M) = \mathcal{O}^*(M)$.*

Lemma 1 comes from the maximum number of possible critical paths ($n2^n$) that lead to different C_2 values. Lemma 2 is based on the consideration that the maximum value of C_2 will not exceed $(n+1)M$.

Theorem 2 *$OptPerm(S)$ can be computed by Dynamic Programming as follows :*

$$OptPerm(S) = \underset{\substack{k \in S; \\ m \in \{1,\ldots,|OptPerm(S \setminus \{k\})|\}}}{MinPerm} \Big(OptPerm(S \setminus k, m).\{k\} \Big)$$

Theorem 2 is directly based on Theorem 1.

According to Lemma 1, for a fixed S with t elements there are $\mathcal{O}^*(2^t)$ different $OptPerm(S \setminus k, m)$ to consider. The function $MinPerm$ employs an existing algorithm ([2]) for finding non-dominated criteria vectors $< C_2, C_3 >$, with a complexity of $\mathcal{O}(N \log N)$ for N vectors. Therefore, computing $OptPerm(S)$ from solved subproblems yields a complexity of $\mathcal{O}^*(2^t \log 2^t) = \mathcal{O}^*(2^t)$.

The algorithm traverses across all subsets of problem, the overall time complexity for calculating $OptPerm(S)$ is

$$\sum_{t=1}^{n} \binom{n}{t} \mathcal{O}^*(2^t) = \mathcal{O}^*(3^n)$$

Alternatively, based on Lemma 2, the time complexity can also be expressed as

$$\sum_{t=1}^{n} \binom{n}{t} \mathcal{O}^*(M) = \mathcal{O}^*(M2^n)$$

C_{max} can be calculated trivially from $OptPerm(S)$ in $\mathcal{O}^*(2^n)$ (or $\mathcal{O}^*(M)$) time, which does not change the established complexity. The space complexity is also $\mathcal{O}^*(3^n)$ (or $\mathcal{O}^*(M2^n)$) considering the storage of all necessary *Pareto Permutations*.

3 Conclusion and Perspectives

The algorithm is being computationally evaluated while we are still trying to achieve further theoretical improvements. Several questions were naturally raised.

Are there really $\mathcal{O}^*(2^t)$ points in the Pareto Front in the worst case ? This number is calculated by considering the number of possible critical paths that have different C_2 values. However, with the natural constraints of $F3\|C_{max}$, it is not clear to construct an instance for which all these paths are *critical*.

Can we improve the space complexity of the algorithm ? An initial experiment showed that the space requirement of the algorithm might be more critical than the time requirement. So this is also our consideration for further improvements.

Finally we are also thinking about generalising this algorithm to $F_m\|C_{max}$.

Références

1. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O. : Scheduling partially ordered jobs faster than 2 n. Algorithmica **68**(3), 692–714 (2014)
2. Kung, H.T., Luccio, F., Preparata, F.P. : On finding the maxima of a set of vectors. J. acm **22**(4), 469–476 (1975)
3. Lenté, C., Liedloff, M., Soukhal, A., T'Kindt, V. : Exponential algorithms for scheduling problems (2014). URL https ://hal.archives-ouvertes.fr/hal-00944382
4. Lenté, C., Liedloff, M., Soukhal, A., T'Kindt, V. : On an extension of the *Sort & Search* method with application to scheduling theory. Theoretical Computer Science **511**, 13–22 (2013)

MISTA 2015

Nurse Rostering Problem: Tighter Upper Bound for Pricing Problem in Branch and Price Approach

Antonín Novák · Roman Václavík · Přemysl Šůcha · Zdeněk Hanzálek

1 Introduction

The Nurse Rostering Problem (NRP) is a well-known combinatorial problem, in general $\mathcal{NP}$-hard, dealing with the rostering of human resources in a hospital. The high quality of hospital rosters is very desirable since it rises many advantages for an employer, employees (nurses) and, last but not least, patients. Specifically, the balanced roster, i.e. a fair distribution of shifts among the available personnel in a simplified way, causes a bigger satisfaction of nurses and a better hospital budget mainly due to a good utilization of nurses' workloads. Altogether, it leads to an increase in a quality of overall health care.

In this paper, we focus on the exact method. Branch and Price (BaP) approach can be considered as one of the most promising exact methods [3,6]. However, it consists of several parts which are executed every time from scratch and are very time demanding. For example, the pricing problem being $\mathcal{NP}$-hard [7], which generates new rosters for nurses, takes about 90% of the entire algorithm runtime [6]. Since this process is repeated frequently it would be profitable to use an experience/knowledge from the previous iterations to speed up the process in the actual or future iterations.

Therefore, the main contribution of this paper is to apply a machine learning technique within the pricing problem. Namely, a regression is employed to tighten up the upper bound of the objective function of the pricing problem. The upper bound is used for better pruning of the search tree and, thus, less nodes have to be looked through.

2 Problem Statement

The nurse rostering problem [4] is parametrized by the number of nurses n, the number of days during the planning period d, the number of shifts s and the set of constraints. Then the solution to this problem is roster R which is a binary matrix such that $\forall i \in \{1 \dots n\}, \forall j \in \{1 \dots d\}, \forall k \in \{1 \dots s\}$ $R_{ijk} = 1$ iff shift k is assigned to nurse i on

A. Novák · R. Václavík · P. Šůcha · Z. Hanzálek
Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Prague, Czech Republic
E-mail: novakan9@fel.cvut.cz, vaclarom@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz

day j and 0 otherwise. The quality of roster R is given by objective function Z which is defined as $Z(R) = (\sum_{i=1}^{n} Z_i(R)) + C(R)$, where Z_i is the quality of the assignment related to nurse i and C is the cover penalty (=0 if all shifts are assigned). The quality of an assignment Z_i is given by the number of violations of soft constraints. The hard constraints do not contribute to the objective function Z since they determine whether the roster is feasible or not. Then, the goal of the NRP is to assign demanded shifts to nurses while the value of objective function Z is minimized and the roster is feasible.

The branch and price approach is a general framework for solving huge problems [1] which can be described in the following way. At the beginning, there is an original problem formulation described in the previous paragraph which has to be decomposed to the so called Master Problem (MP) and Pricing Problem (PP). After that, the Restricted Master Problem (RMP) is created in such a way that only few columns are considered. In our case, the column represents the roster of one employee and RMP decides whether the given column is assigned to a nurse or not. The dual prices from the solution of relaxed RMP are then used to find the new column within the so called pricing problem. The column has to have a negative reduced cost which means it will violate the current dual problem and therefore it can improve the objective value of RMP. If such a column is found it is added to the RMP and the relaxed RMP is solved again. On the other hand, if the column is not found the solution of RMP is checked whether it is integral or not. If yes, the solution is final and optimal. Otherwise, the branching has to be performed and, thus, the new relaxed RMPs, with fixed variables according to branching rules, have to be solved.

3 Regression for Tighter Upper Bound

As it can be seen from the outline of branch and price approach, the pricing problem is called very often. Around 90% of the whole algorithm runtime is spent only just in the pricing problem [3]. Moreover, each iteration of pricing problem solves the problem from scratch without considering already observed information from the previous iterations. Therefore, we propose a machine learning technique like a linear regression to tighten up the upper bound and, thus, to prune more nodes in the search tree.

Since the heterogeneous environment (i.e. various instances which means completely different data) is taken into account, the online learning is the best way how to tackle this issue [2]. Therefore, the learning framework (LF) is the following. At each iteration, LF observes new features values and has access to the previous features and their target values. Based on this information, LF outputs its new prediction of the upper bound which is used in the pricing problem. After the run of the pricing problem, LF obtains the current true value.

The values are predicted from the continuous interval and, thus, we face a regression problem. Arguably the most popular and one of the simplest methods to solve the problem is Ordinary Least Squares Method (LSM) [8]. However, in our case, the LSM is not suitable since our loss function is skewed in a sense that the underestimating of the true value of Z_i is unwanted. Moreover, the predicted values cannot be expected to be equal to the target values since the small number of datapoints are available which means the LF is not likely to reach the highest possible accuracy. So, some reasonable small distance from target values should be used. Therefore, the criterion is a sum of skewed *epsilon* insensitivity loss functions. Additionally, the discounting function is employed since the contribution of older datapoints to the final loss function is less

important than the contribution of the recent observations. These requirements lead us to develop our custom criterion

$$\min_{\mathbf{w},\mathbf{r}} \sum_{i\in\mathcal{D}} c_i^+ r_i^+ + c_i^- \max\{r_i^- - \epsilon, 0\} \tag{1}$$

subject to

$$\forall i \in \mathcal{D}: \quad \mathbf{w}^T\mathbf{x}_i + r_i^+ - r_i^- = y_i \tag{2}$$

$$\forall i \in \mathcal{D}: \quad r_i^+, r_i^- \geq 0 \tag{3}$$

$$\mathbf{w} \in \mathbb{R}^n \tag{4}$$

where $c_i^{\pm}$ is a discounting constant for datapoint $i \in \mathcal{D}$ and the value of $r_i^{\pm}$ measures an error made by prediction for datapoint i. Then, the predictive hypothesis is given as $y = \mathbf{w}^T\mathbf{x}$. Finally, the whole LF is outlined as a pseudo-code in Algorithm 1.

Algorithm 1: Upper bound prediction for pricing problem

```
1  do
2  |  π ← current dual solution of restricted master problem (RMP)
3  |  ûb ← w^T φ(π)
4  |  y_il ← solve pricing problem with upper bound ûb
5  |  if pricing problem is infeasible then
6  |  |  y_il ← solve pricing problem without upper bound
7  |  end
8  |  add column y_il into the RMP
9  |  add new columns r_i^± into the prediction model
10 |  add constraint in form of (2)
11 |  w ← solution of (1)
12 while column with negative reduced cost exists;
```

If the pricing problem with the predicted bound is not able to find any column (line 5), it cannot guarantee that column with negative reduced cost does not exist. In that case, it is needed to run pricing solver without the tighter upper bound (line 6). Fortunately, it does not need to start completely from the scratch — partial solutions that were pruned in the previous stage due to the tighter upper bound can be stored and re-expanded in this stage.

Figure 1 depicts the tighter upper bound over the iterations of pricing problem. One can see that LF has some minor issues at the beginning since it does not have enough data to make very good predictions. However, from the 7th iteration, LF starts to copy the optimal value of upper bound very accurately.

4 Experimental results

To demonstrate the usability of our approach, *two pricing problem solvers were considered*: A* algorithm and ILP (Integer Linear Programming) solver. A* algorithm is based on the branch and bound method and our tight upper bound can be used directly for the comparison with lower bound. ILP solver uses the tight upper bound in a similar way, i.e. to prune nodes which do not lead to the optimal solution.

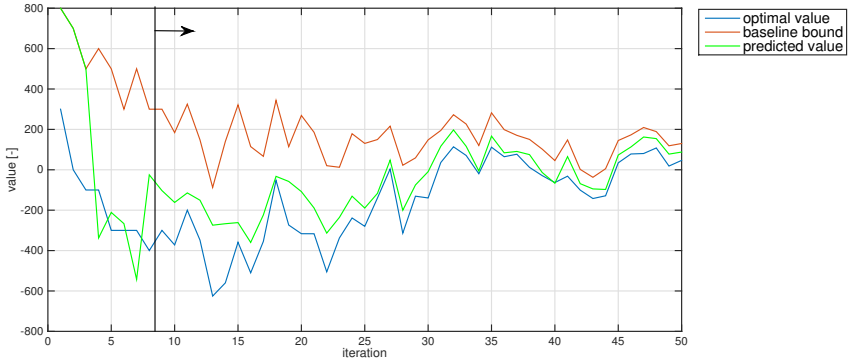


Fig. 1 Tighter upper bound for pricing problem on Millar instance.

Table 1 shows the preliminary results of the impact of our approach on the standard benchmark instances [5]. The numbers in the second column indicate the ratio of the visited nodes with to the visited nodes without applying our approach in the A* algorithm. Similarly, the third column represents the ratio of the algorithm runtime with to the algorithm runtime without applying our approach in the ILP solver.

Table 1 Impact of tighter upper bound on the performance. Values < 1 indicates a positive speedup.

Instance	# visited nodes ratio [-]	Runtime ratio [-]
Millar	0.95	-
WHPP	0.75	-
Valouxis	-	0.79
Azaiez	-	0.55
SINTEF	-	0.94
Average	0.85	0.76

So far, with using our approach, we are able to visit 15% less nodes, on average, in A* algorithm and to achieve 25% speedup, on average, for ILP solver. Moreover, the preliminary results for the whole branch and price method with the described approach and other improvements (e.g. subproblem skipping, symmetry breaking, etc.)[7] seem to be promising since they are better than the results reported in [3] on significant number of instances.

References

1. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.
2. L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.
3. E. K. Burke and T. Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71 – 81, 2014.

4. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *J. of Scheduling*, 7(6):441–499, November 2004.
5. T. Curtois. Employee scheduling benchmark data sets, September 2014.
6. B. Maenhout and M. Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13(1):77–93, 2010.
7. A. Novák, R. Václavík, P. Šůcha, and H. Hanzálek. Methods of the efficient state space search for the nurse rostering problem using branch-and-price approach. Technical report, Czech Technical University in Prague, 2015.
8. C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, Cambridge, Massachusetts, USA, 2006.

MISTA 2015

A framework of constructive heuristics for permutation-type scheduling problems

Victor Fernandez-Viagas · Manuel Dios · Paz Perez-Gonzalez · Jose M. Framinan

1 Introduction

Many scheduling problems of practical interest are known to be NP-hard, therefore the bulk of the effort in scheduling research has focused on developing *heuristics*, i.e. procedures aimed at providing good solutions for the problem but without guaranteeing its optimality.

Among heuristics, it is customary to distinguish between constructive heuristics and improvement heuristics. Broadly speaking, the former *construct* a solution from scratch, while the latter require one or several initial solution(s) to generate neighbour solutions and to (hopefully) obtain better solutions. Usually, such initial solutions are provided by one or more constructive heuristics.

While a great deal of effort has been carried out in improvement heuristics (including the development of an array of templates or *metaheuristics* that can be tailored for different combinatorial optimization problems), the development of constructive heuristics is a hand-craft -like process, conducted in many cases through trial-and-error. As a consequence, in many cases there is a poor understanding of how constructive heuristics are designed, and of which elements of their design influence their performance.

Our research aims at establishing a framework for constructive heuristics for permutation-type scheduling problems, which include many well-known scheduling problems (including single-machine and permutation flowshops, among others). Within this framework, we identify a general template that is adopted by the most efficient constructive heuristics. By doing so, we hope to provide further insights on how existing constructive heuristics for scheduling problems work, as well as to set guidelines for the development of new constructive heuristics. More specifically, in this contribution we present a framework or template of constructive heuristics for scheduling problems.

Victor Fernandez-Viagas, Manuel Dios, Paz Perez-Gonzalez, Jose M Framinan
Industrial Management, School of Engineering,
University of Seville. Camino de los Descubrimientos s/n. 41092 Seville, Spain
E-mail: {vfernandezviagas,mdios,pazperez,framinan}@us.es

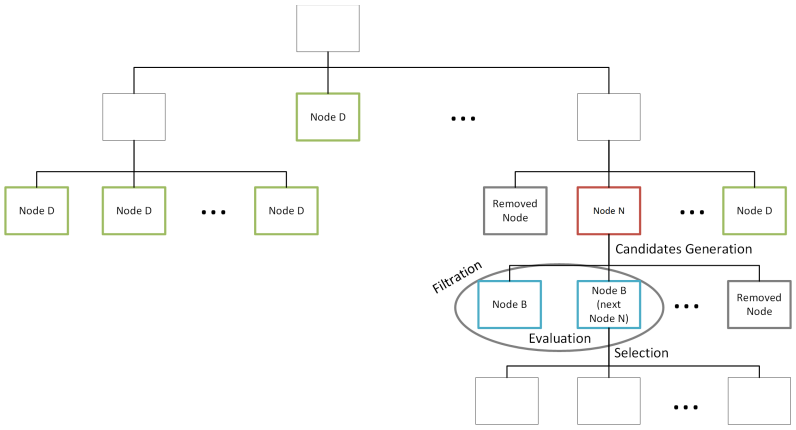


Fig. 1 Example

2 A framework for constructive heuristics

In this Section, we try to identify a common template which is adopted by many constructive heuristics in scheduling problems as well as by other traditional algorithms such as Beam Search, Branch and Bound, or A+. All these methods have in common that a sequence (a solution for the problem) is constructed by appending jobs one by one. In this process of construction, several decisions have to be taken, such as e.g.: which are the candidates to be chosen? how are the candidates evaluated? are discarded nodes considered in the following iterations? is it possible to recover removed nodes? ... All these decisions are classified in the proposed template. We identify five phases which are followed by most of the constructive heuristics in their iterations (see example in Figure 1):

Let us assume that the solution of the scheduling problem can be represented by a sequence of n jobs. Typically, a constructive algorithm iterates along $k = 1, \ldots, n$ stages. Let us denote N_i^k the i-th partial solution (node) in stage k ($i = 1, \ldots, x_k$, where x_k is a parameter of the algorithm, typically constant). Clearly, each partial solution is composed of k jobs, i.e. $|N_i^k| = k$. Let us denote by U_i^k the set of jobs not in N_i^k. Additionally, we have a list $\mathcal{D}^k$ of nodes discarded from selection.

1. For $i = 1$ to $i = n$:
   (a) **Candidates Generation**: In this phase, a set of $\mathcal{C}$ candidate nodes are generated:
      i. For each N_i^k, append each job in U_i^k at the end of the sequence. In this manner, $(k-1) \cdot x_k$ candidates are generated. Let us $\mathcal{B}_i^k$ be the set of the candidates obtained from node N_i^k, and
      ii. optionally, each node in $\mathcal{D}^k$ is added to the set.

$$\mathcal{C}^k := \cup_{i=1,\ldots,x} \mathcal{B}_i^k \cup \mathcal{D}^k$$

   (b) **Filter**: A threshold or filter is established in this phase, so some nodes in $\mathcal{C}^k$ are removed (removed nodes in the following) if they do not pass the threshold. The threshold may be based on Problem Properties (PP), Dominance Rules (DR), Comparison among bounds (CB) (e.g. lower bound lower than upper bound),

Heuristic	IO	Gen.	Filtr.	Evaluation		Sel.	Recov.	LS	Reference
				Prior.	TC				
Examples	X	$\mathcal{B}$	—	PM	FSU	$G[x]$	—	—	[Fernandez-Viagas and Framinan, 2015]
Based on NEH	—	$\mathcal{B}$	L	PM	—	—	I	—	E.g. [Nawaz et al., 1983]
FRBX	—	$\mathcal{B}$	L	PM	—	—	I_1, I_2	—	[Rad et al., 2009]
WY	—	$\mathcal{B}$	L	PM	—	$G[1]$	I	—	[Woo and Yim, 1998]
FL	—	$\mathcal{B}$	L	PM	—	—	I, T	—	[Framinan and Leisten, 2003]
FF(a)	X	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	—	[Fernandez-Viagas and Framinan, 2015]
LR(a)	X	$\mathcal{B}$	—	PM	FU	$L[1,x]$	—	—	[Liu and Reeves, 2001]
LR(a)-FPE(b)	X	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	T	[Liu and Reeves, 2001]
IC1	X	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	I	[Li et al., 2009]
IC2	X	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	I+T	[Li et al., 2009]
IC3	X	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	I+T	[Li et al., 2009]
Based on B&B	X	$\mathcal{B}+\mathcal{D}$	B, DR	PM	LB	$G[1]$	—	—	[Wang and Liu, 2013]
PW	X	$\mathcal{B}$	—	PM	FU	$G[1]$	—	—	[Pan and Wang, 2012]
MM, PF, wPF	X	$\mathcal{B}$	—	PM	—	$G[1]$	—	—	[Ronconi, 2004], [Pan and Wang, 2012]
Branch-and-bound	—	$\mathcal{B}+\mathcal{D}$	B	PM	LB	X	—	—	E.g. [Blazewicz et al., 2007]
Beam Search	—	$\mathcal{B}$	—	PM	—	$L[1,x]$	—	—	E.g. [Valente and Alves, 2008]
A*	—	$\mathcal{B}+\mathcal{D}$	—	P-PM	X	$G[1]$	—	—	E.g. [Russell and Norvig, 2007]
Greedy Search	—	$\mathcal{B}$	—	X		$G[1]$	—	—	E.g. [Russell and Norvig, 2007]

Table 1 Adaptation of several Constructive Heuristics of the PFSP to our template

list of jobs (L) (in each iteration the first job of the list is the only candidate and is removed from the list)... The idea behind this phase is to decrease the number of candidates which are evaluated in the next phase.

(c) **Evaluation**: The filtered candidates are evaluated. The nodes can be evaluated by means of: priority evaluation where the last node is considered with (PM) or without (P) consideration of the previous ones; total cost where the evaluation is performed by an estimation of final cost though upper bounds (UB), lower bounds (LB), forecast based on the unsequenced jobs (FU), forecast based on the sequenced and unsequenced jobs (FSU),...

(d) **Selection**: Once filtered nodes are evaluated, x_{k+1} nodes are selected according to either:
- a global approach ($G[x_{k+1}]$) where x_{k+1} nodes are selected among all nodes in $\mathcal{C}^k$, or
- a local approach ($L[y_i, x_k]$) where the best y_i nodes of each $\mathcal{B}_i^k$ (with $i \in [1, \ldots, x_k]$) for iteration k are selected ($\sum y_i = x_{k+1}$).

Then, the $\mathcal{D}^{k+1}$ set of non selected nodes, that may be eventually employed in the Candidates Generation of the next iteration, is updated.

(e) **Recovery**: This phase tries to recover nodes which are potentially good nodes but were removed and/or discarded from previous iterations. The recover elements are typically chosen based on insertion (I) or interchange (T).

Several examples of constructive heuristics for the Permutation Flowshop Scheduling problem are shown in Table 1. Last four rows shown some traditional algorithms under the proposed templates. Note that first column indicates the procedure to determine the initial exploration nodes and 9th column shows if some local search methods are applied after the iterated procedure.

Due to the limitations of the extended abstract, the following aspects will be discussed in the conference:

- Discuss the contribution of the different elements identified within this framework, and
- Illustrate the discussion using some examples from well-known scheduling problems which clearly improve the state-of-the-art algorithms.

References

[Blazewicz et al., 2007] Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Weglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. Springer.

[Fernandez-Viagas and Framinan, 2015] Fernandez-Viagas, V. and Framinan, J. M. (2015). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers and Operations Research*, 53(0):68 – 80.

[Framinan and Leisten, 2003] Framinan, J. and Leisten, R. (2003). An efficient constructive heuristic for flowtime minimisation in permutation flowshops. *OMEGA, The International Journal of Management Science*, 31:311–317.

[Li et al., 2009] Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155–164.

[Liu and Reeves, 2001] Liu, J. and Reeves, C. (2001). Constructive and composite heuristic solutions to the $P||\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452.

[Nawaz et al., 1983] Nawaz, M., Enscore Jr., E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

[Pan and Wang, 2012] Pan, Q.-K. and Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, 40(2):218–229.

[Rad et al., 2009] Rad, S. F., Ruiz, R., and Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, 37(2):331–345.

[Ronconi, 2004] Ronconi, D. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, 87(1):39–48.

[Russell and Norvig, 2007] Russell, S. and Norvig, P. (2007). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[Valente and Alves, 2008] Valente, J. and Alves, R. (2008). Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Computers and Operations Research*, 35(7):2388–2405.

[Wang and Liu, 2013] Wang, S. and Liu, M. (2013). A heuristic method for two-stage hybrid flow shop with dedicated machines. *Computers and Operations Research*, 40(1):438–450.

[Woo and Yim, 1998] Woo, H.-S. and Yim, D.-S. (1998). A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers & Operations Research*, 25(3):175–182.

MISTA 2015

Manufacturing Scheduling Systems: What are they made of?

Manuel Dios · Victor Fernandez-Viagas · Paz Perez-Gonzalez · Jose M. Framinan

1 Introduction

Many companies use the so-called Decision Support Systems (DSSs) for manufacturing scheduling, i.e. computerised systems for handling scheduling of operations. These systems are built upon a robust body of scheduling research that has been carried out during the last 5 decades, involving different approaches from various research disciplines, methods and techniques for tackling the decision problems arising in manufacturing scheduling. In view of this rather large source of information at hand, it would be interesting to classify existing implementations of DSSs for manufacturing scheduling in order to investigate which of these techniques and approaches are embedded into these DSSs.

Therefore, the main goal of our paper is to analyse how the different case studies about DSSs in manufacturing scheduling published in literature face the problem of obtaining a schedule, i.e. which are the most common disciplines, approaches and techniques applied. To do so, 97 contributions of literature have been reviewed describing 84 different DSSs for manufacturing scheduling. Given that a widely recognised gap between research and practice in the scheduling field [MacCarthy and Liu, 1993, Framinan and Ruiz, 2010] has been claimed, our work may serve to identify the most application-oriented areas within the scheduling research, and to identify the techniques that –at least from a practical viewpoint– are more employed, together with the description of their main advantages and shortcomings.

2 Review Methodology

The procedure followed to select the DSSs to review can be split into two stages. First, a systematic review was developed for papers published from 2000 to 2014. We focused on this period as we are not so interested in older papers, given the processing and graphical capabilities of computers prior to that date. Nevertheless, in a second stage (described below) we also consider previous contributions in order not to miss any

Manuel Dios, Victor Fernandez-Viagas, Paz Perez-Gonzalez, Jose M. Framinan
Industrial Management, School of Engineering, University of Seville
E-mail: {mdios,vfernandezviagas,pazperez,framinan}@us.es

relevant contribution. For the systematic review we used the SCOPUS search engine by Elsevier, given that the majority of relevant journals and conference proceedings are indexed. We launched a set of different search queries, such as "(manufacturing OR production) AND scheduling AND (DSS OR "Decision Support System")". Due to the heterogeneity and ambiguity of the results, not all of them were suitable for our study. Therefore, we followed a three-step procedure to filter the results.

- Title. First we rejected those works whose title was not relevant for our study.
- Abstract. The abstracts of those works that seemed to be relevant were carefully read and those that did not focused on the topic under study were excluded from the review.
- Full Document. We analysed in depth those works that still remained as relevant and obtained the final set of contributions for the review.

In a second stage we extended the resulting set of contributions by adding the relevant references cited by the set of contributions in the first stage, in this case without discriminating referenced dated prior to 2000. To filter the resulting contributions, we adopted the same three-step procedure as explained above. Moreover, we included some book chapters that were not considered in previous stage, but that were listed in the references.

After completing this process, a set of 97 works dealing with the implementation of DSS in manufacturing scheduling in a period covering from 1988 and 2014 were obtained. The full reference list is available on the URL in Section 4.

3 Classification Framework

In this section we briefly describe the framework utilized for classifying the DSSs. First, we classify the references according with the supporting disciplines or body of knowledge. Furthermore, they are also classified according to the approach adopted to model the scheduling problem. Finally, the specific technique employed for solving the scheduling problem is also employed to classify the references. The categories employed for the classification are based on the "Design Platform for Planning, Scheduling and Control Systems" in [Monfared and Yang, 2007], but we also integrate the classification regarding the different decision models in [Suri, 1985] (also in [Blazewicz et al., 2001]), where authors distinguish between generative, i.e. those models looking for a "good enough" candidate solution, and evaluative decision models, i.e.those models searching within a given set of possible solutions.

In addition to the supporting disciplines, approaches and techniques applied in each system, we also highlight the variety of sectors where they have been deployed in practice.

3.1 Supporting Discipline

Within this category we classify the research field used for handling the scheduling problem. As it has been stated by different authors(e.g. [Wiers, 1997]), manufacturing scheduling has been addressed from different research communities, each of them with their benefits and shortcomings. Therefore, it is interesting to analyse how the problem is treated in the different DSSs. Here we consider "Operations Research" (OR) and

Decision Models	Modelling Approach	Supporting Discipline	Number
Generative Models	Exact Techniques	Operations Research	19
		Computer Science	8
	Heuristics	Operations Research	36
		Computer Science	12
	Neural Networks	Operations Research	1
		Computer Science	-
	Artificial Intelligence	Operations Research	20
		Computer Science	40
Evaluative Models	Simulation	Operations Research	11
		Computer Science	8
	Queuing Theory	Operations Research	2
		Computer Science	1
	Fuzzy Logic	Operations Research	-
		Computer Science	1

Table 1 Results of the Review (I).Supporting Disciplines, Approaches and Decision Models.

"Computer Science"(CS). According to the platform by [Monfared and Yang, 2007] we take out "Control Theory" from our framework as we are focusing on scheduling decisions and those systems within that discipline mainly concentrate on control (i.e. monitoring of the execution).

3.2 Modelling Approach

For this category we scatter those approaches commented in the previously mentioned platform, to get a more precise idea of how the reviewed DSSs face the problem of scheduling. Here we differentiate between "Exact Techniques"(ET), "Heuristics" (H), "Neural Networks" (NN), "Artificial Intelligence" (AI), "Simulation" (S), "Queuing Theory" (QT) and "Fuzzy Logic" (FL).

3.3 Technique

Finally, within each modelling approach, we classify the specific technique used for solving the scheduling problem. We can see the techniques used in our classification in Table 2.

4 Review of Manufacturing Scheduling DSSs

In Tables 1 and 2 we can see a summary of the results obtained from the review. Due to space problems, the complete review and comments about some of the contributions will be done in the conference. Nevertheless, the whole review can be consulted in `http://taylor.us.es/componentes/mdr/MISTA/Review_MISTA_2015.pdf`.

Finally, in Table 3 we can see the main sectors where these systems has been deployed. In the review, apart from systems specifically developed for a sector we found some prepared for different environments, these appear in Table 3 as "Generic Systems". Moreover, we also found some systems developed for a concrete case but that hadn't been evaluated yet. These systems are classified in the category "Theoretical Systems". Finally, as we identified many different sectors where these systems had been deployed, we grouped into "Other Sectors" those that didn't belong to any of the most typical ones.

Modelling Approach	Technique	Number
Exact Techniques	Mixed Integer Linear Programming (MILP)	12
	Branch & Bound (B&B)	2
Heuristics	Dispatching Rules (DR)	18
	Specific Heuristcs (SH)	42
	Simulated Annealing (SA)	3
	Genetic Algorithm (GA)	9
	Tabu Search (TS)	1
	Ant Colony (AC)	3
Neural Networks	Feed Forward NN (FF)	1
	Multi-Layered Perceptron (MLP)	1
Artificial Intelligence	Expert Systems (ES)	33
	Constraint Programming (CP)	24
	Case Based Reasoning (CBR)	1
	Multi Agent Systems (MAS)	6
Simulation	Discrete Event Simulation (DES)	15
Queuing Theory	Queuing Networks (QN)	1

Table 2 Results of the Review (II). Approaches and Techniques.

Sector	Number
Metallurgical Industry	12
Paper Industry	10
Electronics Industry	8
Aerospace Industry	5
Chemical Industry	4
Automotive Industry	3
Generic Systems	15
Theoretical Systems	8
Other Sectors	21

Table 3 Results of the Review (III). Main sectors of deployment.

References

[Blazewicz et al., 2001] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J. (2001). Computer integrated production scheduling. In *Scheduling Computer and Manufacturing Processes*, pages 421–468. Springer Berlin Heidelberg.

[Framinan and Ruiz, 2010] Framinan, J. M. and Ruiz, R. (2010). Architecture of manufacturing scheduling systems: Literature review and an integrated proposal. *European Journal of Operational Research*, 205(2):237 – 246.

[MacCarthy and Liu, 1993] MacCarthy, B. L. and Liu, J. (1993). Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1):59–79.

[Monfared and Yang, 2007] Monfared, M. and Yang, J. (2007). Design of integrated manufacturing planning, scheduling and control systems: a new framework for automation. *The International Journal of Advanced Manufacturing Technology*, 33(5-6):545–559.

[Suri, 1985] Suri, R. (1985). An overview of evaluative models for flexible manufacturing systems. *Annals of Operations Research*, 3(1):13–21.

[Wiers, 1997] Wiers, V. b. (1997). A review of the applicability of or and ai scheduling techniques in practice. *Omega*, 25(2):145–153.

MISTA 2015

How to Unload Bulk Carriers Quickly?

Mathematical Models to Identify Efficient Loading Patterns

Heiner Ackermann · Karl-Heinz Küfer · Neele Leithäuser · Andreas Meyer · Sebastian Velten

1 Introduction

Increasing globalization leads to a steady rise of the quantity of goods to be transported. The main mode of transportation for bulk materials (like coal or sand, but also grain) is transportation via ship. As central hubs, ports handling bulk materials play a crucial role in the underlying logistic networks. Well-balanced unloading, transportation and stockyard capacities are essential for the economic success of these terminals. Moreover, strategic decisions involve high investments and have to be well prepared.

In this context, simulation models are important tools to analyze the complex interactions between unloading, loading and (intermediate) storage. However, to obtain meaningful results for the complete system, the decision problems of the sub-systems have to be understood first.

In this paper we study one of these sub-systems namely the unloading process of a single ship carrying bulk materials. Given a fixed infrastructure of unloading facilities as well as a ship with a given loading pattern and amount of bulk material, our goal is to find unloading sequences which minimize the total unloading time.

The unloading facilities that are taken into account are cranes and conveyer belts. The cranes unload the bulk materials onto the conveyor belts. During this process several cranes can work in parallel, but two or more cranes can use one conveyor belt at the same time if and only if they unload the same type of bulk material. In addition, a set of safety rules as well as technical restrictions have to be respected to obtain feasible unloading sequences.

Bulk carriers are structured into different compartments called hatches or cargo holds. We consider bulk carriers with 5 to 7 hatches, but larger ships with more hatches will be operating in future as well (see [3]). In a hatch a single type of bulk material is stored, but different hatches can contain different types. Furthermore, a loading

Heiner Ackermann, Karl-Heinz Küfer, Neele Leithäuser, Sebastian Velten
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
E-mail: heiner.ackermann@itwm.fraunhofer.de, karl-heinz.kuefer@itwm.fraunhofer.de, neele.leithaeuser@itwm.fraunhofer.de, sebastian.velten@itwm.fraunhofer.de

Andreas Meyer
Interdisciplinary Center for Scientific Computing IWR, Heidelberg University, Germany
E-mail: andreas.meyer@iwr.uni-heidelberg.de

pattern describes the assignment of different product types to the hatches of a ship and, depending on the unloading facilities, may have significant influence on the minimal unloading time.

We propose and compare two mathematical models to optimally solve the problem. The first one is a mixed integer linear program, the second a constraint program.

On the one hand optimal unloading times are an important input for simulations of the overall logistic system of a port. The results of this work have been used in practice in exactly this way to given decision support during the design of a logistic system. On the other hand the mathematical models can be used to analyze and compare different loading patterns. This information can be used to obtain optimal policies for the operation of the port.

To the best of our knowledge models to optimize the unloading time of bulk carriers have only been studied once before (see [1] and references therein). In contrast to our work, in [1] heuristic solution approaches and mixed integer linear programs providing lower bounds are presented.

2 Problem Description

We consider a ship with a given set of hatches $\mathcal{H}$. Due to stability issues of the separating walls it is in general not possible to unload a hatch completely before its adjacent hatches are unloaded at least partly. To model this requirement we split each hatch in a set of levels $\mathcal{L}$. The number of levels is identical for each hatch (see Figure 1).

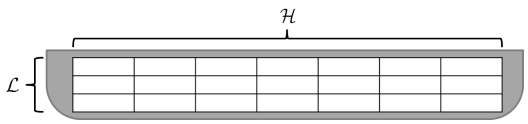


Fig. 1 Hatch/Level structure of a bulk carrier.

Next, let $\mathcal{P}$ be the set of different products and $\mathcal{P}(h)$ the product loaded in hatch $h \in \mathcal{H}$. These products are unloaded using a set of cranes $\mathcal{C}$ and a set of conveyor belts $\mathcal{B}$. Furthermore, let $\mathcal{C}(h)$ be the set of cranes that can access hatch $h \in \mathcal{H}$ and let $\mathcal{B}(c)$ the set of belts crane $c \in \mathcal{C}$ can unload on.

Given these index sets let $t^c_{(h,l)}$ be the time needed to unload level l in hatch h using crane c. These values may differ for different hatch/level combinations and cranes because of different fill levels and crane speeds. Moreover, unloading lower levels takes more time than unloading upper levels since wheel loaders have to be used to clear the corners of a hatch. In addition to the unloading time we have to consider the driving time of the cranes between the hatches, Δ_{h_1,h_2} $(h_1, h_2 \in \mathcal{H},\ h_1 \neq h_2)$, the time to clear a conveyor belt between two different products, Δ^B, and the time to put the wheel loader in a hatch using a crane, Δ^C.

The goal is to find an assignment for each hatch/level combination to a crane and belt as well as a schedule for each unloading facility so that the total unloading time is minimized. Cranes can unload only one hatch at a time. However, two or more cranes can be assigned to the same belt at the same time if they unload the same product and the capacity of the belt is not exceeded. Furthermore, adjacent hatches cannot be

unloaded at the same time and the unloading of hatch/level combination (h, l) has to be completed before the unloading of the following hatch/level combinations begins: $(h-1, l+1)$, $(h, l+1)$, $(h+1, l+1)$ (if $h-1$, $h+1$, $l+1$ exist and $l = 1$ being the highest level). In addition, the unloading process needs to be balanced. This means that the amount unloaded from the left side of the ship is not allowed to differ too much from the amount unloaded from the right side, and vice versa. We model this by requiring the difference between the sum of finished hatch/level combinations on the left and on the right side to be at most 1 at all time. Finally, the time for putting the wheel loader into a hatch before processing the last level of this hatch has to be respected.

3 Mathematical Models

To solve the optimization problem presented in the previous section, we propose two mathematical models. A feasible solution in one corresponds to a feasible solution in the other with the same objective value. They are summarized in this section.

Mixed Integer Linear Program: The mixed integer linear program is mainly based on two types of variables. On the one hand, variables representing the assignment of hatch/level combinations to cranes, respectively belts. On the other hand, variables modeling the sequence in which the hatch/level combinations assigned to a crane, respectively belt, are processed. Given these variables we use depended variables for the start and end times of the hatch/level unloading tasks. Moreover, we apply a set of variables representing the end time of the hatch/level unloading tasks in non-decreasing order. These variables are needed to model the balance requirements appropriately. To keep the number of variables as small as possible, all variables are defined so that a time index is avoided.

Constraint Program: The constraint program is developed to apply the automatic search of the commercial constraint solver *IBM ILOG CPLEX CP Optimizer* (see [2]). Therefore, we use the generic modeling components provided in this library. In this regard, hatch/level tasks are modelled with optional interval variables. For these variables the library contains generic constraints with which the required precedences and the assignments to cranes and belts can be incorporated. To obtain the assignment constraints, it is important that the interval variables are optional, which means that the solver decides whether an interval variable belongs to the final solution or not. Furthermore, variable step functions are used to model the restricted capacity of the unloading facilities and the balance requirement. At last, the aspect that more than one crane can be assigned to a belt if and only if they unload the same product is represented using predefined state functions.

4 First Results

Both mathematical formulations have been implemented on the basis of the class libraries *IBM ILOG Concert*, *CPLEX Optimizer* and *CPLEX CP Optimizer* (Version 12.6, see [2]). The C++ API of these libraries has been used and computations are executed on a *Intel(R) Core(TM) i5-2400* Processor with 3.10GHz and 4 cores.

The examples in Figure 2 show how the proposed models can be used to optimize loading patterns for given product distributions. A product distribution describes how many hatches for each product type are given. In both examples ships with 7 hatches

are considered and ship sizes as well as speed and capacity of unloading facilities are taken from a real port. For each product distribution there are 105 loading patterns so that in total 205 unloading times have been determined using the models of Section 3.

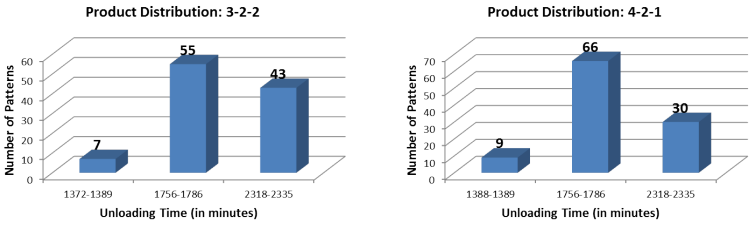


Fig. 2 Unloading times and clustering of loading patterns for two product distribution.

It becomes clear that the loading pattern may have a signification influence on the minimal unloading time. Note that in practice it will not always be possible to apply the pattern with the minimal unloading time. However, by solving the unloading problem for all possible loading patterns, clusters of loading patterns can be determined that should be avoided.

Comparing the solution times it becomes clear that the constraint programming approach outperforms the mixed integer linear formulation. The average solution times are faster by one order of magnitude, the maximal ones even more (see Table 1).

	CP			**MIP**		
Product Distribution	Min.	**Avg.**	Max.	Min.	**Avg.**	Max.
3 - 2 - 2	0.19	**3.61**	8.09	3.25	**38.44**	211.04
4 - 2 - 1	1.54	**3.23**	6.40	4.18	**39.63**	235.72

Table 1 Minimal, average and maximal solution times (in s).

5 Further Research

Further research activities in this area comprise the development of improved model formulations and the analysis of different objectives. Moreover, problem specific solution procedures may lead to faster solution times which is especially important if the unloading is part of a larger simulation.

References

1. K. Hazeghi and F. Weinberg, Optimization of the Unloading Strategy for Bulk Carriers, OR Spektrum, Volume 11, 101-110 (1989)
2. IBM ILOG CPLEX Optimization Studio, www.ibm.com/software/products/en/ibmilogcpleoptistud
3. Wikipedia, Bulk carrier — Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Bulk_carrier&oldid=645677638, (2015)

MISTA 2015

Scheduling with incompatible jobs: model and algorithms

Gustavo Campos Menezes · Geraldo Robson Mateus · Martín Gómez Ravetti

1 Introduction

In production systems, the integration of planning and scheduling problems is critical for the profitability of companies and the correct use of resources to meet deadlines. These problems are applicable in a broad range of sectors, such as the casting industry [5], the food industry [4], and cargo transportation in port terminals [9]. In this paper, a scheduling problem that appears as a subproblem in a decomposition algorithm in bulk cargo terminals is investigated.

Consider a scheduling problem with a set of jobs to be performed within a limited number of time period. For each job, we know its duration (processing time). The goal is to assign the start and end times for all of these jobs, considering incompatibility constraints. The set of constraints states that for some jobs pairs i and j their processing cannot overlap. Preemption is not allowed and the objective is to minimize the makespan. Hereafter, this scheduling problem is called the scheduling problem with incompatibility jobs (SPIJ). We propose two approaches to solve this problem: The first use a mathematical programming and optimization package. The second, based on GRASP (Greedy Randomized Adaptive Search Procedure). Instances for computational experiments were generated based on a real application.

The remainder of this article is structured as follows: Section 2 defines the integrated production planning and scheduling problem . Section 3 presents the scheduling problem, the mathematical model and algorithms. Section 4 is dedicated to computational results. Finally conclusions and future research directions.

Gustavo Campos Menezes
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brasil and Centro Federal de Educação Tecnológica de Minas Gerais, Brasil. E-mail: gcm@dcc.ufmg.br

Geraldo Robson Mateus
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brasil E-mail: mateus@dcc.ufmg.br

Martín Gómez Ravetti
Departamento de Engenharia de Produção, Universidade Federal de Minas Gerais, Brasil, E-mail: martin.ravetti@dep.ufmg.br

2 Product flow planning and scheduling problem

To better understand the planning and scheduling problem consider the following scenario. There exist a reception with a set of supply nodes (where the products are available for transportation), storage areas for such products (stockyards) and demand nodes. The products are available at the supply nodes, stored at the stockyards and delivered on certain dates to meet the demand. To perform the transportation of these products between the three set of nodes (supply, stockyard and demand), equipments with pre-defined capacities are used. They work together in a given sequence between nodes. This equipment sequence will be called a route. Figure 1 provides a schematic representation of the problem.

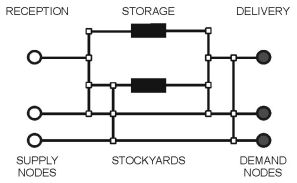


Fig. 1 Reception, storage and delivery system.

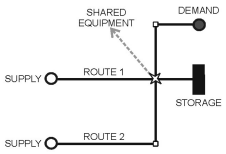


Fig. 2 Routes with shared equipments.

The amount of equipments is limited, they have flow capacities and the various routes may share equipments. Thus, if different products are assigned to routes that share equipment, these routes must be active at time intervals that do not overlap. Figure 2 shows a case where two routes (routes 1 and 2) share the same equipment.

The problem consists in defining the amount and destination of each product from supply nodes to stockyard or to demand nodes, or from stockyard to demand nodes, and simultaneously establishing a set of feasible routes (where there is no conflict regarding equipment allocation) to guarantee that the products are transported on schedule. In the remainder of this article, this problem will be called the Product Flow Planning and Scheduling Problem or $PFPSP$.

In this article, we investigate only the scheduling problem. It is considered that production planning has been solved and the production variables are available for production scheduling. The remaining sections only investigate the scheduling problem.

3 Scheduling with Incompatible Jobs

An example is used to clarify the definition of the SPIJ and to illustrate the jobs and constraints. Assume that the relaxed $PFPSP$ (disregarding scheduling constraints) was solved and that the variables (jobs) for a specific time period were extracted from the solution (Table 1).

In the first row of the Table 1, the variable x_2^1 (column *Variables*) represents that the product 2 should be carried by Route 1, and the total time to transport this product on Route 1 will be 4 hours (column *Values*). This variable corresponds to job or vertex A in the conflict graph (Figure 4). The remaining rows of the Table 1 have similar operations.

Table 1 Solution of the relaxed PFPSP (disregarding scheduling constraints)

jobs	Variables	Values	Routes
A	x_2^1	4	R_1
B	x_5^2	6	R_2
C	x_5^8	3	R_8
D	x_3^9	5	R_9
E	x_6^{10}	6	R_{10}
F	x_4^6	3	R_6

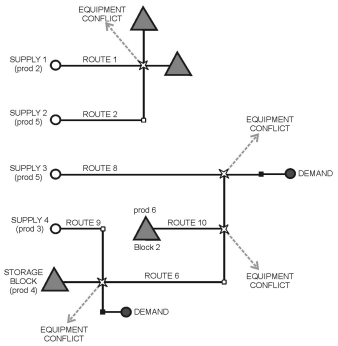


Fig. 3 Routes with conflicts

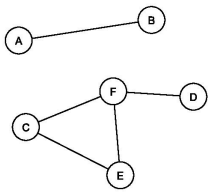


Fig. 4 Conflict graph

Figure 3 illustrates the conflict between routes. The routes 1 and 2 share equipment and therefore cannot operate simultaneously. The same is true for the routes numbered 6 and 9 and three other routes sharing equipment among themselves (routes 6, 8, and 10). In the conflict graph G (Figure 4), the vertices are the jobs (column jobs, table 1) and the edges are created from the conflicts presented in Figure 3. For instance, the vertices A and B must be connected in the conflict graph. A mathematical programming model that represents all SPIJ features is described in the next section.

3.1 SPIJ Formulation

To model the SPIJ consider the following parameters, sets and variables: Let p_i be a parameter related to the processing time of job i, parameter K a high-value constant, set J of jobs, and set E of all pairs of incompatible jobs (defining an edge in the conflict graph), and let the following variables: $Cmax$ the makespan, t_i the job i start time, and $u_{i,j}$ a binary variable that defines the precedence between pairs of incompatible jobs.

A SPIJ formulation can be given by:

$$\min \quad Cmax \tag{1}$$

subject to:

$$Cmax - (t_i + p_i) \geq 0, \forall i \in J \tag{2}$$

$$t_i - t_j - p_j \geq -Ku_{i,j}, \forall (i,j) \in E \tag{3}$$

$$t_j - t_i - p_i \geq -K(1 - u_{i,j}), \forall (i,j) \in E \tag{4}$$

$$t_i \geq 0, \forall i \in J \tag{5}$$

$$u_{i,j} \in (0,1) \geq 0, \forall (i,j) \in E \tag{6}$$

The objective function (1) minimizes the makespan. Constraints (2) set the value of Cmax, equations (3 and 4) guarantee the incompatible constraints to establish the order of jobs that share equipments. If $u_{i,j} = 1$ then (3) are redundant, and (4) ensures that the start time t_i of job i precedes t_j; if $u_{i,j} = 0$, t_j precedes t_i. Finally, (5) and (6) are domain constraints. Previous authors as [3], [7], [1] and [2] have shown that the SPIJ and its variations are NP-complete for various processing times, preemption or non-preemption and graph classes. The following section describes the Heuristic applied to solve the SPIJ.

3.1.1 SPIJ Heuristic

To efficiently find good solutions, a greedy randomized search procedure (GRASP) was implemented. GRASP is an iterative algorithm proposed by Feo and Resende [6] that basically consists of two phases, greedy construction and local search. The greedy construction phase builds a feasible solution s, whereas the local search investigates the neighborhood of s, searching for better solutions. The main phases of the heuristic are described in the following.

```
1: procedure GRASP-SCHEDULING()                                    ▷
2:     jobs = Production variables (relaxed PFPSP);
3:     for i=1 do MaxIteration
4:         Solution = GreedyRandomizedConstruction(Seed);
5:         Solution = LocalSearch(Solution);
6:         Solution = UpdateSolution(Solution, BestSolution);
7:     end for
8:     Return BestSolution;
9: end procedure
```

At each iteration of procedure *GreedyRandomizedConstruction*, the algorithm considers the jobs extracted from planning not yet scheduled, as the list of candidate elements. A greedy solution for the SPIJ is constructed as follows: Select randomly a job i from list of candidate elements at random. Next, define the lowest start time for the job, keeping already scheduled jobs that conflict with i without overlapping. Once all jobs are scheduled, a feasible solution for the SPIJ is provided.

The local search consists in exchanging the order of jobs found in the greedy construction phase. Two neighborhoods are explored. The first consists of exchanging the first and the last job of the sequence, then the second and the penultimate, etc. The second neighborhood explores exchanges between job pairs, i.e., the first and the second are exchanged with the last and the penultimate, following the same sequence of the first neighborhood.

4 Computational Experiments

As previously stated, the experiments are conducted considering the values obtained by solving only the production planning, were jobs and their processing times are extracted. Instances with 10,20, 50 and 100 jobs are analysed. The conflicts between jobs were generated based on analysis of a real case of a port terminal of iron ore (results are depict in Table 2).

The experiments were conducted using a computer with a 6-core Intel(R) Core(TM) i7 980 processor and 24 GB physical memory, running version 12.5 of the Cplex solver. For all instances, a time limit of 1 hour was set. The first and second columns of the table 2 contain the instance number and the number of jobs. Column *Cplex* provides the upper bound obtained with the CPLEX branch-and-cut algorithm. The *GAP* column provides the integrality gap (the value of objective function for the integer solution and its linear relaxation) obtained with the solver, $t(s)_1$ and $t(s)_2$ are the elapsed computational time to obtain the optimal or the best solution using the Cplex solver and the SPIJ heuristic respectively, expressed in seconds. Finally, the column *SPIJ Heuristic* provides the best solution obtained with the heuristic.

Table 2 Instances based on PFPSP problem

Instance	Jobs	Cplex	GAP	$t(s)_1$	SPIJ Heuristic	$t(s)_2$
1	10	23,19	0	0,08	23,48	3
2	10	26,95	0	0,76	27	2
3	10	15,89	0	0,35	**15,89**	2
4	10	17,37	0	0,15	**17,37**	6
5	10	18,17	0	0,26	**18,17**	4
6	20	12,05	0	0,89	13,37	32
7	20	15,34	0	0,19	16,61	12
8	20	13,26	0	0,56	13,95	18
9	20	16,91	0	0,77	17,86	17
10	20	14,07	0	10,78	16,94	248
11	50	15,15	0	20,78	15,58	350
12	50	26,69	9,10%	1h	31,96	227
13	50	50,43	27,87%	1h	62,37	424
14	50	40,51	2,47%	1h	53,88	208
15	50	41,21	9,10%	1h	42,36	586
16	100	30,29	41,38%	1h	**29,43**	3233
17	100	98,73	76,41%	1h	**85,9**	3564
18	100	67,2	65,00%	1h	**55,64**	2856
19	100	45,14	41,72%	1h	45,96	2575
20	100	86,48	67,66%	1h	**85,28**	2860

The optimization package managed to get optimal solutions for all instances with 10 and 20 jobs in Table 2. For instances with 50 jobs, the solver can find the optimal solution for only one instance (number 11). From 100 jobs, the solver appears to be ineffective and the gaps are all greater than 40%. Regarding the heuristic approach, it was possible to find the optimal solution for three instances (numbers 3, 4 and 5). In addition, for instances with 100 jobs, the heuristic found better solutions in less time. The exception was for instance number 19, where the makespan found by the solver was slightly lower (45.14 solver and 45.96 heuristic).

5 Conclusions

We consider in this work an integrated problem of planning and scheduling. We investigated the scheduling subproblem, which was named in this article as scheduling problem with incompatible jobs (SPIJ). We propose a mathematical programming formulation and a GRASP heuristic. Future works includes adaptation of these methods, as well as new methods. Efforts are also concentrated in an approach to deal in an integrated manner with the PFPSP problem.

Acknowledgements This research is supported by the following institutions: VALE, FAPEMIG and CNPq.

References

1. Mohamed Bendraouche and Mourad Boudhar. Scheduling jobs on identical machines with agreement graph. *Computers & Operations Research*, 39(2):382 – 390, 2012.
2. I. Blchliger and N. Zufferey. Multi-coloring and job-scheduling with assignment and incompatibility costs. *Annals of Operations Research*, 211(1):83–101, 2013.
3. Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55(3):219 – 232, 1994.
4. Damiao R. C. and R. Morabito. Scheduling of production and logistics operations of steam production systems in food industries". *J Oper Res Soc*, 65(12):1896–1904, Dec 2014.
5. Victor C. B. Camargo, Leandro Mattiolli, and Franklina M. B. Toledo. A knapsack problem as a tool to solve the production planning problem in small foundries. *Comput. Oper. Res.*, 39(1):86–92, January 2012.
6. Thomas A Feo and Mauricio G.C Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989.
7. Rajiv Gandhi, MagnsM. Halldrsson, Guy Kortsarz, and Hadas Shachnai. Improved bounds for sum multicoloring and scheduling dependent jobs with minsum criteria. In Giuseppe Persiano and Roberto Solis-Oba, editors, *Approximation and Online Algorithms*, volume 3351 of *Lecture Notes in Computer Science*, pages 68–82. Springer Berlin Heidelberg, 2005.
8. G. R. Mateus, M. G. Ravetti, M. C. Souza, and T. M. Valeriano. Capacitated lot sizing and sequence dependent setup scheduling: an iterative approach for integration. *Journal of Scheduling*, 13(3):245–259, 2010.
9. T. Robenek, N. Umang, and M. Bierlaire. A branch-and-price algorithm to solve the integrated berth allocation and yard assignment problem in bulk ports. *European Journal of Operational Research*, 235(2):399 – 411, 2014.

MISTA 2015

Scheduling preventive railway maintenance activities with resource constraints

Rita Macedo · Rachid Benmansour · Dragan Urošević · Abdelhakim Artiba · and Nenad Mladenović

1 Introduction

In this paper, we focus on the scheduling of preventive railway maintenance activities. The objective is to keep the railway infrastructure in good operating conditions at low costs, also taking into account the limited available resources in what concerns crew members. Equipments degrade with usage and age and a good preventive maintenance program can greatly reduce their unreliability in the sense that expectable failures can be anticipated. We propose a mixed integer programming formulation for the problem of scheduling preventive railway maintenance activities and a Variable Neighborhood Search (VNS) algorithm to solve large instances of the problem.

2 Problem description

There is a set of maintenance activities to perform during a planning horizon composed of $|T|$ periods. We consider two different kinds of maintenance activities: routine works with smaller durations and projects with larger durations. Routine works, such as inspections, cleaning operations and small repairs, are conducted on periodic basis, whereas projects are considered to be conducted once within the horizon (ballast

Rita Macedo
Institut de Recherche Technologique Railenium, F-59300 Famars, France
E-mail: rita.macedo@railenium.eu

Rachid Benmansour
University of Valenciennes and Hainaut Cambrésis, France
E-mail: Rachid.Benmansour@univ-valenciennes.fr

Dragan Urosevic
Mathematical Institute, Serbian Academy of Science and Arts, Belgrade, Yugoslavia
E-mail: draganu@turing.mi.sanu.ac.rs

Abdelhakim Artiba
University of Valenciennes and Hainaut Cambrésis, France
E-mail: abdelhakim.artiba@univ-valenciennes.fr

Nenad Mladenović
University of Valenciennes and Hainaut Cambrésis, France
E-mail: nenad.mladenovic@univ-valenciennes.fr

cleaning, rail grinding, etc.). More formally, the maintenance activities to be performed belong to one of two different sets: routine maintenance activities (R) or projects (P). The set of all activities is therefore defined as $A = R \cup P$. Routine maintenance activities are performed at a single period and are cyclic. Each activity $a \in R$ has a defined frequency $F^a = \left\lfloor \frac{|T|}{L^a} \right\rfloor$, where L^a denotes the duration of the interval between two consecutive repetitions of activity a. Projects are performed only once but have a duration D_p that is typically larger than one period. Each project $p \in P$ must begin at a period belonging to a defined interval T_p and its activities continue to be performed during the $D_p - 1$ subsequent periods. Whenever at least one activity is being performed at a period $t \in T$, the rail link must be closed and the system incurs into a holding cost c_t. This cost is independent of the number of activities being performed. This means that it is interesting to try to combine activities to be allocated to the same periods. However, there are some activities that are incompatible. A given set $C = \{(a_1, a_2)|$ activities $a_1, a_2 \in A$ can be performed at the same period$\}$ defines the compatible activities, i.e. the pairs of activities that can be performed at a same period. This problem was first described by Budai et al. [1]. We further consider an additional cost α_t in the problem. It is related to the fact that performing activities implies having a sufficiently large number of crew members and equipments. It is defined that the capacity of the company in what concerns these two resources is of performing at most θ activities at the same period. Each additional activity incurs into a penalization that may represent additional costs of outsourcing, paying extra hours to workers, renting additional machines, among others.

3 Mathematical formulation

Let x_t^a, y_t^p, m_t be binary variables defining respectively whether activity $a \in A$ is performed at period $t \in T$ or not, whether project $p \in P$ is started at period t or not and whether there is any activity being performed at instant $t \in T$ or not. The difference between the sum of allocated activities at a given period $t \in T$ and the maximum desirable number of activities θ is represented by the integer variable δ_t. This problem can be modeled with the following MIP formulation with assignment and positional date variables.

$$\min \sum_{t \in T} c_t m_t + \sum_{t \in T} \alpha_t \delta_t \tag{1}$$

$$s.t. \sum_{t=1}^{L^a} x_t^a = 1 \quad \forall a \in R \tag{2}$$

$$x_t^a = x_{t+qL^a}^a \quad \forall a \in R, t \in \{1, \ldots, L^a\}, 1 \leq q \leq F^a - 1 \tag{3}$$

$$\sum_{t \in T_p} y_t^p = 1 \quad \forall p \in P, \tag{4}$$

$$x_s^p \geq y_t^p \quad \forall p \in P, t \in T_p, s = t, \ldots, t + D_p - 1, \tag{5}$$

$$x_t^m + x_t^n \leq 1 \quad \forall t \in T, (m, n) \notin C, \tag{6}$$

$$m_t \geq x_t^a \quad \forall t \in T, \forall a \in A, \tag{7}$$

$$\delta_t \geq \sum_{a \in A} x_t^a - \theta \quad \forall t \in T, \tag{8}$$

$$\delta_t \geq 0 \quad \forall t \in T, \tag{9}$$

$$x_t^a, y_t^p, m_t \in \{0,1\} \quad \forall t \in T, \forall a \in A, p \in P \tag{10}$$

The objective function (1) minimizes the total operational costs, which comprise the track occupancy costs and the penalization costs of assigning more than θ activities to the same period. The unit penalization cost of period t is denoted by α_t. Every cyclic routine maintenance activity $a \in R$ must be performed at regular intervals of L^a periods (constraints (2) and (3)), and every project must begin at a period within its beginning interval T_p and continue through the $D_p - 1$ subsequent periods (constraints (4) and (5)). Constraints (6) ensure that only compatible activities are performed at the same period and constraints (7) guarantee that a track occupancy cost will be taken into account for every period with at least one allocated activity. Finally, constraints (8) and (9) define the number of activities that surpass θ for every time period.

4 Variable Neighborhood Search algorithm

We propose a Variable Neighborhood Search (VNS) [3] for this problem. A solution S is represented by $S = \{s_a | a \in A\}$, where $1 \leq s_a \leq L^a, \forall a \in R$, and $s_a \in T_a, \forall a \in P$. A solution may be infeasible if there is any pair of incompatible activities being performed in the same period. Thus, the objective function for a solution S is defined as $f(S) = \sum_{t \in T} c_t m_t(S) + \sum_{t \in T} \alpha_t \delta_t + P_f \times NbConf(S)$, where $m_t(S)$ is equal to 1 if there is any activity being performed at period t and equal to 0 otherwise, and where P_f is a penalty factor and $NbConf(S)$ is the number of pairs of conflicts in solution S. We define two different neighborhood structures within the solution space and start with an initial random solution. The local search implemented corresponds to a Variable Neighborhood Descent (VND) algorithm [2]. The two neighborhood structures are based on the removal of respectively one or two activities of the solution and their optimal reinsertion with a dynamic programming method. We conducted computational experiments on a set of 30 instances adapted from [1], with 15, 20 or 25 routine activities and a number of projects between 0 and 2. The results of VNS were compared with the ones obtained by solving model (1)-(10) with CPLEX. Both methods were run within a time limit of 1800 seconds. CPLEX solved 60% of the instances to optimality, within the time limit. For those instances, VNS always found the optimal solution. CPLEX did not prove the optimality of the solutions of 12 instances. For 6 of them, VNS found the same solution and for the other 6 it found better solutions. CPLEX took an average computational time of 1047.7 seconds, while VNS only took on average 1.2 seconds to reach the best found solutions.

References

1. Budai G, Huisman D, Dekker R (2006) Scheduling preventive railway maintenance activities. Journal of the Operational Research Society 57(9):1035–1044
2. Hansen P, Mladenović N, Pérez JAM (2010) Variable neighbourhood search: methods and applications. Annals of Operations Research 175(1):367–407
3. Mladenović N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24(11):1097–1100

MISTA 2015

Stability and Flexibility of Crew and Aircraft Schedules

Lucian Ionescu · Natalia Kliewer

1 Introduction

One main task in airline scheduling is the assignment of crews and aircraft for operating flights. Traditionally, the goal is to minimize the planned costs. However, airline operations frequently has to deal with disruptions like bad weather conditions, late passengers or technical issues which may lead to expensive recovery actions. This problem is addressed by robust resource scheduling when both planned cost effiency and robustness of schedules are considered as competing objectives. In the following we refer to this as robust efficiency.

The robustness can be improved by increasing the degree of stability or flexibility. Stability describes the ability of a system to work properly without changes and adjustments in case of disruptions. In our context, a resource schedule is stable if it remains feasible under changing operational environments. The main instrument for increasing stability is the incorporation of buffer times between tasks. In contrast, flexibility means the ability to be adapted to changing environments by manageable and mostly cost-neutral actions, e.g. swap opportunities for resources. In particular, a high degree of flexibility implies that feasibility can be restored at low cost in operations.

In this study we examine the potential of simultaneously considering stability and flexibility in resource scheduling. Therefore, different strategies for increasing the robustness are put in contrast with each other. Besides evaluating of the trade-off between cost-efficiency and robustness, the main question is if stability and flexbility also affect each other.

2 Framework for Robust Crew and Aircraft Scheduling

The presented study is based on an aircraft and crew scheduling framework, illustrated in Figure 1. For a given flight schedule, crew and aircraft schedules are generated in a Branch&Price&Cut approach concerning certain robustness indicators. The consideration of planned cost efficiency is determined by the cost structure for crews and

Department of Information Systems, Freie Universität Berlin, Garystr. 21, 14195 Berlin, Germany
E-mail: lucian.ionescu@fu-berlin.de · E-mail: natalia.kliewer@fu-berlin.de

aircraft usage. However, robustness has to be taken into account by considering delay occurrences and resulting propagation effects, see e.g. [Yen and Birge (2006)] and [Dück et al (2012)] for details. In this context we distinguish between primary and secondary delays. Primary delays are a result of exogenous disruptions that cannot be avoided by scheduling decisions. In succession, insufficient buffer times may cause propagation of delays on consecutive flights which is called secondary delay. Operational recovery may imply high additional costs and it is therefore desirable to already consider delay tolerance during scheduling.

In a previous study, primary delay prediction based on historical data has been assessed, see [Ionescu et al (2015)]. The resulting models and findings can be used as groundwork for a delay generator, enabling both resource scheduling and delay propagation simulation closer to reality.

Based on this, the evaluation of schedule robustness can be performed by an event-based simulation. Whenever delay occurs, it is either absorbed by buffer times or propagated to subsequent flights. However, this approach only considers the stability. The additional consideration of flexibility asks for recovery strategies in order to adapt the schedule to certain delay scenarios. The implemented recovery strategies are oriented towards [Shebalov and Klabjan (2006)] and [Ionescu and Kliewer (2011)].

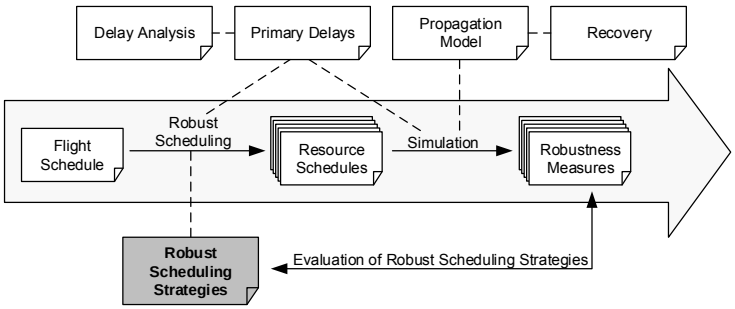


Fig. 1 Framework for Evaluating Robust Scheduling Strategies

The measurement of schedule robustness targets at the consideration of operating costs instead of planned cost. A practicable way to measure the robustness is the on-time performance (OTP) of flights in a simulation environment. The on-time performance can be described as the percentage of flights arriving or departing on-time for a given delay tolerance threshold, e.g. 15 minutes. However, exogenous primary delay cannot be influenced by scheduling decisions. Therefore, the relevant figure to consider is the secondary delay propagated due to insufficient buffer times between flights connected by the same resource. In consequence, a schedule A is more robust than schedule B if the amount of propagated secondary delay is less in A than in B.

3 Evaluating the Trade-off between Stability and Flexibility

In this study, the presented framework is used for evaluating the mutual impacts of stability and flexibility. Separate stable and flexible scheduling approaches provided by [Dück et al (2012)] and [Ionescu and Kliewer (2011)] are used as starting point. They are compared to each other concerning the robust efficiency of generated schedules, i.e. planned costs and on-time performance. Afterwards, stability and flexibility are put into relation within various advanced scheduling strategies.

As a first result, we evaluate to what extent an increasing degree of stability leads to a changing degree of flexibility and vice versa. Subsequently, it is evaluated if pareto-optimal resource schedules with the highest possible degree of both stability and flexibility can be generated that lead to a better trade-off between cost efficiency and robustness. The results give an insight on the potential of robust efficiency in resource scheduling.

Acknowledgements This research was supported by a grant from the German Research Foundation (DFG, Grant No. KL2152/3-1).

References

[Dück et al (2012)] Dück V, Ionescu L, Kliewer N, Suhl L, (2012) Increasing stability of crew and aircraft schedules. Transportation research part C: emerging technologies 20(1):47–61

[Ionescu and Kliewer (2011)] Ionescu L, Kliewer N, (2011) Increasing flexibility of airline crew schedules. Procedia-Social and Behavioral Sciences 20:1019–1028

[Ionescu et al (2015)] Ionescu L, Gwiggner C, Kliewer N, (2015) Data analysis of delays in airline resoure networks. Submitted to: BISE – Business & Information Systems Engineering

[Shebalov and Klabjan (2006)] Shebalov S, Klabjan D, (2006) Robust airline crew pairing: Move-up crews. Transportation Science 40(3):300–312

[Yen and Birge (2006)] Yen JW, Birge JR, (2006) A stochastic programming approach to the airline crew scheduling problem. Transportation Science 40(1):3–14

MISTA 2015

Scheduling Complex Job-Shops using Batch Oblivious Disjunctive Graphs

A Scheduling Approach for the Diffusion and Cleaning Area in Semiconductor Manufacturing

Sebastian Knopp · Stéphane Dauzère-Pérès · Claude Yugma

1 Introduction and Problem Description

In a competitive economical setting, semiconductor manufacturing is the core process in the production of today's everyday electronic devices. In this work, we cope with a scheduling problem arising in the diffusion and cleaning area of semiconductor manufacturing facilities. We are given a complex job-shop environment with batching machines. For each job in a given set, a fixed sequence of operations (route) must be performed on given machines. Each operation belongs to a family which specifies the machines that are qualified for its processing. Processing durations depend on selected machines. Some machines are capable of batching: They can process multiple operations of the same family at the same time as long as machine capacity restrictions are observed (p-batching). To each job is associated a ready date and a due date. We are interested in a combination of several objectives such as the total weighted tardiness or the number of processed operations in a given planning horizon.

Existing solution approaches for complex job-shop problems with batching machines rely on an adapted disjunctive graph representation where dedicated nodes represent batching decisions. We propose a novel approach where the graph is oblivious to batches: Batching decisions are taken dynamically during graph traversal and no dedicated batch nodes are introduced. This procedure cooperates with an overlying heuristic that alters ordering and assignment decisions. We show that for any given regular criterion, there exists a conjunctive graph such that our dynamic batching algorithm yields an optimal schedule. We see two advantages in this approach. First, batching decisions automatically adapt to assignment and ordering decisions. Second, the reduced graph complexity eases implementation efforts and facilitates the integration of more complex routing structures as presented in Knopp et al. (2014). We apply this dynamic batch evaluation algorithm within a heuristic based on the idea of greedy randomized adaptive search procedures (GRASP, see Feo and Resende (1995)).

Sebastian Knopp · Stéphane Dauzère-Pérès · Claude Yugma
École des Mines de Saint-Étienne, CMP Georges Charpak
Department of Manufacturing Sciences and Logistics
CNRS UMR 6158 LIMOS, F-13541, Gardanne, France
E-mail: {sebastian.knopp, dauzere-peres, yugma}@emse.fr

2 Related Work

The problem considered in this work is NP-hard since it is a generalization of both the NP-hard classical job-shop scheduling problem as well as the NP-hard single-machine scheduling problem with TWT objective (see Garey et al. (1976)). Consequently, heuristic methods are more appropriate when designing solution approaches for instances of industrial dimensions. An overview of existing approaches for scheduling in semiconductor manufacturing can be found in Mönch et al. (2011). Methods for scheduling problems with batching are reviewed in Mathirajan and Sivakumar (2006).

Starting with the work of Ovacik and Uzsoy (1997), several approaches for complex job-shop scheduling problems are based on adoptions of the shifting bottleneck heuristic of Adams et al. (1988). This heuristic decomposes the problem into multiple parallel-machine scheduling problems and subsequently applies appropriate subproblem solution procedures. For this setting, Ovacik and Uzsoy (1997) introduce a disjunctive graph representation that represents batches using dedicated nodes. This representation was also used in Mason and Oey (2003) and Mönch and Rose (2004), where the authors show that a modified shifting bottleneck heuristic outperforms classical dispatching rules. Results were improved in Mönch et al. (2007) by using a genetic algorithm in the subproblem solution procedure. Recently, Bilyk et al. (2014) propose an improved method to solve the underlying parallel-machine scheduling subproblem. A different approach is presented in Yugma et al. (2012). A more global optimization technique is proposed by introducing moves which operate on the whole graph. As well as in the aforementioned work, dedicated nodes are introduced to represent batches. In the present work, we propose a novel approach to take batching decisions: Instead of adding auxiliary nodes, we dynamically compute batches during a traversal of the graph.

3 Batch Oblivious Disjunctive Graphs

To represent schedules, we use a known disjunctive graph model for flexible job-shops (see Dauzère-Pérès and Paulli (1997)). In a disjunctive graph, nodes represent operations and arcs represent dependencies induced by either the route of a job or the ordering of operations on machines. A disjunctive graph models all possible assignments of operations to machines and sequences of operations of the machines. By replacing each disjunctive arc by a conjunctive arc while satisfying some feasibility constraints, a conjunctive graph is constructed which corresponds to a given assignment of each operation to a machine and a given sequence of all operations on the machines. All redundant arcs are removed.

In a conjunctive graph $G = (V, A)$, for each job and each machine, there is a unique edge disjoint path from an artificial start node 0 to an artificial end node $*$. For each node $v \in V$, let us denote by p_v its processing duration, by $r(v) \in V$ its route predecessor, and by $m(v) \in V$ its machine predecessor. In the case without batching, start dates can be computed by traversing the nodes of the graph in topological order. Topological orderings can be computed in linear time (see Kahn (1962)). The start date s_v of a node $v \in V$ depends only on its direct predecessors and can be computed inductively as $s_v := \max(s_{r(v)} + p_{r(v)}, s_{m(v)} + p_{m(v)})$. The topological ordering guarantees that $s_{r(v)}$ and $s_{m(v)}$ are computed before v is visited. For the artificial start node 0, s_0 is set to zero.

Now, we modify the computation of start dates to include batching. Still, we traverse the nodes of the graph using an (arbitrary) topological ordering. Our procedure preserves all ordering and assignment decisions inherent in the given conjunctive graph. Nodes are called *compatible* if they are associated to operations of the same family. Such nodes can be combined in the same batch. All operations of the same batch must start at the same time. Consider a node $v \in V$ visited during the traversal. If $s_{r(v)} + p_{r(v)} \leq s_{m(v)}$, we set $s_v := s_{m(v)}$ to combine v and $m(v)$ in the same batch. Conceptually, this can be seen as changing the duration of the arc between v and $m(v)$ to zero and introducing a zero weighted arc between $m(v)$ and v. In the other case, where $s_{r(v)} + p_{r(v)} > s_{m(v)}$, v is not included in the same batch as $m(v)$. This inclusion would not be tractable in a one-pass graph traversal since a modification of the already computed value $s_{m(v)}$ would trigger the need to propagate the change to prior batch predecessor nodes as well as reachable successor nodes. Note that capacity constraints can be checked in a straightforward way: The current capacity usage is tracked for each node and batches are not extended if the capacity is exceeded.

Algorithm 1 A batching algorithm for a given conjunctive graph G

```
computeStartDates (G)
  s_0 ← 0
  for v ∈ computeTopologicalOrdering(G \ {0})
    if isCompatible(v, m(v)) and isCapacitySufficient(v) and s_{r(v)} + p_{r(v)} ≤ s_{m(v)}
        s_v ← s_{m(v)}
    else
      s_v ← max(s_{r(v)} + p_{r(v)}, s_{m(v)} + p_{m(v)})
```

Algorithm 1 provides the pseudo-code for our dynamic graph evaluation algorithm. Its greedy strategy yields a well defined schedule. Batching decisions taken by the algorithm are not necessarily optimal. However, we can rely on the existence of a conjunctive graph for which our batching algorithm yields an optimal schedule. This is shown in Theorem 1. Let us recall that an optimization criterion is called regular if it is an increasing function of the completion times of jobs.

Theorem 1 *For any given regular criterion, there exists a conjunctive graph G such that Algorithm 1 yields an optimal schedule.*

Proof Consider a feasible schedule that is optimal with respect to the given regular criterion. Its operation start dates s_v are given. We construct a conjunctive graph that defines the order of operations on machines as follows:

a) The graph respects all machine assignment decisions of the optimal schedule.
b) Ordering decisions on the machines respect the start dates of the optimal schedule: If $s_v < s_w$ for $v, w \in V$, then v is ordered before w.
c) Nodes $v, w \in V$ that are part of the same batch (i.e., $s_v = s_w$) are ordered as follows: If $s_{r(v)} + p_{r(v)} < s_{r(w)} + p_{r(w)}$, then v is ordered before w.
d) For two nodes $v, w \in V$ of the same batch with $s_{r(v)} + p_{r(v)} = s_{r(w)} + p_{r(w)}$, their relative order can be arbitrarily decided.

Since those rules are derived from a feasible schedule, this graph is constructed without any cycles. Property c) guarantees that, for all adjacent nodes $v, m(v) \in V$ of the same batch, $s_{r(v)} + p_{r(v)} \leq s_{m(v)}$ holds. This ensures that these nodes are combined in the

same batch by Algorithm 1. Now consider two operations with $s_{r(v)} + p_{r(v)} \leq s_{m(v)}$ that are not part of the same batch in the given optimal schedule. Our algorithm combines them in the same batch and thus yields a different schedule. However, since no operations are postponed and the objective function is regular, the resulting schedule is optimal as well. □

4 Heuristic Algorithms

In the following, we provide an overview of our solution methods based on the introduced batch oblivious disjunctive graph modeling. We utilize a heuristic based on the idea of greedy randomized adaptive search procedures (GRASP, see Feo and Resende (1995)). Initial solutions are created using a starting heuristic comparable to the ones described in Yugma et al. (2012) and Knopp et al. (2014): Jobs are successively inserted in the order of their ratio between due date and weight, and the best insertion position for each operation is selected. The randomization is achieved by disturbing the ordering of job insertions. Multiple initial solutions can be constructed in parallel. To improve constructed solutions, we use a simulated annealing heuristic. Therein, we utilize moves based on insertions and removals of nodes in the conjunctive graph which treat resequencings and reassignments of operations in a uniform way (see Dauzère-Pérès and Paulli (1997)).

Although the graph is unaware of batches, we still can use results of the batching algorithm to determine moves. We compare two strategies. A first strategy modifies batches explicitly by introducing the following moves (cf. Yugma et al. (2012)):

- Swap two random nodes belonging to batches of the same family,
- Move simultaneously all nodes of a random batch to a randomly chosen position,
- Randomly move a single node (potentially changes machine assignment).

In each iteration, one type of moves is randomly selected and a corresponding move is generated. A second strategy performs only simple moves and fills up batches dynamically by reassigning and resequencing operations during graph traversals. The computation of batches is performed after each move and the objective function value can be derived from the obtained start dates.

5 Results and Conclusion

We implemented our algorithms in C++ and conducted preliminary numerical experiments on an Intel Xeon E5-2620 2.1 GHz machine using academic and industrial instances. To assess the performance of our approach, we utilize test instance from a less general parallel machine scheduling problem of Mönch et al. (2005). We observe that the second strategy which uses a simple move together with dynamic resequencings and reassignments outperforms the first strategy of batch aware moves. Table 1 provides first results in terms of average values for total weighted tardiness. The results verify the applicability of our approach. Since the used test instances stem from a less general parallel machine scheduling problem, an important future step would be a systematic assessment of our approach using job-shop based instances. Initial experiments using industrial data show that job-shop instances with batching of industrial dimensions can be handled.

		#Machines			Batch size	
	Time (s)	m=3	m=4	m=5	B=4	B=8
Our approach	120	410	302	227	384	242
Mönch et al. (2005)	27 216	412	300	231	389	240
Yugma et al. (2012)	178	411	278	206	367	229

Table 1 Average total weighted tardiness values for instances of Mönch et al. (2005)

A major advantage of our approach is its simplicity: Avoiding the complexity of additional batching nodes enables the inclusion of further constraints. We see this as a promising direction towards the inclusion of complex properties appearing in real-world problems. In particular, we aim to combine this approach with the more complex routing structures presented in Knopp et al. (2014).

Acknowledgements This work is supported by the ENIAC European Project INTEGRATE.

References

Adams, J., E. Balas, and D. Zawack (1988). The shifting bottleneck procedure for job shop scheduling. *Management science 34*(3), 391–401.

Bilyk, A., L. Mönch, and C. Almeder (2014). Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering* (0), –.

Dauzère-Pérès, S. and J. Paulli (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research 70*, 281–306.

Feo, T. A. and M. G. Resende (1995). Greedy randomized adaptive search procedures. *Journal of global optimization 6*(2), 109–133.

Garey, M. R., D. S. Johnson, and R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research 1*(2), 117–129.

Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM 5*(11), 558–562.

Knopp, S., S. Dauzère-Pérès, and C. Yugma (2014). Flexible Job-Shop Scheduling with Extended Route Flexibility for Semiconductor Manufacturing. In *Proceedings of the 2014 Winter Simulation Conference (WSC)*, pp. 2478–2489. IEEE Press.

Mason, S. J. and K. Oey (2003). Scheduling complex job shops using disjunctive graphs: a cycle elimination procedure. *International journal of production research 41*(5), 981–994.

Mathirajan, M. and A. Sivakumar (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology 29*(9-10), 990–1001.

Mönch, L., H. Balasubramanian, J. W. Fowler, and M. E. Pfund (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research 32*(11), 2731 – 2750.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling 14*(6), 583–599.

Mönch, L. and O. Rose (2004). Shifting-Bottleneck-Heuristik für komplexe Produktionssysteme: Softwaretechnische Realisierung und Leistungsbewertung. *Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik 2*, 145–159.

Mönch, L., R. Schabacker, D. Pabst, and J. W. Fowler (2007). Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops. *European Journal of Operational Research 177*(3), 2100–2118.

Ovacik, I. M. and R. Uzsoy (1997). *Decomposition methods for complex factory scheduling problems*. Kluwer Academic Publishers Boston.

Yugma, C., S. Dauzère-Pérès, C. Artigues, A. Derreumaux, and O. Sibille (2012). A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing. *International Journal of Production Research 50*(8), 2118–2132.

MISTA 2015

The Intermittent Traveling Salesman Problem

San Pham · Patrick De Causmaecker

1 Introduction

This article introduces the Intermittent Traveling Salesman Problem (ITSP) - a new variant of the Traveling Salesman Problem (TSP). The ITSP arises from a practical polishing/drilling application where the temperature of the workpiece is taken into account. Consider a device, such as a laser, visiting a number of spots on a work piece. The aim of the visit is to machine the work piece at this particular spot. The machining causes the piece to heat up locally. Consequently, machining cannot go on for ever and at some point in time the device has to leave the spot, to come back to continue only after a certain timelap. The objective of the problem is to find the minimum time to process all spots, several of which have to be visited a number of times. Since TSP is the special case where all spots have to be visited exactly once, this problem is NP-hard.

In the next sections, we propose two Mixed Integer Programming(MIP) models for the ITSP. An outline of branch-and-bound algorithms for those models is also mentioned.

2 MIP model

Consider a graph $G = (V, E)$ with n nodes. Each arc $(i, j) \in E$ associates with a time duration $c_{ij} > 0$. Each node has a processing time $p_i >= 0$. To simplify the model, we assume that in the beginning of the process, the temperature of each node is 0. The node's temperature increases during its serving time linearly, i.e. one degree per time unit. When not being visited, the workpiece will cool down with the rate of one degree per time unit. The maximum temperature allowed is B.

We construct the graph $G' = (V', E')$ as follow: Each node $i \in V$ is split into $k_i = \lceil p_i/B \rceil$ nodes. We call this set K_i. The processing time of each node $p_i^j = B$ for

San Pham
KU Leuven, KULAK
E-mail: san.pham@kuleuven-kulak.be

Patrick De Causmaecker
KU Leuven, KULAK
E-mail: patrick.decausmaecker@kuleuven-kulak.be

$j = 1..k_i - 1$ and $p_i^{k_i} = (p_i \mod B)$. Clearly, the set K_i for node i with $p_i \leq B$ includes one element only. For each arc $(i,j) \in E'$, $c_{ij} = c_{th}$ with $(t,h) \in E$, $i \in K_t$ and $j \in K_h$. It is easy to see that $c_{ij} = 0$ if $t = h$. To simplify the model, we create two virtual nodes S and T denoting the departing depot and returning depot, $c_{Si} = c_{iT} = 0 \quad \forall i \in V'$.

To solve the problem, we need to find a Hamiltonian cycle for G' which respects the temperature constraints: the difference between the starting time of nodes i and j belonging to the same set K_t must be larger than or equal to B. As one can realize, those time constraints are similar to the ones in TSP with time window (TSPTW) (Desrochers et al. [4]). Therefore, based on the models of the classical TSPTW, we propose two models for the ITSP.

In both two models, the arc variables x_{ij} are included to denote that node j is visited right after node i. The first model is based on the standard model of TSPTW (Desrochers and Laporte [3]) which involes node variables s_i denoting the start time of node i. The link between x and s variables is ensured by a generalization of Miller-Tucker-Zemlin inequalites

$$s_i + c_{ij} + p_i \leq M * (1 - x_{ij}) + s_j \qquad \forall i \in V', j \in V' \setminus \{S\} \tag{1}$$

where M is a large number. To satisfy the temperature constraints, the difference between the starting time of the nodes belonging to the same set must be larger than or equal to B.

$$s_i - s_j \geq B \qquad \forall i,j \text{ from the same set } K_t, i > j \tag{2}$$

Note that after processing a node, the devide can wait for a while before moving to the next one. The condition $i > j$ is used to break the symmetry structure.

The objective of the problem is to mimimize the visiting time of the returning depot. The complete model is represented as below:

$$\min \; s_T \tag{3}$$

subject to (1), (2) and

$$\sum_{j \in V'} x_{ij} = 1 \qquad \forall i \in V' \tag{4}$$

$$\sum_{i \in V'} x_{ij} = 1 \qquad \forall j \in V' \tag{5}$$

$$x_{TS} = 1 \tag{6}$$

In the second model, adopting the idea of Van Eijl [5] we introduce a set of variables y_{ij} denoting the visiting time of node i before visiting j. $y_{ij} = 0$ if $x_{ij} = 0$. The link between x_{ij} and y_{ij} is provided through the inequalities:

$$\sum_{i \in V'} (y_{ij} + (p_i + c_{ij}) * x_{ij}) \leq \sum_k y_{jk} \qquad \forall j \in V' \setminus \{S\} \tag{7}$$

$$y_{ij} <= M * x_{ij} \qquad \forall i,j \in V' \tag{8}$$

The objective function of the second model is also to minimize the visiting time of the node T:

$$\min y_{TS} \tag{9}$$

respects to (4), (5), (6) from model 1, the linking constraint (7), (8) plus the temperature constraints below:

$$\sum_{k \in V'} y_{ik} - \sum_{h \in V'} y_{jh} \geq B \qquad \forall i, j \text{ from the same set } K_t, i > j \tag{10}$$

3 Branch-and-cut algorithm

Each of the two above models has its own strength and weakness. In this work, we try to draw comparisons between them by performing the branch-and-cut algorithm on each model.

There are several factors motivating a branch-and-cut algorithm: the branching rules, the exploring order, the quality of cutting planes... In this paper, we use the branch-and-cut framework of the Gurobi solver [6] where all of the above factors are taken care of by the solver. To strengthen the search, in addition to default cuts of Gurobi, user-defined cutting planes are added. Since most of the cutting planes for the TSPTW presented in [1] are still valid for our model, we take advantage of them into our algorithm. The detailed cutting planes and experimental results will be presented in the final paper.

Acknowledgements Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX. (http://comex.ulb.ac.be), Research Foundation Flanders (FWO) and the Marie Curie ITN STEEP (Grant Agreement no. 316560, http://www.steep-itn.eu/steep/index.aspx). We would like to thank Prof. Frits Spieksma for his suggestion on simplifying the problem.

References

1. Ascheuer, Norbert, Matteo Fischetti, and Martin Grötschel. "Solving the asymmetric travelling salesman problem with time windows by branch-and-cut." Mathematical Programming 90.3 (2001): 475-506.
2. Desrochers, Martin, and Gilbert Laporte. "Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints." Operations Research Letters 10.1 (1991): 27-36.
3. Desrochers, M., and Lenstra, J.K. Karel and Savelsbergh, M.W.P. and Soumis, F. "Vehicle Routing with Time Windows: Optimization and Approximation", in Vehicle Routing: Methods and Studies, Golden, B.L. and Assad, A.A., Elsevier Science Publishers B.V. 1988. 65-84.
4. Van Eijl, C. A. A polyhedral approach to the delivery man problem, Memorandum COSOT 95, Eindhoven University of Technology, 1995.
5. Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual", 2015, http://www.gurobi.com

MISTA 2015

Single machine scheduling: finding the Pareto Set for jobs with equal processing times with respect to criteria L_{max} and C_{max}.

Alexander Lazarev · Dmitry Arkhipov · Frank Werner

1 Introduction

In this paper, the special case of the classical NP-hard scheduling problem $1|r_j|L_{max}$ is considered. There is a single machine and a set of jobs $N = \{1, 2, \ldots, n\}$ to be executed with identical processing times $p_j = p$ for all jobs $j \in N$. We define a *schedule* (or sequence) π as the execution sequence $K_1(\pi), K_2(\pi), \ldots, K_n(\pi)$, where

$$K_1(\pi) \cup K_2(\pi) \cup \cdots \cup K_n(\pi) \equiv N.$$

The equality $K_i(\pi) = j$ means that job $j \in N$ is executed under the ordinal number i in the schedule π. The execution of the job $K_i(\pi) = j$ starts at time

$$R_j(\pi) = \max\{C_{K_{i-1}}(\pi), r_{K_i(\pi)}\}$$

(where $C_{K_0}(\pi) = 0$) and finishes at time

$$R_j(\pi) + p = C_j(\pi),$$

where $C_j(\pi)$ is the *completion time* of the job $j \in N$. Let us denote the *lateness* of job j under the schedule π as

$$L_j(\pi) = C_j(\pi) - d_j.$$

The maximum completion time and the maximum lateness are denoted as $C_{\max}$ and $L_{\max}$, respectively. Let us call the schedule π *allowable* for the set N if all jobs according to the schedule π execute without preemptions and intersections. We denote the set of all

Alexander Lazarev
V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russian Federation;
Lomonosov Moscow State University, Moscow, Russian Federation;
Moscow Institute of Physics and Technology, Dolgoprudny,
Russian Federation;
National Research University Higher School of Economics,Moscow, Russian Federation
E-mail: jobmath@mail.ru

Dmitry Arkhipov
V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russian Federation
E-mail: miptrafter@gmail.com

Frank Werner
Faculty of Mathematics, Institute of Mathematical Optimization, Otto-von-Guericke University Magdeburg, Germany
E-mail: frank.werner@ovgu.de

allowable schedules as Π. The goal is to find a feasible schedule $\pi \in \Pi$, which satisfies the following optimization criterion:

$$\min_{\pi \in \Pi} \max_{j \in N} L_j(\pi).$$

2 The auxiliary problem

Let us formulate an auxiliary problem. We consider the same set of jobs $N = \{1, 2, \ldots, n\}$ and a *bound on the maximum lateness* y. The goal is to construct a schedule satisfying the following optimization criterion:

$$\min_{\pi \in \Pi} \max_{j \in N} C_j(\pi) | L_{\max}(\pi) < y.$$

For each set of due dates $d_1, \ldots, d_n$ and the bound on the lateness y, *deadlines* D_j can be calculated by the following formula:

$$D_j = d_j + y.$$

An allowable schedule satisfying this restriction is called *feasible*. To construct the solution of the auxiliary problem, we consider the approach presented in [3]. Next, we briefly recall the main idea from this paper.

The **auxiliary** algorithm works as follows. While the completion times of all jobs are lower than its deadlines, schedule the jobs according to the algorithm, presented in [4]. If for any job $X \in N$, the inequality

$$C_X \geq D_X$$

holds, then execute the special procedure $CRISIS(X)$. This procedure finds the job A, which is already scheduled with the latest completion time, but for which

$$D_A > D_X$$

holds. This job is called $Pull(X)$ and all jobs which are already scheduled after $Pull(X)$ and X constitute the *restricted set* $S(A, X]$. We define $r_{S(A,X]}$ to be the earliest time when the jobs of $S(A, X]$ can start their execution. The procedure $CRISIS(X)$ reschedules the jobs of the set $\{A\} \cup S(A, X]$. The procedure fails when a job $Pull(X)$ for a crisis job X does not exist. After a successful execution of the procedure $CRISIS(X)$, Schrage's algorithm [4] is used to schedule the jobs. Such a scheduling is repeated until any call of the procedure $CRISIS()$ fails or all jobs from the set N have been successfully scheduled.

3 Solution of the main problem

Next, we consider the main problem $1|r_j, p_j = p|L_{max}$. We also present an algorithm to obtain the Pareto set of schedules with respect to the criteria L_{max} and C_{max}. First, we introduce a procedure $CHECK(\pi, N, y)$ which constructs the schedule π^* as follows.

$CHECK(\pi, N, y)$

1. Set the lateness bound y and a time $t = \min_{i \in N} r_i$.
2. Set the deadlines $D_i := d_i + y$.
3. If all jobs from the set N have been scheduled, go to step 7.
4. While t is not in the interval $[r_{S(A,X]}, D_X)$ for any restricted set $S(A, X]$ from the schedule π that has not yet been completely performed, execute the jobs under π^* according to Schrage's algorithm.
5. Otherwise, execute only the jobs from the set $S(A, X]$ under the schedule π, and then go to step 3.

6. If in steps 4-5 any job Y experiences a crisis, run the procedure $CRISIS(Y)$.
7. $return(\pi^*)$.

Lemma 1 *Let π and π' be the schedules constructed by the auxiliary algorithm for the bounds y and y', respectively, and*

$$\pi^* = CHECK(\pi, N, y).$$

If $y < y'$, then

$$\pi^* = \pi'.$$

holds.

Next, we describe the main algorithm M to obtain the Pareto set with respect to the criteria L_{max} and C_{max}.

MAIN ALGORITHM (Algorithm M)

1. Set the bound $y_0 := +\infty$.
2. Construct the schedule π_1 according to the auxiliary algorithm,
   and add it to Φ, i.e.: $\Phi := \{\pi_1\}$;
   set the counter $k := 1$;
   set the bound $y_1 := L_{max}(\pi_1)$.
3. Construct the schedule $\pi_{k+1} = CHECK(\pi_k, N, y_k)$.
   a) If the schedule $CHECK(\pi_k, N, y)$ exists, then:
      add π_{k+1} to the set Φ, i.e.: $\Phi := \Phi \cup \pi_k$;
      set $y_k = L_{max}(\pi_k)$;
      increase the counter k, i.e.: $k := k + 1$;
      repeat step 3.
   b) Otherwise, $return(\Phi)$.

At last, we formulate and prove some important lemmas and a theorem, which show that algorithm M finds the Pareto set Φ in $O(n^2 \log n)$ operations.

Lemma 2 *If any job becomes a crisis job for the second time, then the algorithm stops.*

Theorem 1 *After the execution of Algorithm M, the Pareto set of schedules Φ according to the criteria L_{max} and C_{max} has been constructed, where the schedules Φ_1 and $\Phi_{|\Phi|}$ are optimal according to criteria L_{max} and C_{max} respectively. For this set*

$$|\Phi| \leq n + 1$$

holds.

Lemma 3 *The complexity of Algorithm M is $O(n^2 \log n)$.*

4 Metric analysis

The metric ρ for the instances of problem $1|r_j|L_{max}$ was introduced in [5]. We estimate a metric distance $\rho^p(A)$ between an arbitrary instance A which holds $p_1^A \leq \cdots \leq p_n^A$ and a set of polynomial solvable instances with the identical processing times of jobs as:

$$\rho^p(A) \leq \sum_{i=1}^{[(n-1)/2]} p_{n-i+1}^A - p_i^A.$$

The prove that estimated bound is tight and present a polynomial algorithm to find the instance B for an arbitrary instance A which satisfy

$$\rho(A, B) = \rho^p(A).$$

The results of numerical experiments are also presented.

References

1. Kravchenko, S.A. and Werner, F.: Parallel Machine Problems with Equal Processing Times. *Journal of Scheduling*. Vol. 14, No. 5, 2011, 435 - 444.
2. Garey, M.R.; Johnson, D.S.; Simons, B.B. and Tarjan, R.E.: Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. COMPUT*. Vol. 10, No. 2, May 1981, 256 - 269.
3. Simons, B.B.: A fast algorithm for single processor scheduling. *In 19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978)*, 246-252.
4. Schrage, L.: Solving Resource-Constrained Network Problems by Implicit Enumeration: Non Preemptive Case. *Operations Research*. Vol. 18 Issue 2, 1970, 263 - 278.
5. Lazarev, A.A.: The Pareto-optimal set of the NP-hard problem of minimization of the maximum lateness for a single machine. *Journal of Computer and Systems Sciences International. M.: SP MAIK Nauka/Interperiodica.* 2006. 45, No. 6. . 943-949.

MISTA 2015

Scheduling Jobs in a Two-Stage Flexible Flow Shop with Batch Processing Machines

Yi Tan • Lars Mönch • John Fowler

1 Introduction and Problem Setting

Scheduling problems for flexible flow shops with batch processing machines arise in semiconductor manufacturing [5]. A batch is defined as a set of jobs that can be processed together at the same time on a single machine. It is reasonable to study two-stage flexible flow shops with batching because two consecutive batch processes of oxidation and nitration, respectively, occur in current wafer fabs. We assume that each job j belongs to family $f_s(j) \in \{1,\ldots,F_s\}$ at stage s where F_s is the number of incompatible families at stage s. Only jobs of the same family can be batched together. The common processing time of all jobs of family f at stage s is given by p_{fs}. Each job j has a due date d_j, a ready time r_j, and a weight w_j. The completion time of the operation of job j at stage s is denoted by C_{js}. We assume that stage s contains m_s identical parallel machines with a maximum batch size of B_s. The performance measure total weighted tardiness (TWT) is the summation of the weighted tardiness $w_j T_j$ over all jobs $j = 1,\ldots,n$, where $T_j := \max(C_{j2} - d_j, 0)$. Using the $\alpha \mid \beta \mid \gamma$ notation from scheduling theory, the researched problem can be represented as follows:

$$FF2 \mid p\text{-}batch, incompatible, r_j \mid TWT, \qquad (1)$$

where we denote by $FF2$ a two-stage flexible flow shop with identical parallel machines at each stage. Problem (1) is NP-hard since it contains the NP-hard problem $1 \| TWT$ as a special case. We know from [2,6] that the shifting bottleneck heuristic does not work well for flow shops. A two-machine flow shop scheduling problem for the mean flow time criterion is discussed in [3]. The machine at the first stage is a batch processing machine. There is a buffer with limited capacity between the two machines. A differential evolution algorithm is used to batch jobs at the first stage. A two-stage flexible flow shop with a limited waiting time

Yi Tan
University of Hagen
E-mail: Yi.Tan@fernuni-hagen.de

Lars Mönch
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

John W. Fowler
Arizona State University
E-mail: John.Fowler@asu.edu

constraint and batching machines at the first stage is discussed in [7]. A mixed integer programming formulation and a heuristic is proposed for the makespan objective. The diffusion area in a wafer fab is modeled in [9] as a flexible job shop where all machines are batching machines. A combined objective is considered. A sampling procedure and a simulated annealing scheme are proposed that are based on a disjunctive graph representation. However, in the present paper, we consider the TWT performance measure which is different from [9]. A two-machine flow shop problem is discussed in [8]. In the present paper, we extend this approach to a flexible flow shop setting. In addition, the number of machines at each stage is also a design factor in our computational experiments.

2 Decomposition Heuristic

The decomposition approach is based on the idea that the scheduling subproblems at the two stages can be decoupled and solved independently from each other if appropriate due dates and ready times can be determined for the first and the second stage, respectively. We propose an iterative scheme to come up with internal due dates and ready times. Therefore, the due date of operation j at the first stage is denoted by $d_{j1}^{(1)}$, where $l \geq 1$ is an iteration counter. The ready times of the operations at the second stage are denoted for iteration l by $r_{j2}^{(1)}$. The slack of job j is given by $\Delta_j := d_j - p_{f_2(j)2} - p_{f_1(j)1} - r_j$. We add half of this slack to the earliest start time of the second stage to obtain the first stage due date for iteration 1, i.e. $d_{j1}^{(1)} := r_j + p_{f_1(j)1} + 1/2\,\Delta_j$, while the following update scheme is used for the due dates in iteration $l \geq 2$:

$$d_{j1}^{(l)} := (1-\alpha)\, r_{j2}^{(l-1)} + \alpha\left(d_j - p_{f_2(j)2} - w_{j2}^{(l-1)}\right) + \beta\, w_{j2}^{(l-1)}. \tag{2}$$

Here, $w_{j2}^{(l-1)}$ is the waiting time of job j observed at the second stage in iteration $l-1$ and $\alpha, \beta \in [0,2]$ are parameters. After the first stage subproblem is solved in iteration l, we know the completions times of the jobs at this stage that are denoted by $C_{j1}^{(l)}$. We then set

$$r_{j2}^{(l)} := C_{j1}^{(l)} \tag{3}$$

for the ready times of the jobs at the second stage in iteration l. We show that the update scheme given by (2) and (3) can be interpreted as a specific instance of an iterative simulation procedure where the evaluation of the schedule in a given iteration is carried out by a simple deterministic forward simulation. This means that starting from an initial guess of the internal due dates and ready times, we iterate to improve these values. Typically, not more than 20 iterations are required. Each of the resulting subproblems of the form $P \mid p\text{-}batch, incompatible, r_j \mid TWT$ is either solved by the Time Window Decomposition (TWD) heuristic based on Batched Apparent Tardiness Cost (BATC) dispatching proposed in [4] or by the variable neighborhood search (VNS) scheme described in [1]. The VNS scheme is based on five neighborhood structures that work on the final solution representation, i.e. batches assigned to machines and the corresponding sequences of batches. The local search procedure within the VNS scheme balances the workload on the machines, moves batches across machines, swaps jobs between batches, and swaps batches across machines. The two heuristics are called $TWD(iter)$ and $VNS(iter)$, where $iter$ is the number of iterations.

3 Computational Results

We generate problem instances according to the following design. In each instance, 60 jobs per first stage family are considered. There are eight machines across the two stages. Three fami-

lies are used at the first stage. At the second stage, each first stage family will become either one or two families with equal probability. The processing times are integers that are selected with a given probability [4]. The maximum batch size of the two stages is $B_s \in \{2,4\}$. Ready times of the form $r_j \sim U\left(0, a\left(1/(B_1 m_1)\sum_{j=1}^{n} p_{f_1(j)1} + 1/(B_2 m_2)\sum_{j=1}^{n} p_{f_2(j)2}\right)\right)$ are used, where $a \in \{0.25, 0.75\}$ is a parameter. The due dates are chosen by the expression $d_j := r_j + FF\left(p_{f_1(j)1} + p_{f_2(j)2}\right)$, where $FF \in \{1.1, 1.3\}$. We use $w_j \sim U[0,1]$. The workload of stage s is measured by $WL_s(m_s) = 1/(B_s m_s)\sum_{j=1}^{n} p_{f_s(j)s}$. Starting from the total number of machines m, we consider all possible pairs $(m_1, m - m_1)$. A balanced workload is given by the machine pair $(m_1^*, m_2^*) := \arg\min_{(m_1,m_2)} |WL_1(m_1) - WL_2(m_2)|$. If possible, we consider the two additional pairs $(m_1^* - 1, m_2^* + 1)$ and $(m_1^* + 1, m_2^* - 1)$ that mimic the situation that the first or the second stage is the bottleneck. The three different configurations are denoted by $BL, BN1, BN2$, respectively. Overall, we consider 16 factor combinations, each of them with three independent problem instances. An instance is represented by a specific job set. We determine the ratio of the TWT value obtained by $VNS(20)$ and the TWT value obtained by $TWD(1)$ for the same configuration. The average over all corresponding instances is shown in Table 1. The shown VNS computing time refers to a single pair (α, β). The computing time of $TWD(1)$ is negligible.

Table 1: Computational Results for Problem (1)

	TWT of $VNS(20)$ relative to TWT of $TWD(1)$			Computing Time (s)		
Configuration	$BN1$	BL	$BN2$	$BN1$	BL	$BN2$
Factor/Level (B_1, B_2)						
(2, 2)	0.792 (1.047)	0.680 (0.376)	0.826 (1.062)	448	453	478
(2, 4)	0.761 (0.468)	0.801 (0.511)	0.855 (1.522)	437	450	479
(4, 2)	0.935 (1.734)	0.735 (0.432)	0.713 (0.445)	413	467	450
(4, 4)	0.808 (0.952)	0.721 (0.539)	0.793 (0.920)	443	429	448
a						
0.25	0.928 (1.085)	0.851 (0.581)	0.867 (1.025)	488	512	540
0.75	0.720 (1.015)	0.617 (0.349)	0.728 (0.952)	383	388	388
FF						
1.1	0.882 (1.190)	0.796 (0.509)	0.812 (0.917)	436	450	466
1.3	0.766 (0.911)	0.672 (0.421)	0.782 (1.059)	435	450	461
Overall	**0.824 (1.050)**	**0.734 (0.465)**	**0.797 (0.988)**	**435**	**450**	**464**

We solve each instance for the three configurations using $TWD(1)$ and $VNS(20)$. We consider different pairs (α, β) from an appropriate grid. The smallest TWT value for the grid points and each instance and configuration is considered. To make the three configurations comparable, we present the ratio of the TWT value obtained from $VNS(20)$ and the average TWT value obtained by $TWD(1)$ for the three configurations in Table 1 in brackets. We can see from

Table 1 that $VNS(20)$ clearly outperforms $TWD(1)$. The BL configuration leads often to the smallest TWT values among the three configurations and is therefore a favorable setting. In future work, we are interested in carrying out more detailed computational experiments and in designing a VNS scheme for problem (1) that does not rely on decomposition.

References

1. Bilyk, A., Mönch, L., Almeder, C. Scheduling with ready time and precedence constraints on parallel batch machines using metaheuristics. Computers & Industrial Engineering, 78, 175-185 (2014)
2. Demirkol, E., Uzsoy, R. Decomposition methods for reentrant flow shops with sequence-dependent setup times. Journal of Scheduling, 3, 155-177 (2000)
3. Fu, Q., Sivakumar, A. I., Li, K. Optimizing of flow-shop scheduling with batch processor and limited buffer. International Journal of Production Research, 50(8), 2267-2285, (2012)
4. Mönch L., Balasubramanian, H., Fowler, J., Pfund, M. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers & Operations Research, 32(11), 2731-2750 (2005)
5. Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., Rose, O. Scheduling semiconductor manufacturing operations: problems, solution techniques, and future challenges. Journal of Scheduling, 14(6), 583-595 (2011)
6. Mukherjee, S., Chatterjee, A. K. Applying machine-based decomposition in 2-machine Flow Shops. European Journal of Operational Research, 169, 723-741 (2006)
7. Su, L. H. A hybrid two-stage flow shop with limited waiting time constraints. Computers & Industrial Engineering, 44 (3), 409–424 (2003)
8. Tan, Y., Mönch, L., Fowler, J. W. A decomposition heuristic for a two-machine flow shop with batch processing. Proceedings of the 2014 Winter Simulation Conference (2014)
9. Yugma, C., Dauzère-Pérès, S., Artigues, C., Derreumaux, A., Sibille, O. A batching and scheduling algorithm for the diffussion area in semiconductor manufacturing. International Journal of Production Research, 50(8), 2118-2132 (2012)

MISTA 2015

Modeling Uncertainty in Vessel Crew Scheduling

Seda Sucu • Alexander Leggate • Kerem Akartunalı • Robert Van Der Meer

1 Introduction

Crew scheduling is an extensive and important subject for scheduling problems in optimization. It is concerned with the assignment of crew members to create work timetables (schedules) for an organization depending on the requirements and aims of the organization. There are several factors that have to be taken into consideration to decide the assignment of crew members. Apart from the required number of staff and required skills of employees, assignments should conform to the rules and regulations of the specific sector.

The basic input of most crew scheduling problems is the set of crew members and the set of tasks that should be carried out according to the definition of tasks and skill levels of employees. The common features of these problems are that the tasks should be completed in the defined time window, in an actual task environment by taking into consideration the legal and contractual requirements. According to these characteristics, the solution method searches for the best allocation of staff members in general.

Seda Sucu
Department of Management Science, University of Strathclyde
E-mail: seda.sucu@strath.ac.uk

Alexander Leggate
Department of Management Science, University of Strathclyde
E-mail: alexander.leggate@ strath.ac.uk

Kerem Akartunalı
Department of Management Science, University of Strathclyde
E-mail: kerem.akartunali@strath.ac.uk

Robert Van Der Meer
Department of Management Science, University of Strathclyde
E-mail: robert.van-der-meer@ strath.ac.uk

In this study we aim to evaluate reasonable solution methods for a cost minimization problem based on changes in the scheduling of vessel crews. The focus of this study will be based on crew scheduling in transportation settings specific to the maritime industry. The crew cost for this transportation cost problem depends on different variables. The cost includes the salary of the crew members, accommodation and food expenses and movement cost of crew members. Movement cost is calculated with respect to the travel expenses from a gateway city near their home to the departure port of the ship, airfare, visa expenses, hotel, meals, and crew members return expenses after their duty. As a result, crew cost is a comparatively significant cost factor for shipping companies [8].

Apart from the importance of the crew costs, crew scheduling problems are inherently intractable problems due to having binary problems with linear constraints [6]. This feature makes them attractive as optimization problems. To obtain an optimal solution with a solver in a reasonable time is not generally possible for problems of a realistic size. Common constraint in staff scheduling problems assigning staffs to the operations while preventing overlaps and being sure all of the operations are covered. Due to this constraint, most of the crew scheduling problems is known to be NP-Hard problems [7]; as the size of the problem becomes larger, the complexity level of this kind of problems also increases. Even though crew scheduling problems in the transportation industry have a significant place in the scheduling literature, maritime crew scheduling problems are not as popular as airline settings. There are several reasons to explain the lack of studies for crew scheduling problems in vessels. These reasons can be explained by the long planning time horizons in maritime context, lower visibility of the shipping industry compared to rail, road and air, and the higher level of uncertainty and the greater need for recovery schedules [3].

In our study, instead of scheduling the crew members from scratch, we are more interested in how the current schedule could be recovered in the face of environmental uncertainty. Since crew schedules in vessels tend to be affected by environmental factors, such schedules need to be adjusted or updated. This feature changes the basis of the study to the field of recovery scheduling problems. Barnhart et. al. [1] give a place to the recovery problem in their study by presenting a crew recovery model developed by Lettovsky et. al. [11]. In this model, the cost of adjusted pairings, reserve crew, deadheaded crews and cancellation are aimed to be minimized. However, there are not too many studies for the recovery problem: some heuristic search algorithms, dynamic programming algorithm, and column generation methods play a role in the existing literature [1]. A good survey paper is provided by [4] which include recent disruption management (recovery) methods in the airline industry. The authors give comprehensive information about the recovery problem in an airline setting with respect to different objective functions for the different resources of the disruptions. Guo [9] has a different approach to the recovery problem. His study is aimed at minimizing the changes in the current schedule. The problem is formulated as a set partitioning problem and he used both a column generation approach and a hybrid of a genetic algorithm with a local search.

In relation to the above studies, it is hard to have a completely deterministic data set and construct a model without making strong assumptions. Since we aim to obtain practical results for a real life problem, we want to explore the source of uncertainty in this problem setting and to look for contributions from robust optimization. Berstimas and Sim [2] addressed data uncertainty for discrete optimization and proposed a robust integer programming problem of moderately larger size and an algorithm for robust network flows that deals with robust counterpart by solving a polynomial number of nominal minimum cost flow problems in a modified network. Ehrgott and Ryan [5] discussed solution method for bi-criteria optimization problem in airline crew scheduling with maximizing robustness of crew schedules and minimizing cost; although, the improved the robustness cost minimization is not supported at the same time. Therefore, their method provides robust schedules with a small increase in cost. Weare and Fagerholt [10] study on a real life supply vessel planning problem to minimize installation costs. They gave a deterministic mathematical model for their problem. Afterwards, they suggested some robust approaches by adding slack to the voyages and schedules for the uncertain weather conditions impact on the installations. Considering these

robustness approaches, they suggested an algorithm which is combination of optimization and simulation methods and they showed that for the actual sized problems at least %3 savings are possible. Their problem has some similarities with our problem in terms of the final schedule. They obtain schedules for vessels while we construct schedules for crew members.

In this section, the motivation behind this study and a small part of the literature review is given. In the following sections, the problem will be described and possible findings will be discussed.

2 Vessel Crew Scheduling Problem Description

Different types of operation modes are available in a maritime transportation system. The variety between operation types causes variation in the regulations, rules, length of planning horizons and the working conditions of the crew. Apart from the operation settings, the policy of the company has a significant effect on the decision process for the crew schedules. Liner, tramp and industrial shipping are the three main types of maritime operations. Liner shipping carries goods or people on a pre-determined route; while tramp shipping involves cargo transportation to meet supplies and demands on a route that is not necessarily pre-determined. Industrial shipping has similarities with tramp shipping in terms of cargo transportation; but a difference is that in industrial shipping vessel owners carry their own cargo. In addition to these three settings, offshore supply vessels support the branch of the maritime industry concerned with constructions, installations, remote control of vehicles. In this study, we assume to be working with offshore supply vessels that have a default crew schedule. In this case, there are regular rotations and crew assignment for defined routes. The schedules are constructed for these routine rotations by considering several rules and regulations that are based on special requirements relating to their company policy, work and safety regulations in vessels. This does not mean that these schedules are resistant to the disruption factors.

Off-shore supply vessels largely have long planning horizons which can affect the resilience of a constructed schedule. The need of back-up schedules for vessel crews is based on the need for a long planning horizon. The length of this planning horizon may vary between three to six months and accordingly the staff duty periods are also long: generally about four weeks. Since the schedules for this kind of long periods are organized in advance, the need for change in the current schedule becomes inevitable. To achieve a robust recovery schedule after any changes in a reasonable solution time with minimum cost is the main concern of this study. Crew change cost can vary according to the contract type of crew members, their countries of origin and the locations of vessels. Some of the employees are contracted, while some others are agency crew. If contracted employees are not sufficient, agency members can be assigned to any operation. The crew's working conditions are important in formulating the constraints. All crew members need to rest by meeting the minimum rest and maximum working time conditions and they are not allowed to work consecutively for more than a particular number of weeks (generally 10 weeks). This number also changes according to the contract of each employee. Other basic constraints are related to the skill level of employees. Not all duties have the same skill level requirement and, similarly, not all crew members have the same skill level. Accordingly, employees should only be assigned to the tasks for which they are eligible. In addition to these as essential constraints of crew scheduling problems, all tasks should be covered and crew members cannot be assigned more than one task in the same time period involving the objective function minimizing crew change cost.

3 Solution Approaches

To solve this problem, we have first of all proposed an integer programming model and heuristic method by assuming all the parameters are known and certain. Both solution methods are applied to a real life sized problem with generated data. The IP model has a long solution

time with Xpress-Ive Fico Solver, which is hardly applicable and not practical for showing resistance to sudden changes.

In the heuristic method, we start with a feasible initial solution and do neighborhood search with forward- backward extension or swaps between the assigned crew members by taking into consideration all of the constraints through randomly listed employees; and we apply the changes if there is any improvement in terms of minimizing cost. If there are no improvements, we look for the second minimum solution, and keep these changes. After that point another random list is generated and these steps are recurred until the iteration or time limit is exceeded. Since we have a time and iteration limit, this method is efficient in finding alternative solutions in practically applicable solution times. This method is also tested in Xpress-Ive. To obtain more efficient results in terms of the optimality gap, this heuristic is being coded in C++. Our studies for improvements of this heuristic are still continuing.

Since working with uncertain parameter sets is another concern in this study, we are now also planning to work with robust optimization methods. Transportation costs for crew members can hardly be assumed to be fully deterministic based on the timing of sudden changes. This situation leads to uncertain factors in the objective function minimizing crew change cost. We are thus aiming to define the parameter of transportation cost with intervals instead of exact values, and then to minimize the crew change cost in the worst possible case with these given intervals to have more robust solutions for crew schedules in vessels.

References

1. C. Barnhart, A. M. Cohn , E. Johnson, D. Klabjan, G. L. Nemhauser, P. H. Vanc , Airline Crew Scheduling: Handbook of transportation science, 517-560, Springer, US, (2003).
2. D. Bertsimas, M. Sim, Robust discrete optimization and network flows, Mathematical Programming, 98: 49–71, (2003).
3. M. Christiansen, K. Fagerholt, B. Nygreen and D. Ronen, Maritime Transportation, In: Barnhart, C. Laporte, G. (Eds.), Handbook in OR and MS, 4: 189-284, (2007).
4. J. Clausen, A. Larsen, J. Larsen, N.J. Rezanova, Disruption management in the airline industry concepts, models and methods, Computers and Operations Research, 37 (5):809-821, (2010).
5. M. Ehrgott and D. M. Ryan, Constructing robust crew schedules with bicriteria optimization, Journal of Multi-Criteria Decision Analysis, 11: 139–150, (2002).
6. L. S. Franz, J. L. Miller, Scheduling medical residents to rotations: solving the large-scale multi period staff assignment problem, Operations Research, 41(2), 269-279, (1993).
7. M. R. Garey, D. S. Johnson, Computers and Intractability: A guide to NP-completeness, W.H. Freeman, San Francisco (1979).
8. R. E. Giachetti, P. Damodaran, S. Mestry and C. Prada, Optimization-based decision supportsystem for crew scheduling in the cruise industry, Computers and Industrial Engineering, 64(1): 500-510, (2013).
9. Y. Guo, A decision support framework for the airline crew schedule disruption management with strategy mapping, In Operations Research Proceedings, 158-165, (2004).
10. E. E. Halvorsen-Weare, K. Fagerholt, Robust supply vessel planning, In Network Optimization, Springer Berlin Heidelberg, 559-573, (2011).
11. L. Lettovsky, E. L. Johnson, G. L. Nemhauser, Airline crew recovery, Transportation Science, 34(4), 337-348, (2000).

7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015)
25-28 August 2015, Prague, Czech Republic

Multimode Time-Constrained Scheduling Problems with Generalized Temporal Constraints and Labor Skills

Tamara Borreguero Sanchidrián · Christian Artigues · Álvaro García Sánchez · Miguel Ortega Mier · Pierre Lopez

1 Introduction

Scheduling problems have been the subject of continuous research since the early days of operations research. They are NP-hard optimization problems and, in practice, among the most intractable classical ones. Henceforth, most of research works on scheduling are focused on one of the most difficult problems that is the Resource Constrained Project Scheduling Problem (RCPSP), which consists in scheduling several tasks subject to resource and precedence constraints [1]. Brucker *et al.* [2] provided a classification for this kind of problems together with an overview on existing solution methods. There have been a wide range of studies on both heuristic and metaheuristic methods for solving the RCPSP, as well as different Mixed-Integer Linear Programming (MILP) models [3], [4], [5], [6], [7]. Recently, Koné *et al.* [8] proposed the use of event-based formulations that, despite a poor LP relaxation have the advantage to be able to tackle instances having large time horizons. However, the event-based formulations proposed

Tamara Borreguero Sanchidrián
Airbus Defence & Space; Industrial Engineering and Logistics Reserach Group, ETSII, Polytecnic University of Madrid
E-mail: Tamara.borreguero@airbus.com

Christian Artigues
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France Univ de Toulouse, LAAS, F-31400 Toulouse, France
E-mail: artigues@laas.fr

Álvaro García Sánchez
Industrial Engineering and Logistics Reserach Group, ETSII, Polytecnic University of Madrid. Jos Gutierrez Abascal 2 (28006) Madrid
E-mail: alvaro.garcia@upm.es

Miguel Ortega Mier
Industrial Engineering and Logistics Reserach Group, ETSII, Polytecnic University of Madrid. Jos Gutierrez Abascal 2 (28006) Madrid
E-mail: miguel.ortega.mier@upm.es

Pierre Lopez
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France Univ de Toulouse, LAAS, F-31400 Toulouse, France
E-mail: pierre.lopez@lass.fr

by Koné *et al.* are suitable for the standard RCPSP, which includes some assumptions that are too restrictive for many applications [5]. Therefore, it is of great interest to improve this kind of formulations so that they can be used on more practical contexts.

Furthermore, the RCPSP focuses on minimizing the project makespan given a set of available resources. Although this is appropriate for many real problems there are other wide range of cases where the main objective is to minimize the resource consumption.

This is the case in industries, such as aircraft or ships manufacturing, where the total makespan is usually fixed by the expected production rate or the client demand [9]. It can also be applied to the field of project scheduling, where the makespan is fixed and the objective is to engage a minimum number of resources. Those problems are Time-Constrained Scheduling Problems (TCSP), which have been very little treated in the literature. Although Möhring introduced them in 1984 [10], few recent references address this type of problems: [11], [12], [13] and [9]. Neither of them provide with an exact method for this problem and, moreover, the first two address some very specific mono-mode cases.

In this work we have solved a TCSP related to the scheduling of tasks to be done on a platform from an aircraft final assembly line. It is a multimode TCSP with generalized temporal constraints and several labor skills. Actually, our contribution can be stated as follows. First, on the modeling side, it includes the allowance of multiple modes per task, not only linked to the number of workers but also to their skill. Furthermore, it uses general temporal constraints, including some that are not commonly addressed for neither the TCSP nor the RCPSP. Moreover, we have developed two MILP event-based formulations that provide us with an exact MILP formulation for the TCSP. Finally, we are currently working on the use of constraint programming to solve the problem, in order to be able to compare the performance of both types of paradigms and, ultimately, to hybridize them.

2 Problem Statement

Aeronautical assembly lines consist of a series of platforms where different works are executed. Each product has to go through all the platforms. At the same time, the line is synchronized, which means that the time that each product remains on a platform is always the same and equal to the rate at which the assembly line produces its output. Each platform has a fixed makespan and a number of tasks to be performed. The scheduling decision consists on establishing the order in which the tasks will be done along with the resources allocated to each of them, given the line *takt time* (pace of production) and a set of workers per platform.

In accordance with the classification $\alpha \,|\, \beta \,|\, \gamma$ introduced by Brucker *et al.* [2], the scheduling of the tasks from an aeronautical platform is denoted by:
$MPS_m, \sigma,\ \rho \mid temp \mid\ \sum c_k \max r_{kt}$, with:

- $\alpha = MPS_m, \sigma, \rho$. Each activity can be processed in several alternative modes and there exists a set of renewable resources available for each time period during the project execution: the number of operators (each of them belonging to a profile) and the space in each of the platform's working areas. Also, each mode for an activity defines a combination of operator profile, number of operators and durations. All the operators assigned to an activity must be from the same profile and the range of possible numbers of allocated operators per task is independent of the chosen profile.

- $\beta = temp$. There are precedence constraints (task w' cannot start until task w has been completed), non-parallel (also called disjunctive) constraints (tasks w and w' cannot be in progress at the same time, but there is no predefined precedence relation between them), and maximal time lags between tasks (task w' must start within a maximal time after w has been completed). All the temporal constraints are independent from the mode in which a task is executed.
- $\gamma = \sum c_k \max r_{kt}$. The objective function is to minimize the resource investment, which is in this case equivalent to the minimization of the labor cost of the assembly, as the operators once assigned to a platform stay working on it for all the takt time.

3 Solving Techniques: Two MILP Formulations and CP

Different MILP formulations have been proposed to solve the RCPSP. The first ones were *Discrete time formulations*, proposed by Pritsker *et al.* [14]. Afterwards, *Continuous time formulations* were proposed by Alvarez-Valdés and Tamarit (Forbidden sets formulations, [15]) and Artigues *et al.* (Flow-based continuous time formulations, [16]). More recently, *Event-based formulations* were developed by Koné *et al.* in 2011 [8] from a model introduced by Zapata *et al.* [17]. They provided different methods (including MILP exact methods and a heuristic) and concluded that event-based formulations outperformed the previous MILP models and performed even better than the heuristic for some highly cumulative instances, as it is our case.

However, Koné *et al.*'s event-based formulations are suitable for the standard RCPSP, which includes some assumptions that are too restrictive for many applications [5], including ours. Extension of previously existing event-based models to our problem is not straightforward. The main variables x_{we}, y_{we} and z_{we} (related to start, finish, or in-process for activity w at event e, respectively) have been modified with two new sub-indices in order to deal with the multiple modes per task. As well as this, the original formulations included only general precedence constraints. Thus, new constraints and variables have been added in order to take into account the maximal time lags and non-parallel constraints. Finally, the objective function has been modified to tackle with the time constraint approach.

For the performance analysis, we have created a new set of instances of up to 11 tasks each, due to the uncommon structure of the problem with respect to the previous existing instance libraries. Throughout the computational results we have solved the instances up to optimality with both event-based formulations and compared the performance of each of them. The results of these comparisons are consistent with the ones reported by Koné *et al.* for the single mode RCPSP with only precedence constraints [8]: one of the formulations (so-called *On/Off*) outperforms the other on most of the instances. Finally, we have identified the number of events as one of the major factors that have an impact on the instance hardness (see Figure 1).

Our next step is to improve these results by incorporating alternative approaches in our method. In particular, Constraint Programming (CP) has been proven to be an efficient method on several combinatorial optimization problems, especially scheduling problems. Therefore, our current works are on the proposition of a global scheduling constraint, and associated filtering techniques, to solve the RCPSP under study. CP solution scheme encompasses also constraint propagation techniques that can be used as pre-processing to calculate the relevant number of events; this issue would lead to major performance improvements. A promising research direction is to propose a

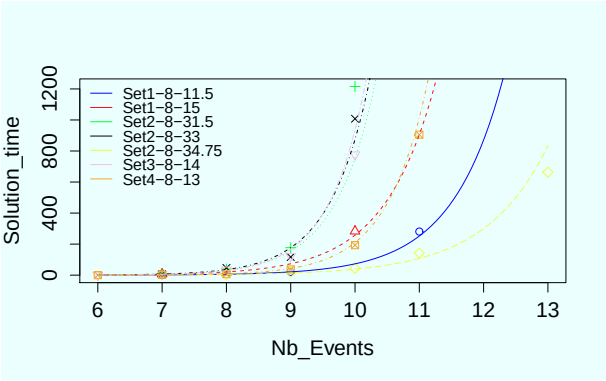


Fig. 1 Solution time of event-based formulations *vs.* number of events

hybrigd MILP/CP large neighborhood search heuristic, as the one proposed for the MISTA challenge 2013 on the multi-mode RCPSP [18].

References

1. W. Herroelen, B. D. Reyck, E. Demeulemeester, Resource-constrained project scheduling: A survey of recent developments, Computers & Operations Research 25 (4) (1998) 279 – 302.
2. P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, European Journal of Operational Research 112 (1) (1999) 3 – 41.
3. P. Brucker, S. Knust, Complex Scheduling, GOR-Publications, Springer, 2011.
4. C. Artigues, S. Demassey, E. Néron, Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications, Wiley, 2010. doi:10.1002/9780470611227.
5. S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem, European Journal of Operational Research 207 (1) (2010) 1 – 14.
6. H. Wang, T. Li, D. Lin, Efficient genetic algorithm for resource-constrained project scheduling problem, Transactions of Tianjin University 16 (5) (2010) 376–382.
7. N. Nouri, S. Krichen, T. Ladhari, P. Fatimah, A discrete artificial bee colony algorithm for resource-constrained project scheduling problem, in: 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO), 2013, pp. 1–6. doi:10.1109/ICMSAO.2013.6552557.
8. O. Koné, C. Artigues, P. Lopez, M. Mongeau, Event-based MILP models for resource-constrained project scheduling problems, Computers & Operations Research 38 (1) (2011) 3 – 13.
9. N. M. Najid, M. Arroub, An efficient algorithm for the multi-mode resource constrained project scheduling problem with resource flexibility, International Journal of Mathematics in Operational Research 2 (2010) 748–761.
10. R. H. Möhring, Minimizing costs of resource requirements in project networks subject to a fixed completion time, Operations Research 32 (1) (1984) pp. 89–120.
11. T. A. Guldemond, J. L. Hurink, J. J. Paulus, J. M. J. Schutten, Time-constrained project scheduling, J. Scheduling 11 (2008) 137–148.
12. J. L. Hurink, A. L. Kok, J. J. Paulus, J. M. J. Schutten, Time-constrained project scheduling with adjacent resources, Computers & Operations Research 38 (1) (2011) 310 – 319.
13. U. Dorndorf, E. Pesch, T. Phan-Huy, A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints, Management Science 46 (10) (2000) 1365–1384. arXiv:http://pubsonline.informs.org/doi/pdf/10.1287/mnsc.46.10.1365.12272,

doi:10.1287/mnsc.46.10.1365.12272.
URL http://pubsonline.informs.org/doi/abs/10.1287/mnsc.46.10.1365.12272

14. A. A. B. Pritsker, L. J. Watters, P. M. Wolfe, Multiproject scheduling with limited resources: A zero-one programming approach, Management Science 16 (1) (1969) pp. 93–108.
15. R. Alvarez-Valdes, J. M. Tamarit, The project scheduling polyhedron: Dimension, facets and lifting theorems, European Journal of Operational Research 67 (1993) 204–220.
16. C. Artigues, P. Michelon, S. Reusser, Insertion techniques for static and dynamic resource-constrained project scheduling, European Journal of Operational Research 149 (2) (2003) 249 – 267.
17. J. C. Zapata, B. M. Hodge, G. V. Reklaitis, The multimode resource constrained multi-project scheduling problem: alternative formulations, AIChE Journal 54(8) (2008) 2101 – 19.
18. C. Artigues, E. Hébrard, MIP relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem, in: 6 th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Ghent, Belgium, 2013, pp. 814–819.

MISTA 2015

Multiprocessor Scheduling with Inserted Idle Time to Minimize the Maximum Lateness

N.S. Grigoreva

1 Introduction

The problem of minimizing the maximum lateness while scheduling tasks to parallel identical processors is a classical combinatorial optimization problem. Following the 3-field classification scheme proposed by Graham *et al.* [1], this problem is denoted by $P|r_j|L_{max}$. This problem relates to the scheduling problem [2], it has many applications, and it is NP-hard [3]. A lot of research in scheduling has concentrated on the construction of nondelay schedule. A nondelay schedule has been defined by Baker[4] as a feasible schedule in which no processor is kept idle at a time when it could begin processing a task. An inserted idle time schedule (IIT) has been defined by J.Kanet and V.Sridharam [5] as a feasible schedule in which a processor is kept idle at a time when it could begin processing a task. J.Kanet and V.Sridharam [5] reviewed the literature with problem setting where IIT scheduling may be required. Most of papers considered problem with single processor. In [6] we considered scheduling with inserted idle time for m parallel identical processors and proposed branch and bound algorithm for multiprocessor scheduling problem with precedence-constrained tasks. The goal of this paper is to propose IIT schedule for $P|r_j|L_{max}$ problem. We propose an approximate IIT algorithm named EDD/IIT (earliest due date/ inserted idle time) and branch and bound algorithm, which produces a feasible IIT(inserted idle time) schedule for a fixed maximum lateness L. The algorithm may be used in a binary search mode to find the smallest maximum lateness. A new method for evaluating partial solutions, selecting the next task and new ways of reducing the exhaustive search was designed. To illustrate the effectiveness of this approach we tested it on randomly generated set of tasks.

We consider a system of tasks $U = \{u_1 u_2, \ldots, u_n\}$. Each task is characterized by its execution time $t(u_i)$, its release time $d(u_i)$ and its due dates $D(u_i)$. Release time $d(u_i)$ - the time at which the task is ready for processing and due dates $D(u_i)$ specifies the time limit by which should be completed. Set of tasks is performed on parallel identical processors. Any task can run on any processor and each processor can perform no more than one task at a time. Task preemption is not allowed.

N.S. Grigoreva
Department of Mathematics and Mechanics, St.Petersburg State University, Russia
E-mail: n.s.grig@gmail.com

A schedule for a task set U is the mapping of each task $u_i \in U$ to a start time $\tau(u_i)$ and a processor $num(u_i)$. Maximum lateness of schedule S is the quantity

$$L_{\max} = \max\{\tau(u_i) + t(u_i) - D(u_i) | u_i \in U\}.$$

First, we propose an approximate IIT algorithm named EDD/IIT (earliest due date/ inserted idle time). Then by combining the EDD/IIT algorithm and $B\&B$ method this paper presents BB/IIT algorithm which can find optimal solutions for multiprocessor scheduling problem.

2 Approximate algorithm EDD/IIT

For each task u_i, we know the earliest starting time $r(u_i)$ and the latest start time $v_{max}(u_i) = D(u_i) - t(u_i)$, which is a priority of task. Let k tasks have been put in the schedule and partial schedule S_k have been constructed.

Let be $time_k[i]$ the time of the termination of the processor i after completion all its tasks. The approximate schedule is constructed by EDD/IIT algorithm as follows:

1. Determine the processor l_0 such as $t_{\min}(l_0) = \min\{time_k[i] | i \in 1..m\}$.
2. Select the task u_0, such as $v_{max}(u_0) = \min\{v_{max}(u_i) | u_i \notin S_k\}$.
3. If $idle(u_0) = r(u_0) - t_{min}(l_0) > 0$ then choose a task $u^* \notin S_k$, which can be executed during the idle time of the processor l_0 without increasing the start time of the task u_0, namely $v_{max}(u^*) = \min\{v_{max}(u_i) | r(u_i) + t(u_i) \leq r(u_0), u_i \notin S_k\}$.
4. If the task u^* is found, then we assign to the processor l_0 the task u^*, otherwise the task u_0.

3 Branch and bound method for constructing a feasible schedule

The branch and bound algorithm produces a feasible IIT(inserted idle time) schedule for a fixed maximum lateness L. In order to optimize over L we must iterate the scheduling process over possible values of L. We recalculate due dates $D(u_i)$ as $D^*(u_i) = D(u_i) + L$, makespan $T_S = \max\{D^*(u_i) | u_i \in N\}$ and the latest start times $v_{max}(u_i) = D^*(u_i) - t(u_i)$.

In order to describe the branch and bound method, it is necessary to determine the set of tasks that we need to add to a partial solution, the order in which task will be chosen from this set and the rules that will be used for eliminating partial solutions.

Let I be the total idle time of processors in the feasible schedule S of length T_S for m processors, then $I = m \cdot T_S - \sum_{i=1}^{n} t(u_i)$. For a partial solution σ_k we know $idle(u_i)$— idle time of processor before start the task u_i.

At each level k will be allocated a set of tasks U_k, which we call the the ready tasks. These are tasks that need to add to a partial solution σ_{k-1}, so check all the possible continuation of the partial solutions.

Definition 1 Task $u \notin \sigma_k$ is called the ready task at the level k, if $r(u)$ satisfies the inequality $r(u) - t_{min}(k) \leq I - \sum_{i=1}^{k} idle(u_i)$.

The main way of reducing of the exhaustive search will be the earliest possible identification unfeasible solutions.

Definition 2 Let the task $u_{cr} \notin \sigma_k$ is such as $v_{max}(u_{cr}) = \min\{v_{max}(u) | u \notin \sigma_k\}$. The task $u_{cr} \notin \sigma_k$ is called the delayed task for σ_k, if $v_{max}(u_{cr}) < t_{min}(k)$.

Obviously if delayed task u_{cr} for a partial solution σ_k exists, then a partial solution σ_k is unfeasible. Below we formulated the rules, which allow deleting a lot of unfeasible partial solutions.

Lemma 1 *Let delayed task u_{cr} for a partial solution $\sigma_k = \sigma_{k-1} \cup u_k$ exists, then*

1. *for any task u, such as $\max\{t_{min}(k-1), r(u)\} + t(u) > v_{max}(u_{cr})$ a partial solution $\sigma_{k-1} \cup u$ is unfeasible;*
2. *if $t_{min}(k-1) + t(u_{cr}) > v_{max}(u_k)$, then the partial solution σ_{k-1} is unfeasible.*

Another method for determining unfeasible partial solutions bases on a comparison of resource requirements of tasks and total processing power. In this case, we propose to modify the algorithm for determining the interval of concentration [7] for the complete schedule and to apply this algorithm to a partial schedule σ_k.

4 Computation results and Conclusions

To test the approximate EDD/IIT algorithm and BB/IIT algorithm, we conducted computational experiment. The quality of the solutions we estimated average ratio of the solution value over the lower bound of the maximum lateness LB. To illustrate the effectiveness of our algorithms we tested random generated problems of up to 200 tasks. We solved these problems with time limit of 60 seconds.

The solution generated with EDD/IIT are on average only 8,1 percent away from the optimal value and this deviation is never more than 11 %. We compared different rules of deleting infeasible partial solutions and received that condition of lemma 1 allows deleting about 43 percent unfeasible solutions. Optimal solutions were obtained for 63% of the cases tested by BB/IIT . The average percentage deviation from lower bound varies between 0.5 and 6.1 %.

In this work, we proposed a new branch and bound method for solving the multiprocessor scheduling problem of maximum lateness minimization. We also presented a new approximate IIT (inserted idle time) algorithm. We found that the problem could be solved within reasonable time for moderate-size systems. Results of our experiment indicated that BB/IIT consistently produces high-quality solutions on the larger randomly generated set tasks.

References

1. J. R.L.Graham, E.L.Lawner and R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey, *Ann. of Disc. Math.* **5 (10)**, pp. 287-326, (1979).
2. P. Brucker. *Scheduling Algorithms*, (1997).
3. J. Ullman. NP-complete scheduling problems *J. Comp. Sys. Sci.* **171**, pp.394-394, (1975).
4. K.R.Baker. *Introduction to Sequencing.* John Wiley & Son, New York(1974).
5. J. Kanet and V. Sridharan. Scheduling with inserted idle time:problem taxonomy and literature review, *Oper.Res* **48 (1)**, pp.99-110, (2000).
6. N.S.Grigoreva. *Branch and bound method for scheduling precedence constrained tasks on parallel identical processors*, in *Lecture Notes in Engineering and Computer Science: Proc. of The World Congress on Engineering 2014, WCE 2014, 2-4 July, 2014*, London, U.K., pp. 832–836.
7. E.Fernandez and B.Bussell. Bounds the number of processor and time for multiprocessor optimal schedules, *IEEE Tran. on Comp.* **4 (11)**, pp. 745-751 (1973).

MISTA 2015

Using Adaptive Large Neighborhood Search to Solve Parallel Machine Scheduling Problems with Dedications and Unequal Ready Times of the Jobs

Raphael Herding • Lars Mönch

1 Introduction and Problem Setting

Scheduling problems for parallel machines are important building blocks for flexible flow shop and job shop scheduling problems when decomposition techniques are applied. Applications of such scheduling problems can be found in wafer fabs that can be modeled as complex job shops with many machine groups [4]. A machine group can contain up to 60 machines in parallel. Each job can often be processed only on a subset of these machines due to quality reasons, i.e., we have machine dedications. We discuss a scheduling problem for identical parallel machines with unequal ready times of the jobs and machine dedications. The performance measure is the total weighted tardiness (TWT). It is the summation of the weighted tardiness $w_j T_j$ over all jobs $j = 1,\ldots,n$, where $T_j := \max(C_j - d_j, 0)$. Here, w_j is the weight of job j, C_j is the completion time, p_j the processing time, and d_j the due date. Using the $\alpha \mid \beta \mid \gamma$ notation, the researched problem can be represented as follows:

$$P \mid M_j, r_j \mid TWT, \tag{1}$$

where P refers to identical parallel machines and r_j is the ready time of job j. The notation M_j indicates that job j can be processed only on a subset of all machines $\{1,\ldots,m\}$. Problem (1) is NP-hard since it contains the NP-hard problem $1 \| TWT$ as a special case. In the present paper, we compare adaptive large neighborhood search (ALNS) with a list scheduling heuristic based on the Apparent Tardiness Cost (ATC) dispatching rule and with a genetic algorithm (GA) that incorporates problem-specific dominance rules. LNS approaches are among the best performing heuristics for vehicle routing problems (VRPs) [7]. Similar to VRPs, parallel machine scheduling problems are partition problems. Therefore, we expect an excellent performance of ALNS. There are only a few papers that deal with LNS approaches in machine scheduling. The papers [5,9] study flexible job shop problems, while parallel machine scheduling problems are considered in [1,8]. However, the objectives and constraints in these papers are different from the ones used in problem (1).

Raphael Herding
University of Hagen
E-mail: Raphael.Herding@fernuni-hagen.de

Lars Mönch
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

2 Genetic Algorithm and a Simple Reference Heuristic

The GA uses a job-based representation that assigns one of the machines from the set M_j to each job j. The resulting single machine scheduling problems have to be solved in order to compute the fitness of a chromosome. Therefore, we apply the Combined Priority Rule for Total Weighted Tardiness (CPRTWT) together with heuristics based on dominance properties that improve the sequence of jobs on each single machine as described in [2].

It is well known that the ATC rule provides high-quality solutions for scheduling problems with TWT objective. Whenever a machine becomes available the next job is selected according to the ATC index

$$I_j(t) := \frac{w_j}{p_j} exp\left(-\frac{\left(d_j - p_j - \max\left(r_j, t\right)\right)}{\kappa \bar{p}} \right), \qquad (2)$$

from the set of jobs that are not already scheduled where κ is a look-ahead parameter, and $\bar{p}$ is the average processing time of the jobs to be scheduled. The quantity t refers to the point of time when the decision is made. The performance of the ATC rule strongly depends on the choice of κ. Therefore, we select κ from the grid $0.1k, k = 1,\ldots,50$.

3 ALNS and LNS Approaches

The ALNS framework is proposed in [6,7]. A set of destroy and repair methods compete in each iteration to improve the current solution. We use the following destroy methods to tailor ALNS to problem (1):

1. **Random destroy** $rd(x,k)$: k jobs are randomly removed from a feasible schedule x.
2. **Worst destroy** $wd(x,k)$: Starting from a feasible schedule x, each job is considered. The TWT value of the schedule with and without the corresponding job is compared. The job that leads to the largest TWT reduction is marked for removal. This procedure is repeated $k \le n$ times for all unmarked jobs. The k marked jobs are removed from x.
3. **Cluster destroy** $cd(x,q_1,q_2)$: Starting from a feasible schedule x, the jobs that are at the positions $q_1 \le p \le q_2$ on the machines are removed. This method is based on the intuition that jobs that are clustered based on their positions are exchangeable without a major TWT increase because their ready times are similar.

The following repair methods are applied:

1. **Dispatching rule-based repair** $drr(\tilde{x},d,S)$: The $k := card(S)$ jobs of the set S to be inserted into the partial schedule $\tilde{x}$ are sequenced according to the dispatching rule d. Then the job of S with the largest priority index is inserted at a feasible position in $\tilde{x}$ that leads to the smallest increase in TWT, while respecting the machine dedication. This procedure is repeated until all the k jobs are inserted. We use the ATC, WSPT, EDD, and COVERT dispatching rule as sequencing rules.
2. **Random repair** $rr(\tilde{x},S,\lambda)$: One of the $k := card(S)$ jobs of the set S to be inserted into the partial schedule $\tilde{x}$ is randomly chosen and inserted into the schedule at a randomly selected position taking into account the machine dedications. This step is repeated k times. The entire procedure is independently repeated λ times. The best solution is chosen.
3. **Position-based repair** $pr(\tilde{x},S)$: We start by choosing $k := card(S)$ positions on the machines. The machines are sorted in non-increasing order of the number of selected positions on them. Starting from the last position on the first machine of the sorted list, all jobs from S that can be processed on this machine will be tested for the TWT

increase caused by inserting the job at the corresponding position. The job with the smallest TWT increase will be inserted in $\tilde{x}$. This procedure is repeated until all positions are covered. When jobs from S cannot be inserted at open positions due to machine dedications then these remaining jobs will be inserted using random repair.

The random destroy method removes around on third of all jobs, while the worst destroy method removes only one fifth of all jobs. The cluster destroy removes the last fourth of the jobs on each machine. A weight is assigned to each destroy and repair method. It controls how often a method is attempted during the search. These weights are used by a roulette wheel method to choose destroy and repair methods for each single iteration. We consider also a LNS approach for comparison reasons. The LNS scheme applies random destroy and ATC-based repair.

4 Computational Results

We generate problem instances with machine dedications according to a design similar to the one in [3]. We consider a first set of 27 small-size instances with two machines and ten jobs where we know the optimal TWT values from complete enumeration. It turns out that GA, LNS, and ALNS are all able to find optimal solutions when 5 minutes of computing time per problem instance are available. ALNS is the quickest algorithm that needs only 30 seconds per problem instance to determine optimal solutions. A second set of 162 medium- and large-size problem instances with up to 90 jobs on up to six machines is used to assess the performance of the different heuristics. The corresponding computational results depending on the available computing time per problem instance can be found in Table 1. We show the TWT values of the corresponding metaheuristic relative to the TWT values obtained by list scheduling using the ATC dispatching rule.

Table 1: Computational Results for Problem (1)

Time	**1 min**			**2 min**			**5 min**		
Compare	**ALNS**	**LNS**	**GA**	**ALNS**	**LNS**	**GA**	**ALNS**	**LNS**	**GA**
m									
3	0.319	0.351	0.597	0.314	0.344	0.572	0.310	0.338	0.554
6	0.507	0.530	0.655	0.499	0.521	0.610	0.492	0.513	0.589
n									
30	0.605	0.605	0.728	0.605	0.605	0.728	0.604	0.604	0.728
60	0.365	0.384	0.581	0.362	0.379	0.562	0.359	0.375	0.561
90	0.340	0.380	0.611	0.332	0.370	0.569	0.326	0.361	0.537
α									
0.25	0.404	0.414	0.646	0.401	0.410	0.612	0.395	0.407	0.591
0.50	0.333	0.365	0.582	0.323	0.356	0.557	0.321	0.350	0.539
0.75	0.368	0.464	0.575	0.361	0.445	0.539	0.357	0.427	0.530
β									
0.25	0.411	0.435	0.652	0.406	0.427	0.627	0.402	0.424	0.601
0.50	0.328	0.365	0.573	0.319	0.359	0.533	0.314	0.347	0.525
1.00	0.289	0.325	0.471	0.281	0.312	0.428	0.278	0.301	0.419

We can see from Table 1 that the GA is outperformed by LNS, while ALNS provides better solutions than LNS. The advantage of ALNS over the two other metaheuristics can be observed even for the smallest amount of computing time. The GA as a population-based approach clearly needs more computing time, but it is not competitive with the remaining metaheuristics even for a computing time of five minutes per problem instance. A small number of machines, a large number of jobs, moderate-spread ready times indicated by

moderate values of α, and wide due dates represented by large values of β lead to the largest TWT improvements compared to list scheduling.

In future work, we are interested in carrying out more detailed computational experiments and in designing mixed integer programming-based repair methods.

References

1. Gacias, B, Artigues, C., Lopez, P. Parallel machine scheduling with precedence constraints and setup times. Computers & Operations Research, 37(12) 2141–2151 (2010)
2. Jouglet, A., Savourey, D., Carlier, J., Baptiste, P. Dominance-based heuristics for one-machine total cost scheduling problems. European Journal of Operational Research, 184, 879-899 (2008).
3. Mönch L., Balasubramanian, H., Fowler, J., Pfund, M., Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers & Operations Research, 32(11), 2731-2750 (2005)
4. Mönch, L., Fowler, J. W., Dauzère-Pérès, S., Mason, S. J., Rose, O. Scheduling semiconductor manufacturing operations: problems, solution techniques, and future challenges. Journal of Scheduling, 14(6), 583-595 (2011)
5. Pacino, D., Van Hentenryck, P. Large neighborhood search and adaptive randomized decompositions for flexible job shop scheduling. Proceedings of the Twenty-second International Conference on Artificial Intelligence, 1999-2003 (2011)
6. Pisinger, D., Ropke, S. Large neighborhood search. Handbook of Metaheuristics, Springer, Gendreau, M., Potvin, J.-Y. (eds.), 399-420 (2010)
7. Ropke, S., Pisinger, D. A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8), 2403-2435 (2007)
8. Wang, P., Reinelt, G., Tan, Y. Self-adaptive large neighborhood search algorithm for parallel machine scheduling. Journal of Systems Engineering and Electronics, 23(2), 208-2015 (2012)
9. Yuan, Y., Xu, H. An integrated search heuristic for large-scale flexible job shop scheduling problems. Computers & Operations Research, 40(12), 2864-2877 (2013)

MISTA 2015

On Task Scheduling Policies for Work-Stealing Schedulers

Steven Adriaensen · Yasmin Fathy · Ann Nowé

1 Background

Parallel computing architectures are becoming more and more mainstream. To take advantage of their parallel processing capabilities, a computation must be divided in a set of interdependent tasks that can be executed in parallel [7]. Such computation can be represented by a Directed Acyclic Graph (DAG), where vertices are the instructions and edges represent execution order dependencies. Instructions that do not depend on each other can be executed in parallel. In this article we consider the fork-join model of computation. Here, fork instructions generate a new (sub)task that can be processed independently (out-degree of 2). Join instructions cause a task to wait for the completion of another (in-degree of 2). All other instructions have an in/out-degree of at most 1 and make up the task. Typically a computation can be divided in many more tasks than there are processing units. This gives rise to the task scheduling problem: *Which tasks are to be executed by which processor, and in which order?*

Work-stealing [2] is a state-of-the-art dynamic, distributed scheduling algorithm. Here, each worker maintains its own local work-pool. One of the workers starts processing the root task. When a fork is encountered, it places one of the tasks in its pool and continues to process the other. If a task stalls (join) or is completed (out-degree 0) the worker will start working on another task from its own pool, or, if it is empty, it will steal work from another worker's pool. Many work-stealing implementations exist [1,5,4], ranging from libraries to runtimes of parallel programming languages. These systems make different design decisions that impact their performance in subtle ways, causing them to perform well in some settings, and poorly in others.

Steven Adriaensen
Vrije Universiteit Brussel
E-mail: steven.adriaensen@vub.ac.be

Yasmin Fathy
University of Surrey
E-mail: Y.Fathy@surrey.ac.uk

Ann Nowé
Vrije Universiteit Brussel
E-mail: ann.nowe@vub.ac.be

In this abstract we'll discuss one of these design decisions, i.e. the scheduling policy used. In Section 2 we discuss the choice of scheduling policy, in particular the impact of the structure of computation thereon. Section 3 describes an existing adaptive scheduling policy and its weaknesses. Finally, Section 4 reports our ongoing research attempts towards more general scheduling policies.

2 Choice of Scheduling Policy

When executing a fork the system is faced with a choice, i.e. continue the current or execute the spawned task? The choice a system makes is determined by its scheduling policy. Here, most systems either always continue the current (help-first [5]) or always execute the spawned task (work-first [1]), i.e. use a pure policy.[1] Some systems implement a mixed policy known as SLAW [4] (see Section 3).

Using help-first, a fork is implemented as a call to the scheduler, which creates and stores an object for the spawned task in the work-pool. Using work-first, a fork is implemented as an ordinary function call, which only returns after the subtask is completed. To steal a continuation, the thief modifies the runtime stack (which holds the continuation) of its victim. Using help-first, steals are more efficient, but the overhead is higher than using work-first. Usually only a fraction of the tasks is stolen and therefore work-first implementations tend to be more efficient on average. Furthermore, minimizing overhead is essential to allow fine task granularity [1]. Work-first also has desirable theoretical properties: Let S_1 be the space required by a serial execution, then a parallel execution on P processors using work-first requires at most $S_1 P$ space, which is existentially optimal to within a constant factor [2].

One might wonder, if work-first is more efficient on average and has attractive theoretical properties, why do (the majority of the) systems use help-first? An important reason is that it is easier to implement as a library (without compiler support). Also in some systems ordinary function calls are expensive [6]. In addition, using work-first, recursive forks can cause the runtime stack to overflow and for particular computation structures it fails to distribute work efficiently if the residual parallelism[2] R is low [3]. Consider the extreme, yet common, example of an iterative parallel loop which forks P sequential body computations ($R \approx 1$). To exploit the parallelism of this computation, each worker should process a single body. Using help-first, the first worker executes the loop task, generating P body tasks and each worker steals one of them in *parallel*. Using work-first, the first worker starts processing the first body and the loop task is handed from worker to worker *sequentially*. Here, help-first clearly distributes work more efficiently. Also, as mentioned before, help-first induces a lower cost on stealing. It is therefore tempting to conclude that help-first performs better when R is low, while work-first performs better when R is high (as in [3]). However, we'll argue this not to be true in general. Consider a recursive parallel loop, reversing the argumentation above, work-first will distribute the body tasks much more quickly than help-first. In general, you can *mirror* any fork-join computation where help-first generates work more quickly[3] than work-first. Rather, if R is low and peer workers are idle, we want to execute the sub-computation with the highest parallelism first.

[1] In literature, the help/work-first policies are also known as child/continuation-stealing.

[2] $R = \frac{T_1}{P * T_\infty}$, where T_n is the minimal execution time on n processors

[3] The same holds for memory consumption

3 SLAW

As discussed in previous section, what scheduling policy performs best depends on factors unknown before execution. SLAW is to date, the only policy that dynamically adapts its scheduling policy to avoid stack-overflows (help-first at threshold depth), keep memory consumption within theoretical bounds (work-first when # active tasks exceeds threshold) and efficiently distribute tasks. The latter is achieved by switching policy periodically from help-first to work-first if the number of times the worker was victimized (stolen from) is smaller than the number of tasks generated during last period (i.e. enough work is available). Here, [4] makes the overgeneralizing assumption that work-first is more time/memory efficient and help-first generates work more quickly. When this assumption does not hold, SLAW can be shown to perform poorly, consistently making the wrong choice. Another downside of SLAW is that the choice of the period introduces a tradeoff between the increase in overhead due to frequent policy switching on the one hand, and the adaptiveness of the system on the other.

4 Ongoing Research

We're currently looking into alternative scheduling policies to overcome the weaknesses of SLAW (see Section 3). One mixed policy that shows promise is Anti-Imitation (AI). Using AI the first worker starts using a random pure policy, when stealing a task, the stealer anti-imitates its victim, using the opposite policy. Independently of the random choice of the initial worker, AI manages to distribute work quickly for a wider range of computations than SLAW. Note that any (iterative or recursive) parallel loop task is stolen at most once before generating all body tasks. As policy switching occurs only when stealing a task, it doesn't increase the overhead (unlike SLAW). AI, however, has weaknesses of its own. When one of the pure policies is more time efficient, on average half of the active workers will be using the slower policy (i.e. be slower). Also, memory efficiency and potential stack-overflows are still concerns that need to be addressed.

Acknowledgements
Steven Adriaensen is funded by a Ph.D grant of the Research Foundation Flanders (FWO).

References

1. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An efficient multithreaded runtime system, vol. 30. ACM (1995)
2. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. Journal of the ACM (JACM) **46**(5), 720–748 (1999)
3. Guo, Y., Barik, R., Raman, R., Sarkar, V.: Work-first and help-first scheduling policies for async-finish task parallelism. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pp. 1–12. IEEE (2009)
4. Guo, Y., Zhao, J., Cave, V., Sarkar, V.: Slaw: A scalable locality-aware adaptive work-stealing scheduler. In: Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, pp. 1–12. IEEE (2010)
5. Lea, D.: A java fork/join framework. In: Proceedings of the ACM 2000 conference on Java Grande, pp. 36–43. ACM (2000)
6. Robison, A.: A primer on scheduling fork-join parallelism with work stealing. Tech. Rep. ISO/IEC JTC 1/SC 22/WG 21, The C++ Standards Committee (2014)
7. Sutter, H.: The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobbs journal **30**(3), 202–210 (2005)

Appendix

In this section we present and discuss some motivating, preliminary results. All results were obtained in simulation, using a cost model valided by using it to accurately reproduce prior experiments (more specifically those in [3,4]).

Generalized Loop Benchmark

In this experiment we consider the Generalized Loop Benchmark (GLB), the pseudo-code of which is given in Figure 1. This computation consists of 2 types of tasks:

LOOP TASK: Performs no work, but splits itself up into a Loop and Body task.
BODY TASK: Performs the actual work,[4] but spawns no further tasks.

A parameter p_{left} determines the probability that the loop task is computed in the current thread, rather than the spawned thread. For p_{left} values 0 and 1, GLB reduces to a recursive and iterative parallel loop respectively. In our experiments the # body tasks (n) is taken equal to the number of processing units ($P = 64$), such that $R = 1$.

While rather artificial, this benchmark was chosen as it clearly illustrates the properties of help and work-first w.r.t. work-distribution, discussed in Section 2.

```
procedure LOOP(i,n)
    if i < n then
        with probability p_left do
            FORK BODY(i)
            LOOP(i+1, n)
        otherwise
            FORK LOOP(i+1, n)
            BODY(i)
        JOIN
    else
        BODY(i)
    end if
end procedure
```

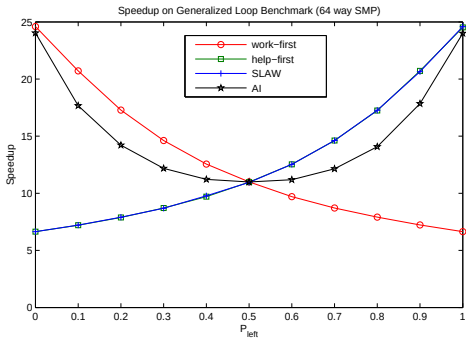


Fig. 1 Code and results for the Generalized Loop Benchmark (GLB)

Observations

Figure 1 shows the Speedup $\frac{T_1}{T_P}$ obtained for GLB, on a 64 way SMP machine, averaged over 1000 runs, using the work-first, help-first, SLAW and AI policies.

We observe that help-first outperforms work-first if and only if $p_{left} > 0.5$, which corresponds exacty to the case where the expected parallelism of the current thread is higher than that of the spawned thread. As R is low, SLAW will always use help-first on this benchmark, failing to distribute work efficiently when $p_{left} < 0.5$. AI on the other hand manages to distribute work reasonably efficiently for all p_{left}, with near oracle performance for $p_{left} \to 0, 1$. Its speedup w.r.t. SLAW ranges from 0.8 to 3.6.

[4] In our experiments, as dummy work, a body task computes a single iteration of the successive over-relaxation (SOR) benchmark on $\frac{1}{P}^{th}$ of a 2000x2000 matrix.

MISTA 2015

Flow Formulation-based Model for the Curriculum-based Course Timetabling Problem

Niels-Christian Fink Bagger · Simon Kristiansen · Matias Sørensen · Thomas R. Stidsen

Abstract In this work we will present a new mixed integer programming formulation for the curriculum-based course timetabling problem. We show that the model contains an underlying network model by dividing the problem into two models and then connecting the two models back into one model using a maximum flow problem. This decreases the number of integer variables significantly and improves the performance compared to the basic formulation. It also shows competitiveness with other approaches based on mixed integer programming from the literature and improves the currently best known lower bound on one data instance in the benchmark data set from the second international timetabling competition.

1 Introduction

Each semester universities face the problem of generating high quality course timetables. A timetable determines when and where a course should take place. The problem of focus in this work is the Curriculum-based Course Timetabling (CCT) Problem from track 3 of the second international timetabling competition (ITC2007) as described by Gaspero et al (2007), in which weekly lectures for multiple courses have to be scheduled and assigned to rooms. A week is divided into days and each day is divided into time slots. A day and time slot combination is referred to as a period. The schedule and room assignment must fulfil some specific hard constraints; all lectures must be scheduled and in different periods, one teacher cannot give two lectures in the same period and a room cannot accommodate two lectures in the same period. Furthermore

Niels-Christian Fink Bagger
Department of Management Engineering, Technical University of Denmark
E-mail: nbag@dtu.dk

Simon Kristiansen
Better Sports ApS

Matias Sørensen
MaCom A/S

Thomas R. Stidsen
Department of Management Engineering, Technical University of Denmark

some courses are grouped into curricula and for each curriculum the courses within cannot be scheduled in the same periods.

Besides the hard constraints there are also soft constraints for which it is wanted to minimize the violation of these. For every lecture it is wanted to be able to accommodate a seat for each student attending. This is denoted as the *RoomCapacity* constraint and when a lecture is scheduled in a room the number of students above the capacity is the counted violation. Each course has a wish for the minimum number of days to spread the lectures across. This is denoted as the *MinimumWorkingDays* constraint and each day below this number in which lectures are not scheduled is counted as one violation. It is wanted to schedule lectures from the same curriculum in adjacent periods. Two periods are considered to be adjacent if they belong to the same day and are in consecutive time slots. If a lecture from a curriculum is scheduled in a period and no lecture from the same curriculum is scheduled in an adjacent period, the lecture is denoted as being *secluded*. Every time there is a secluded lecture this counts as one violation of the *CurriculumCompactness* constraint. Each course should not be assigned to too many different rooms during the week. This is denoted as the *RoomStability* constraint and every room except the first that the course is scheduled in is counted as one violation.

The objective is to find a solution which fulfils all the hard constraints and minimizes a weighted sum of the violations of the soft constraints. The problem will be solved using integer programming and the formulation of the model will be given in Section 2.

2 Mixed Integer Programming Formulation

The problem has been considered using mixed integer programming models before in the literature, see e.g. Burke et al (2010, 2012); Lach and Lübbecke (2012); Cacchiani et al (2013); Hao and Benlic (2011). For an great survey refer to Bettinelli et al (2015). A very common way to formulate the model is to use three-indexed binary variables. Here we will give the formulation similar to the three-indexed formulations from Burke et al (2012) and Lach and Lübbecke (2012). Let C be the set of courses, P be the set of periods and R be the set of rooms. Furthermore there are days D, curricula Q, lecturers L, the periods $P_d \subseteq P$ that belongs to day $d \in D$, the courses $C_q \subseteq C$ which are part of curriculum $q \in Q$ and the courses $C_l \subseteq C$ which are all being taught by lecturer $l \in L$. For each period $p \in P$ we will denote the adjacent periods as $p-1$ and $p+1$ for the periods belonging to the same day as p in the time slot right before and the time slot right after p respectively. When p corresponds to the first (last) time slot on the day, then the period $p-1$ ($p+1$) is undefined and we will define any variable associated with it to always take the value zero.

Let L_c be the number of lectures to be scheduled for course $c \in C$, C_r be the capacity of room $r \in R$, S_c be the number of students attending course $c \in C$ and let $F_{c,p}$ be one if it is allowed to schedule a lecture from course $c \in C$ in period $p \in P$ and zero otherwise. Lastly M_c is the minimum number of days that it is preferred to schedule lectures for course $c \in C$ in.

Let $x_{c,p,r}$ be a binary variable deciding whether to schedule a lecture from course $c \in C$ in period $p \in P$ and room $r \in R$ or not. $t_{c,d}$ is a non-negative variable taking value 1 if course $c \in C$ has at least one lecture at day $d \in D$, and 0 otherwise. w_c is a non-negative variable denoting the number of days below the given minimum that course $c \in$

C has lectures. $z_{c,r}$ is a non-negative variable taking value 1 if course $c \in C$ is occupying room $r \in R$ at least once during the week, and 0 otherwise. κ_c is a non-negative variable counting the number of times that course $c \in C$ is changing room. $s_{q,p}$ is a non-negative variable taking value 1 if curriculum $q \in Q$ has a secluded lecture in period $p \in P$. Let W^{RC}, W^{CC}, W^{WD} and W^{RS} be the weights of the constraints *RoomCapacity*, *CurriculumCompactness*, *MinimumWorkingDays* and *RoomStability* respectively. The formulation is given in Model 1.

$$\min \quad W^{RC} \sum_{c\in C, p\in P, r\in R} (S_c - C_r)^+ \cdot x_{c,p,r} + W^{CC} \sum_{q\in Q, p\in P} s_{q,p} + W^{WD} \sum_{c\in C} w_c + W^{RS} \sum_{c\in C} p_c \tag{1a}$$

$$\text{s. t.} \quad \sum_{p\in P, r\in R} x_{c,p,r} = L_c \qquad \forall c \in C \tag{1b}$$

$$\sum_{r\in R} x_{c,p,r} \le F_{c,p} \qquad \forall c \in C, p \in P \tag{1c}$$

$$\sum_{c\in C} x_{c,p,r} \le 1 \qquad \forall p \in P, r \in R \tag{1d}$$

$$\sum_{c\in C_q, r\in R} x_{c,p,r} \le 1 \qquad \forall q \in Q, p \in P \tag{1e}$$

$$\sum_{c\in C_l, r\in R} x_{c,p,r} \le 1 \qquad \forall l \in L, p \in P \tag{1f}$$

$$t_{c,d} - \sum_{p\in P_d, r\in R} x_{c,p,r} \le 0 \qquad \forall c \in C, d \in D \tag{1g}$$

$$w_c + \sum_{d\in D} t_{c,d} \ge M_c \qquad \forall c \in C \tag{1h}$$

$$\sum_{p\in P} x_{c,p,r} - L_c \cdot z_{c,r} \le 0 \qquad \forall c \in C, r \in R \tag{1i}$$

$$\sum_{r\in R} z_{c,r} - \kappa_c \le 1 \qquad \forall c \in C \tag{1j}$$

$$\sum_{c\in C_q, r\in R} (x_{c,p,r} - x_{c,p-1,r} - x_{c,p+1,r}) \le s_{q,p} \qquad \forall q \in Q, p \in P \tag{1k}$$

$$x_{c,p,r} \in \mathbb{B} \qquad \forall c \in C, p \in P, r \in R \tag{1l}$$

$$z_{c,r} \in \mathbb{B} \qquad \forall c \in C, r \in R \tag{1m}$$

$$0 \le t_{c,d} \le 1 \qquad \forall c \in C, d \in D \tag{1n}$$

$$w_c \ge 0 \qquad \forall c \in C \tag{1o}$$

$$\kappa_c \ge 0 \qquad \forall c \in C \tag{1p}$$

$$s_{q,p} \ge 0 \qquad \forall q \in Q, p \in P \tag{1q}$$

Model 1 A three-index formulation of the CCT problem.

The objective function (1a) consists of the weighted sum of the soft constraint violations and the weights are set according to Gaspero et al (2007):

$$W^{RC} = 1 \tag{1}$$
$$W^{CC} = 2 \tag{2}$$
$$W^{WD} = 5 \tag{3}$$
$$W^{RS} = 1 \tag{4}$$

The constraints (1b) ensures that all lectures of the courses are scheduled. Constraints (1c) ensures that each lecture of a course is scheduled in different periods and only in periods where the course is available. Constraints (1d) make sure that at most one lecture is scheduled in a room in any period. (1e) and (1f) ensures that courses from the same curriculum or taught by the same lecturer is not scheduled in the same periods. The constraints (1g) and (1h) computes which days that the course have been scheduled for lectures and by how much the minimum working days is violated. Constraints (1i) and (1j) calculates which rooms the courses puts into use and how many different rooms they are scheduled in. Lastly the constraints (1k) computes the periods where the curricula have secluded lectures.

2.1 Maximum Flow-based Formulation

The mixed integer programming formulation that we will present here is inspired by the formulation proposed by Lach and Lübbecke (2008, 2012) that consists of decomposing the model into two stages; stage I which is assigning time slots to the courses and stage II which is allocating rooms to the courses based on the assigned time slots from stage I. Instead of solving the two stages separately as Lach and Lübbecke (2008, 2012) we will combine the two stages into one model by using a flow network. This creates new models with a much lower number of integer variables compared to Model 1 at the cost of introducing three-indexed continuous variables. However due to the huge reduction in integer variables (and non-zeros in the constraint matrix) we expect these flow-based models to perform better than Model 1.

At first we will consider only assigning the courses to time slots, i.e. ignore the existence of rooms. This will only account for the *MinimumWorkingDays* and the *CurriculumCompactness* soft constraints. Let $x_{c,t}$ be a binary variable deciding whether to assign course $c \in C$ to time slot $t \in T$ or not. $t_{c,d}$, w_c and $s_{q,t}$ are defined in the same way as for Model 1. The formulation of assigning the courses to time slots is given in Model 2. The description of the objective and constraints follows that of Model 1.

The next step is to consider the room assignment part of the problem. Let $z_{c,r}$ be a binary variable taking value one if course $c \in C$ is allowed to be scheduled in room $r \in R$ and zero otherwise. Let κ_c be a non-negative variable counting the number of times that course $c \in C$ is changing room and let the integer variable $y_{c,r}$ identify the number of times that course $c \in C$ is assigned to room $r \in R$. The formulation is given in Model 3.

Constraints (3d) in Model 3 ensures that for some course $c \in C$ and some room $r \in R$, $z_{c,r}$ is set to one if $y_{c,r} > 0$. Constraints (3c) ensures that the total number of times that a course $c \in C$ is occupying some rooms is equal to the number of lectures to be taught.

$$\min \quad W^{CC} \sum_{q \in Q, p \in P} s_{q,p} + W^{WD} \sum_{c \in C} w_c \tag{2a}$$

$$\text{s. t.} \quad \sum_{p \in P} x_{c,p} = L_c \qquad \forall c \in C \tag{2b}$$

$$x_{c,p} \leq F_{c,p} \qquad \forall c \in C, p \in P \tag{2c}$$

$$\sum_{c \in C_q} x_{c,p} \leq 1 \qquad \forall q \in Q, p \in P \tag{2d}$$

$$\sum_{c \in C_l} x_{c,p} \leq 1 \qquad \forall l \in L, p \in P \tag{2e}$$

$$\sum_{c \in C_q} (x_{c,p} - x_{c,p-1} - x_{c,p+1}) \leq s_{q,p} \qquad \forall q \in Q, p \in P \tag{2f}$$

$$t_{c,d} - \sum_{t \in T_d} x_{c,t} \leq 0 \qquad \forall c \in C, d \in D \tag{2g}$$

$$w_c + \sum_{d \in D} t_{c,d} \geq M_c \qquad \forall c \in C \tag{2h}$$

$$x_{c,p} \in \mathbb{B} \qquad \forall c \in C, p \in P \tag{2i}$$

$$s_{q,p} \geq 0 \qquad \forall q \in Q, p \in P \tag{2j}$$

$$0 \leq t_{c,d} \leq 1 \qquad \forall c \in C, d \in D \tag{2k}$$

$$w_c \geq 0 \qquad \forall c \in C \tag{2l}$$

Model 2 The formulation for assigning only the time slots.

$$\min \quad W^{RC} \sum_{c \in C, r \in R} (S_c - C_r)\, y_{c,r} + W^{RS} \sum_{c \in C} p_c \tag{3a}$$

$$\text{s. t.} \quad \sum_{r \in R} z_{c,r} - p_c \leq 1 \qquad \forall c \in C \tag{3b}$$

$$\sum_{r \in R} y_{c,r} = L_c \qquad \forall c \in C \tag{3c}$$

$$y_{c,r} - L_c \cdot z_{c,r} \leq 0 \qquad \forall c \in C, r \in R \tag{3d}$$

$$y_{c,r} \in \mathbb{N} \qquad \forall c \in C, r \in R \tag{3e}$$

$$z_{c,r} \in \mathbb{B} \qquad \forall c \in C, r \in R \tag{3f}$$

$$p_c \geq 0 \qquad \forall c \in C \tag{3g}$$

Model 3 The formulation ignoring the time aspects and considering only the room stability and the room capacity violations.

If a solution $\overline{x}$ to Model 2 and a solution $\overline{y}$ to Model 3 is given then a new problem emerges; is the combined solution feasible, i.e. is there a feasible mapping from the assigned rooms in $\overline{y}$ to the assigned periods in $\overline{x}$ such that no room is occupied by two courses in the same period and no course is teaching two lectures in the same period. As a first attempt on the flow problem to use for the connection between Model 2 and Model 3 it is tempting to create a graph mapping the course-room assignment $\overline{y}$ into room-period pairs. For each course $c \in C$ and each room $r \in R$ create a node (c, r) and for each room $r \in R$ and period $p \in P$ create a node (r, p). For each course $c \in C$,

room $r \in R$ and period $p \in P$ create an arc from (c, r) to (r, p). Create a source node (u) and a sink node (v) and for each $c \in C$ and $r \in R$ create an arc from node (u) to node (c, r) and for every $r \in R$ and every $p \in P$ create an arc from node (r, p) to node (v).

The capacity on the arc $(r, p) \rightarrow (v)$ for some $r \in R$ and some $p \in P$ is one and always going to be unchanged. The remaining capacities are set based to some solution $(\overline{x}, \overline{y})$ for Model 2 and Model 3. For each course $c \in C$ and room $r \in R$ the capacity of the arc $(u) \rightarrow (c, r)$ is set to $\overline{y}_{c,r}$ and for each $c \in C$, $r \in R$ and $p \in P$ the capacity of the arc $(c, r) \rightarrow (r, t)$ is set to $\overline{x}_{c,p}$. An example of the graph is illustrated in Fig. 1.

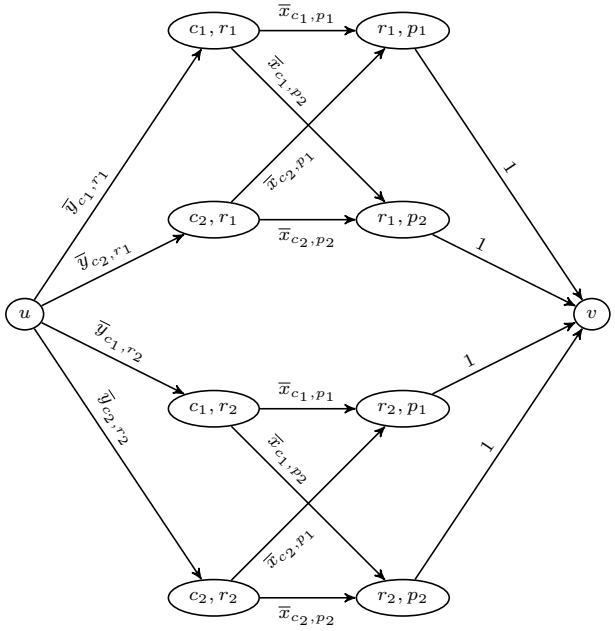


Fig. 1 Illustration of an attempt of the maximum flow graph of an instance with two courses, two rooms and two periods.

For each $c \in C$, $r \in R$ and $p \in P$ the amount of flow on the arc $(c, r) \rightarrow (r, p)$ in the graph in Fig. 1 corresponds to the number of times course c is assigned to room r in period p. Due to the capacities on the arcs at most one amount of flow can go through a node corresponding to a room and period pair (r, p), i.e. at most one course can be assigned to a room $r \in R$ in period $p \in P$. If the maximum flow in this graph is equal to the total sum of lectures of all courses then the solution $(\overline{x}, \overline{y})$ would be identified as being feasible. However, this is not always true. As an example consider an instance with three courses c_1 with one lecture, c_2 with two lectures and c_3 with one lecture, two periods p_1 and p_2, and two rooms r_1 and r_2. Consider a solution $(\overline{x}, \overline{y})$ assumed to be feasible for Model 2 and Model 3 where course c_1 has been assigned one lecture to room r_1 and assigned to period p_1, course c_2 has been assigned one lecture to room r_1 and one lecture to room r_2 and has been assigned both period p_1 and p_2, course c_3 has been assigned to room r_2 and period p_1. The example is illustrated in Fig. 2 where only the arcs with positive capacities are illustrated.

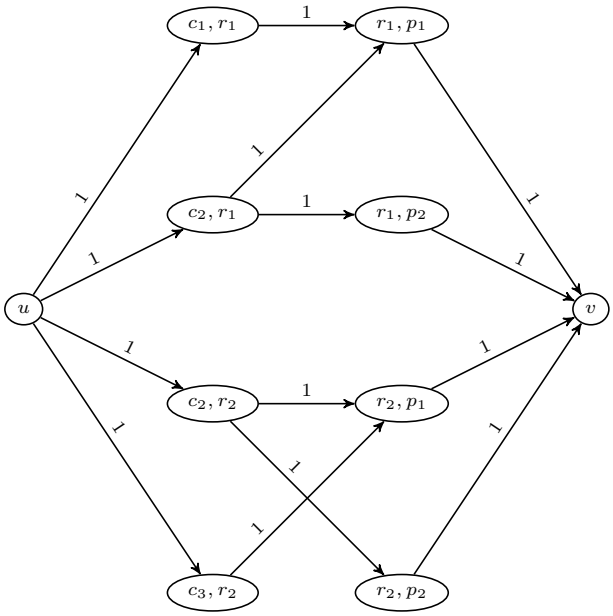


Fig. 2 Illustration of an example of the graph from Fig. 1 with three courses (c_1, c_2 and c_3), two periods (p_1 and p_2) and two rooms (r_1 and r_2). The capacities on the arcs illustrates the room assignments and the period assignments. This graph incorrectly deems the assignments feasible.

In Fig. 2 it can be seen that to get all lectures assigned the flow on the arcs from (u) must all equal the respective capacity. To get the flow out of node (c_1, r_1) it will have to be send to node (r_1, p_1) and to get the flow out of node (c_3, r_2) it will have to be send to node (r_2, p_1). This means that both of the rooms are occupied in period p_1. Since course c_2 has two lectures and there are only two periods then clearly the assignment is infeasible since the course cannot be assigned a room in period p_1. However it is possible to send all the flow through the graph. This is done by sending the flow that comes into node (c_2, r_1) further on to node (r_1, p_2) and sending the flow from node (c_2, r_2) to node (r_2, p_2). By this flow the course c_2 has been assigned two lectures in the same period in two different rooms which is an infeasible assignment for Model 1. However, since the value of the maximum flow is equal to the total number of lectures then this graph is incorrectly stating that the assignments are feasible. Therefore the graph needs to be extended to only allow one unit of flow for each course-period pair. For every room $r \in R$ and period $p \in P$ remove the arc $(r, p) \rightarrow (v)$ and split the node (r, p) into two nodes $(r, p)^1$ and $(r, p)^2$ and add an arc from $(r, p)^1$ to $(r, p)^2$ with a capacity of one. For every course $c \in C$ and period $p \in P$ create a node (c, p), add an arc to node (v) with a capacity of $\overline{x}_{c,p}$ and then for every room $r \in R$ add an arc from node $(r, p)^2$ to node (c, p) with capacity 1. The graph, denoted $\mathcal{G}_{\text{mf}}$, is illustrated in Fig. 3 where the nodes denoted $(r, p)^1$ are to the left in the graph and the nodes denoted $(r, p)^2$ are to the right.

Let the following non-negative variables be defined:

$f^u_{c,r}$: The amount of flow on the arc $(u) \rightarrow (c, r)$.
$f^1_{c,p,r}$: The amount of flow on the arc $(c, r) \rightarrow (r, p)^1$.

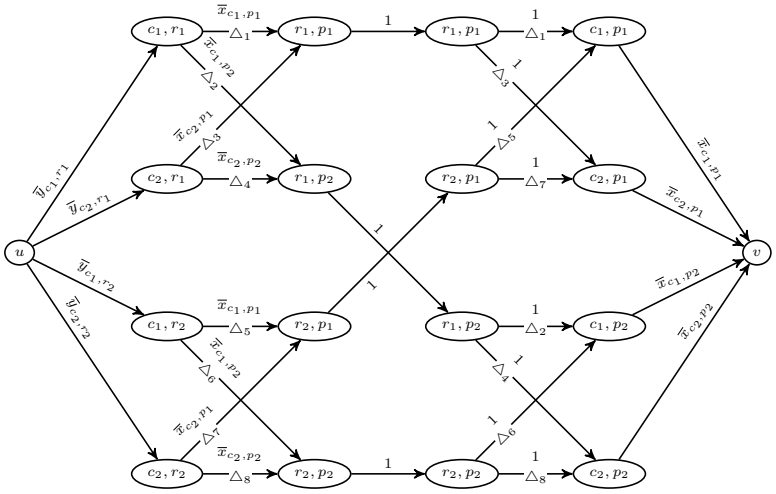


Fig. 3 Illustration of the maximum flow graph of an instance with two courses, two rooms and two periods.

$f_{r,p}$: The amount of flow on the arc $(r,p)^1 \rightarrow (r,p)^2$.
$f^2_{c,p,r}$: The amount of flow on the arc $(r,p)^2 \rightarrow (c,p)$.
$f^v_{c,p}$: The amount of flow on the arc $(c,p) \rightarrow (v)$.

For a course $c \in C$, room $r \in R$ and period $p \in P$ the variable $f^1_{c,p,r}$ is indicating whether course c has a lecture scheduled in period p and room r, but so is the variable $f^2_{c,p,r}$. This means that, for the graph to be correct, if there exists an integer feasible flow f where the total amount of flow is equal to $\sum_{c\in C} L_c$ then there has to exist a flow f' with the same total amount of flow (it is possible that f' is the same flow as f) where $f^1_{c,p,r} = f^2_{c,p,r}$ for every triple (c,p,r). This is illustrated in the graph by marking the pair of arcs with the symbol Δ_i. If two arcs have the same Δ_i symbol then the flow on these two arcs must be equal. This is not taken care of in the standard formulation of the maximum flow problem, but this is not an issue by applying Proposition 1.

Proposition 1 *Let the total amount of flow (the value) of f be denoted $v(f)$ and let A be the set of feasible period-room assignments. Consider the (possibly fractional) maximum flow $f_{\max}$ in $\mathcal{G}_{mf}$ for a given period-room assignment pair $(\overline{x},\overline{y})$. Then we have the following:*

$$v\left(f_{\max}\right) \geq \sum_{c\in C} L_c \iff (\overline{x},\overline{y}) \in A$$

To prove Proposition 1 we will first show that $(\overline{x},\overline{y}) \in A \implies v\left(f_{\max}\right) \geq \sum_{c\in C} L_c$. Next we will show that $v\left(f_{\max}\right) \geq \sum_{c\in C} L_c \implies (\overline{x},\overline{y}) \in A$ by using Proposition 2.

Proposition 2 *Consider some period-room assignment pair $(\overline{x},\overline{y})$ and let $F\left(\mathcal{G}_{mf}\right)$ denote all feasible integer flows in $\mathcal{G}_{mf}$ given this assignment. If there exists a flow $f \in F\left(\mathcal{G}_{mf}\right)$ where $v(f) \geq \sum_{c\in C} L_c$ then there exists a flow $f' \in F\left(\mathcal{G}_{mf}\right)$ where $v(f') = v(f)$ and $f^1_{c,p,r} = f^2_{c,p,r}\ \forall c \in C, p \in P, r \in R$*

The proof of Proposition 2 is given in Appendix A.

Proof (Proof of $(\overline{x},\overline{y}) \in A \implies v(f_{\max}) \geq \sum_{c\in C} L_c$ *from Proposition 1)* Assume that $(\overline{x},\overline{y}) \in A$ and consider some feasible solution for this assignment. Let the variable $f_{c,p,r}$ take value one if course $c \in C$ is assigned to period $t \in T$ and room $r \in R$ in the considered solution. Since we are considering a feasible solution and it is based on the assignment $(\overline{x},\overline{y})$ then the following conditions must be met:

$$\sum_{p\in P} f_{c,p,r} = \overline{y}_{c,r} \quad \forall c \in C, r \in R \tag{5}$$

$$\sum_{c\in C} f_{c,p,r} \leq 1 \quad \forall p \in P, r \in R \tag{6}$$

$$\sum_{r\in R} f_{c,p,r} = \overline{x}_{c,p} \quad \forall c \in C, p \in P \tag{7}$$

We will create a flow f' on the graph $\mathcal{G}_{\text{mf}}$ in the following way: Note that for each course $c \in C$, period $p \in P$ and room $r \in R$ there is a unique path $(u) \to (c,r) \to (r,p)^1 \to (r,p)^2 \to (c,p) \to (v)$ corresponding to the variable $f_{c,p,r}$ and if $f_{c,p,r} = 1$ then we will send one unit of flow on this path, otherwise not. Since we are only considering paths then the node balance constraints must all hold for this flow. Since $\sum_{p\in P} f_{c,p,r} = \overline{y}_{c,r}$ for some course $c \in C$ and room $r \in R$ then the total flow on the arc $(u) \to (c,r)$ is equal to $\overline{y}_{c,r}$ which is the capacity on that arc so the capacity cannot be exceeded. Since $\sum_{c\in C} f_{c,p,r} \leq 1$ for some room $r \in R$ and period $p \in P$ then the total amount of flow on the arc $(r,p)^1 \to (r,p)^2$ cannot exceed one which is the capacity on that arc so the flow is also feasible for this arc. This also means that the flow on the arc $(r,p)^2 \to (c,p)$ cannot exceed one for some course $c \in C$, period $p \in P$ and room $r \in R$ which is the capacity on this arc. Lastly since $\sum_{r\in R} f_{c,p,r} = \overline{x}_{c,p}$ for some $c \in C$ and $p \in P$ then the total flow going through the arc $(c,p) \to (v)$ must be equal to $\overline{x}_{c,p}$ which is the capacity on that arc. Furthermore this also means that the flow on the arc $(c,r) \to (r,p)^1$ can at most be $\overline{x}_{c,p}$ which is the capacity on that arc. This concludes that the flow we created must be a feasible flow for $\mathcal{G}_{\text{mf}}$ with respect to $(\overline{x},\overline{y})$. Since $\sum_{r\in R} f_{c,p,r} = \overline{x}_{c,p}$ for every course $c \in C$ and period $p \in P$ and $\sum_{p\in P} \overline{x}_{c,p} = L_c$ then $\sum_{c\in C, p\in P, r\in R} f_{c,p,r} = \sum_c L_c$ and since each $f_{c,p,r}$ variable corresponds to a path then the total amount of flow $v(f')$ must be equal to $\sum_{c\in C} L_c$. Since f' is a feasible flow then the maximum flow f must have at least the same total amount of flow in $\mathcal{G}_{\text{mf}}$, so we have $v(f) \geq L_c$.

Proof (Proof of $v(f_{\max}) \geq \sum_{c\in C} L_c \implies (\overline{x},\overline{y}) \in A$ *from Proposition 1)* The integrality requirements of the maximum flow problem can be removed from the mathematical model since all capacities are integral (Ahuja et al, 1993, Theorem 6.5). By using (Ahuja et al, 1993, Theorem 6.5) then there must exist a maximum flow f with integer values and if $v(f) \geq \sum_{c\in C} L_c$ then Proposition 2 shows that there must exist an integer maximum flow where $f^1_{c,p,r} = f^2_{c,p,r}$ for every triple (c,p,r). This means that the $f^1_{c,p,r}$ variables and the $f^2_{c,p,r}$ variables describe a feasible assignment based on the solution pair $(\overline{x},\overline{y})$ implying that $(\overline{x},\overline{y}) \in A$.

The LP formulation of the maximum flow model as a feasibility problem, i.e. replacing the objective with a constraint that the value of the flow must be at least the number of lectures, is given in Model 4. We have substituted any occurrence of the flow variables $f^1_{c,p,r}$ and $f^2_{c,p,r}$ in Model 4, since we are only interested in a solution where the two variables are equal, with the non-negative variable $f_{c,p,r}$.

$$\sum_{c\in C, r\in R} f^u_{c,r} \geq \sum_{c\in C} L_c \tag{4a}$$

$$f^u_{c,r} - \sum_{p\in P} f_{c,p,r} = 0 \qquad \forall c \in C, r \in R \tag{4b}$$

$$\sum_{c\in C} f_{c,p,r} - f_{r,p} = 0 \qquad \forall p \in P, r \in R \tag{4c}$$

$$f_{r,p} - \sum_{c\in C} f_{c,p,r} = 0 \qquad \forall p \in P, r \in R \tag{4d}$$

$$\sum_{r\in R} f_{c,p,r} - f^v_{c,p} = 0 \qquad \forall c \in C, p \in P \tag{4e}$$

$$0 \leq f^u_{c,r} \leq \overline{y}_{c,r} \qquad \forall c \in C, r \in R \tag{4f}$$

$$0 \leq f_{c,p,r} \leq 1 \qquad \forall c \in C, p \in P, r \in R \tag{4g}$$

$$0 \leq f_{r,p} \leq 1 \qquad \forall p \in P, r \in R \tag{4h}$$

$$0 \leq f^v_{c,p} \leq \overline{x}_{c,p} \qquad \forall c \in C, p \in P \tag{4i}$$

Model 4 The feasibility flow problem.

Any solution to Model 4 can only fulfil constraint (4a) if the flow send out of the source node on each arc is equal to the capacity so the variable $f^u_{c,r}$ can be replaced with the value $\overline{y}_{c,r}$ in Model 4 which means that constraints (4a) and (4b) are replaced by:

$$\sum_{p\in P} f_{c,p,r} = \overline{y}_{c,r} \quad \forall c \in C, r \in R$$

Constraints (4c) and (4d) and the variable bounds (4h) can be replaced by the constraints:

$$\sum_{c\in C} f_{c,p,r} \leq 1 \quad \forall p \in P, r \in R$$

Finally the constraints (4e) and variable bounds (4g) and (4i) can be replaced by the constraints:

$$\sum_{r\in R} f_{c,p,r} \leq \overline{x}_{c,p} \quad \forall c \in C, p \in P$$

These latter mentioned substitutions together with Model 2 and Model 3 can then be combined into Model 5.

It is not guaranteed that the $f_{c,p,r}$ variables are integers in the solution obtained from Model 5. If the solution returned by the model contains fractional values for the $f_{c,p,r}$ variables then a polynomial algorithm to find an integer feasible solution can be applied. Such an algorithm is given in Algorithm 1 in Appendix A.

3 Computational Results

We have tested the model on the 21 data sets from the ITC2007 competition track 3 described in Gaspero et al (2007). Along the competition a benchmarking tool was provided. The benchmarking tool calculates the amount of time that the algorithms where allowed to run in the competition. This amount of time is usually referred to as

$$\begin{aligned}
\min \quad & W^{RC} \sum_{c,r} (S_c - C_r)^+ \cdot y_{c,r} + W^{CC} \sum_{q \in Q, p \in P} s_{q,p} \\
& + W^{WD} \sum_{c \in C} w_c + W^{RS} \sum_{c \in C} \kappa_c && \text{(5a)} \\
\text{s. t.} \quad & \text{(2b) — (2l)} && \text{(5b)} \\
& \text{(3b) — (3g)} && \text{(5c)} \\
& \sum_{p \in P} f_{c,p,r} = y_{c,r} \quad \forall c \in C, r \in R && \text{(5d)} \\
& \sum_{r \in R} f_{c,p,r} \leq x_{c,p} \quad \forall c \in C, p \in P && \text{(5e)} \\
& \sum_{c \in C} f_{c,p,r} \leq 1 \quad \forall p \in P, r \in R && \text{(5f)} \\
& f_{c,p,r} \geq 0 \quad \forall c \in C, p \in P, r \in R && \text{(5g)}
\end{aligned}$$

Model 5 The combined formulation connecting the period assignments and room assignments using the maximum flow model.

one CPU time unit. We ran the tests in Windows 8.1 on an 3.07GHz Intel® Core™ i7 CPU with 12GB memory. Running the benchmarking tool returned 260 seconds as one CPU unit. All tests has been limited to a single thread.

As mentioned it may be needed to run some flow algorithms on the solutions returned by Model 5. The running times of these algorithms are just a matter of milliseconds even for the largest datasets and so we have neglected these algorithms from the time limits. Furthermore for all our tests the final solutions did not contain any fractional variables so the algorithms were never put to use. If $F_{c,p} = 0$ for some course $c \in C$ and period $p \in P$ then we do not add the variables $x_{c,p}$, $\{f_{c,p,r}\}_{r \in R}$ and $\{x_{c,p,r}\}_{r \in R}$ to the models. This makes the constraints (1c) and (2c) redundant since every course is taught by exactly one lecturer and constraints (1f) and (2e) ensures that each lecturer has at most one lecture scheduled in any period. Furthermore we replace the constraints (1e), (1f), (2d) and (2e) by clique inequalities. This is done by creating a graph where each node corresponds to a course. An edge is connecting two courses if they are in the same curriculum or taught be the same lecturer. We then enumerate all the maximal cliques by running the BronKerbosch algorithm Bron and Kerbosch (1973). Let Γ be the set of cliques and let C_γ be the set of courses in the clique $\gamma \in \Gamma$. Then for each clique $\gamma \in \Gamma$ and period $p \in P$ we add the following constraints to both the basic model and the maximum flow-based formulation:

$$\begin{aligned}
\sum_{c \in C_\gamma, r \in R} x_{c,p,r} \leq 1 \quad & \forall \gamma \in \Gamma, p \in P \\
\sum_{c \in C_\gamma} x_{c,p} \leq 1 \quad & \forall \gamma \in \Gamma, p \in P
\end{aligned}$$

Enumerating all the maximal cliques takes less than a second even for the largest data instances we have tested so we have neglected these enumerations from the time limits when solving the models.

In Table 1 the statistics of the basic model and the maximum flow-based formulation can be seen. For each of the 21 data sets the number of continuous variables, integer variables, constraints and non-zeros is reported.

Table 1: The statistics of the different models of the 21 test instances from ITC2007 track 3; the basic formulation (Basic) and the maximum flow-based formulation (MF). For each data instance and formulation the number of continuous variables (Cont.), the number of integer variables (Int.), the number of rows in the model (Rows) and the number of non-zeros (Non-Zeros) is reported. The number in parenthesis denotes how many of the integer variables that are binary (Bin.).

		Cont.	Int. (Bin.)	Rows	Non-Zeros
comp01	Basic	630	5580 (5580)	2670	61620
	MF	5712	1207 (1027)	2794	29544
comp02	Basic	2324	34112 (34112)	9593	672958
	MF	26916	4161 (2849)	10032	134643
comp03	Basic	2204	29952 (29952)	8628	593908
	MF	24892	3722 (2570)	8973	123222
comp04	Basic	1978	36972 (36972)	8104	528387
	MF	30400	4423 (3001)	8833	141108
comp05	Basic	5436	17982 (17982)	14256	868140
	MF	15993	2145 (1659)	11927	98302
comp06	Basic	2506	50544 (50544)	12333	940504
	MF	39730	5956 (4012)	13223	194715
comp07	Basic	2842	68120 (68120)	15368	1220537
	MF	55002	7848 (5228)	16825	264695
comp08	Basic	2127	40248 (40248)	8186	511003
	MF	32223	4768 (3220)	8897	146461
comp09	Basic	2407	35568 (35568)	8851	604293
	MF	29317	4231 (2863)	9413	137921
comp10	Basic	2480	53820 (53820)	13665	1036975
	MF	41738	6321 (4251)	14593	206834
comp11	Basic	795	6900 (6900)	3855	88395
	MF	7075	1556 (1406)	3910	40313
comp12	Basic	6104	35816 (35816)	22712	1666356
	MF	25904	3736 (2768)	19368	165972
comp13	Basic	2224	40508 (40508)	8114	553425
	MF	32282	4698 (3140)	8937	147523
comp14	Basic	2095	37570 (30570)	9875	699435
	MF	29958	4529 (3084)	10514	148453
comp15	Basic	2204	29952 (29950)	8628	593908
	MF	24892	3722 (2570)	8973	123222
comp16	Basic	2531	56160 (56160)	12599	1013091
	MF	46171	6502 (4342)	13783	220565

Continued on next page

Table 1 – Continued from previous page

		Cont.	Int. (Bin.)	Rows	Non-Zeros
comp17	Basic	2443	43758 (43758)	11050	778699
	MF	35202	5293 (3610)	11836	171848
comp18	Basic	2248	15651 (15651)	7290	304912
	MF	12130	1944 (1521)	6147	62416
comp19	Basic	2168	30784 (30784)	7926	490106
	MF	24168	3743 (2559)	8306	115114
comp20	Basic	2797	59774 (59774)	14067	1103445
	MF	47143	6932 (4633)	15180	228938
comp21	Basic	2608	43992 (43992)	11919	952542
	MF	36574	5271 (3579)	12777	182969
Change	Average	x12.5	x0.1 (x0.1)	x1.0	x0.2
	Min	x2.9	x0.1 (x0.1)	x0.8	x0.1
	Max	x19.4	x0.2 (x0.2)	x1.1	x0.5

It can be seen in Table 1 that the maximum flow-based formulation increases the number of continuous variables on average more that 12 times. However the number of integer variables and non-zeroes in the model is on average a tenth and a fifth respectively compared to the basic model which is why we expect the maximum flow-based formulation to perform better. The model has been solved using the .NET framework provided by Gurobi Optimization (2015) version 6.0.0. The bounds obtained by the maximum flow-based formulation is compared on the first 14 data sets with the following four approaches from the literature:

LL12 An approach based on solving the problem in two stages proposed by Lach and Lübbecke (2012); first assigning the courses into periods and then assigning the first stage assignments into rooms.

BMPR10 A approach based on considering a subset of soft constraints proposed by Burke et al (2010).

HB11 An approach based on partitioning the courses into subsets proposed by Hao and Benlic (2011).

CCRT13 An approach based on splitting the objective into parts proposed by Cacchiani et al (2013).

In Table 2 the lower bounds for the latter mentioned four approaches and the maximum flow-based formulation is reported when running the approaches for one CPU unit (1 T), ten CPU units (10 T) and forty CPU units (40 T). It can be seen that the proposed formulation is able to compete with most of the approaches, except for the proposed method by Cacchiani et al (2013) which seems to perform better on most instances. However the maximum-flow based formulation appears to generate a much better bound on two of the instances; comp05 and comp12. Referring back to Table 1 these are the two only of the first fourteen instances where the formulation actually reduces instead of increasing the number of rows in the model. Furthermore consider Table 5. In this table the number of courses and the number of unavailable time slots are illustrated for each instance. Here it can be seen that the number of unavailable time slots per course is much higher for the two before mentioned instances than for the other of the first fourteen data sets. This can explain why the number of rows is reduced since we did not include the variables of the unavailable periods and so many rows where not added as they were empty.

Since Lach and Lübbecke (2012) and Burke et al (2010) obtain both lower and upper bound these are also compared with the bounds obtained by the maximum flow-based formulation. The results are given in Table 3. Here it can be seen that Burke et al (2010) obtains better lower bounds in most cases for one CPU unit, however for longer running times the maximum flow formulation generates better lower bound on more instances than the other two. As for the upper bounds Lach and Lübbecke (2012) yields better result in more cases than our proposed approach for the short (1 T) and middle (10 T) running time whereas for the long (40 T) running time they yield better upper bounds on an equal amount fo instances making it hard to claim one approach as outperforming the other.

In Table 4 the results of both the basic formulation in Model 1 and the maximum flow based formulation is given. Here it can be seen that the maximum flow formulation clearly outperforms the basic formulation and a new lower bound compared to the best known bound is obtained in one of the instances. This makes the model very interesting as some of the other approaches from the literature based in the basic formulation might also benefit from this reformulation.

	LL12			BMPR10			HB11			CCRT13			MF		
Instance	1 T	10 T	40 T	1 T	10 T	40 T	1 T	10 T	40 T	1 T	10 T	40 T	1 T	10 T	40 T
comp01	4	4	4	0	4	**5**	4	4	4	**5**	**5**	**5**	**5**	**5**	**5**
comp02	0	8	11	0	0	1	**10**	12	12	0	**16**	**16**	0	0	10
comp03	0	0	25	25	33	33	**26**	34	36	24	**52**	**52**	**26**	35	36
comp04	22	28	28	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	23	**35**	**35**
comp05	92	25	108	**119**	111	114	19	69	80	6	6	166	**119**	**171**	**179**
comp06	7	10	10	**13**	**15**	**16**	12	12	**16**	0	11	11	**13**	13	**16**
comp07	0	2	**6**	**6**	**6**	**6**	5	**6**	**6**	0	**6**	**6**	0	**6**	**6**
comp08	30	34	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	27	**37**	**37**
comp09	37	41	46	68	65	66	39	67	67	**92**	**92**	**92**	45	71	76
comp10	2	**4**	**4**	3	**4**	**4**	**4**	**4**	**4**	0	2	2	3	**4**	**4**
comp11	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
comp12	29	32	53	**101**	95	95	43	78	84	0	0	100	85	**115**	**138**
comp13	33	39	41	52	52	54	46	53	55	**57**	**57**	**57**	38	54	56
comp14	40	41	46	**41**	42	42	**41**	43	43	32	**48**	**48**	**41**	42	46
Best	**1**	**2**	**4**	**8**	**6**	**7**	**7**	**5**	**6**	**6**	**10**	**10**	**6**	**8**	**9**
	0	**0**	**0**	**2**	**1**	**0**	**2**	**0**	**0**	**2**	**5**	**5**	**0**	**2**	**2**

Table 2 Comparison of the lower bounds obtained for the different model formulations; Lach and Lübbecke (2012) (LL12), Burke et al (2010) (BMPR10), Hao and Benlic (2011) (HB11), Cacchiani et al (2013) (CCRT13) and the maximum flow based formulation (MF). For each formulation the lower bound is given for one CPU time unit (1 T), ten CPU time units (10 T) and forty CPU time units (40 T). The numbers reported in bold font are the values where the specific models obtained a value which is at least as good as the other formulations. The numbers underlined are the values where the specific models obtained a value which is the better that for the other formulations.

	LL12						BMPR10						MF					
	1 T		10 T		40 T		1 T		10 T		40 T		1 T		10 T		40 T	
Instance	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB
comp01	4	12	4	12	4	12	0	168	4	10	**5**	9	**5**	**5**	**5**	**5**	**5**	**5**
comp02	**0**	239	**8**	93	**11**	**46**	**0**	**114**	0	101	1	63	**0**	253	0	**74**	10	54
comp03	0	194	0	**86**	25	**66**	25	**158**	33	144	33	123	**26**	228	**35**	115	**36**	84
comp04	22	**44**	28	41	28	38	**35**	153	**35**	**36**	**35**	36	23	123	**35**	38	**35**	**35**
comp05	92	965	25	**468**	108	**368**	**119**	1447	111	649	114	629	**119**	**515**	**171**	505	**179**	377
comp06	7	395	10	**79**	10	51	**13**	**277**	**15**	317	**16**	**46**	**13**	897	13	298	**16**	71
comp07	0	**525**	2	**28**	**6**	**25**	**6**	-	**6**	857	**6**	45	0	1095	**6**	215	**6**	58
comp08	30	**78**	34	48	**37**	44	**37**	173	**37**	53	**37**	41	27	195	**37**	**44**	**37**	**40**
comp09	37	115	41	**106**	46	**99**	**68**	**112**	65	115	66	105	45	213	**71**	127	**76**	**99**
comp10	2	235	**4**	**44**	**4**	**16**	**3**	**70**	**4**	49	**4**	23	**3**	994	**4**	311	**4**	44
comp11	**0**	7	**0**	7	**0**	7	**0**	288	**0**	12	**0**	12	**0**	**0**	**0**	**0**	**0**	**0**
comp12	29	**1122**	32	657	53	548	**101**	-	95	889	95	785	85	1844	**115**	**507**	**138**	**485**
comp13	33	**98**	39	**67**	41	66	**52**	556	52	92	54	67	38	461	**54**	102	**56**	**65**
comp14	40	**113**	41	**54**	**46**	**53**	**41**	123	**42**	72	42	55	**41**	180	**42**	84	**46**	58
Best	**2**	**6**	**3**	**8**	**6**	**7**	**12**	**5**	**7**	**1**	**7**	**1**	**8**	**3**	**12**	**5**	**13**	**7**
	0	**6**	**1**	**8**	**1**	**6**	**6**	**5**	**1**	**1**	**0**	**1**	**2**	**3**	**6**	**5**	**5**	**6**

Table 3 Comparison of the bounds obtained by Lach and Lübbecke (2012) (LL12), Burke et al (2010) (BMPR10) and the maximum flow based formulation (MF). For each approach the lower bound is given for one CPU time unit (1 T), ten CPU time units (10 T) and forty CPU time units (40 T). The numbers reported in bold font are the values where the approach obtained a value which is at least as good as the other approaches. The numbers underlined are the values where the specific approaches obtained a value which is better than the other approaches.

			Basic						MF					
	Best Known		1 T		10 T		40 T		1 T		10 T		40 T	
Instance	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB
comp01	5	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
comp02	16	24	**0**	308	**2**	139	5	71	**0**	**253**	0	**74**	**10**	**54**
comp03	52	64	25	1415	28	181	28	109	**26**	**228**	**35**	**115**	**36**	**84**
comp04	35	35	22	182	**35**	59	**35**	**35**	**23**	**123**	**35**	**38**	**35**	**35**
comp05	211	284	117	537	141	**484**	149	387	**119**	**515**	**171**	505	**179**	**377**
comp06	27	27	12	1403	12	**135**	14	124	**13**	**897**	**13**	298	**16**	**71**
comp07	6	6	**0**	**354**	**6**	315	**6**	119	**0**	1095	**6**	**215**	**6**	**58**
comp08	37	37	20	**177**	**37**	65	**37**	61	**27**	195	**37**	**44**	**37**	**40**
comp09	96	96	37	272	65	167	68	159	**45**	**213**	**71**	**127**	**76**	**99**
comp10	4	4	0	**256**	**4**	**94**	**4**	56	**3**	994	**4**	311	**4**	**44**
comp11	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
comp12	100	298	59	**1363**	69	717	106	581	**85**	1844	**115**	**507**	**138**	**485**
comp13	59	59	28	**280**	50	130	53	117	**38**	461	**54**	**102**	**56**	**65**
comp14	51	51	39	305	**42**	112	42	96	**41**	**180**	**42**	**84**	**46**	**58**
comp15	52	64	25	1415	28	181	28	109	**26**	**228**	**35**	**115**	**36**	**84**
comp16	18	18	**8**	**329**	8	103	11	101	7	400	**12**	**74**	**13**	**58**
comp17	56	56	24	**335**	30	268	39	155	**33**	450	**42**	**122**	**43**	**105**
comp18	61	61	14	168	22	165	26	103	**20**	**145**	**26**	**88**	**29**	**83**
comp19	57	57	30	**205**	49	143	52	138	**36**	210	**53**	**62**	**57**	**57**
comp20	4	4	**0**	**1169**	**0**	**120**	**0**	**103**	**0**	1215	**0**	972	**0**	**103**
comp21	74	74	15	847	31	258	49	186	**32**	**527**	**54**	**142**	**57**	**122**
Best			**6**	**11**	**9**	**6**	**7**	**4**	**20**	**12**	**20**	**17**	**21**	**21**
			2	**2**	**6**	**2**	**7**	**3**	**2**	**2**	**7**	**2**	**8**	**4**

Table 4 Comparison of the bounds obtained for the basic model (Basic) and the maximum flow-based formulation (MF). For each formulation the bounds are given for one CPU time unit (1 T), ten CPU time units (10 T) and forty CPU time units (40 T). The numbers reported in bold font are the values where the specific models obtained a value which is at least as good as the other formulations. The numbers underlined are the values where the specific models obtained a value which is as least as good as the best known bounds reported by Scheduling and Timetabling Research Group at the University of Udine (2015).

Instance	$\|C\|$	$\|U\|$	$\|U\|/\|C\|$
comp01	30	53	1.8
comp02	82	513	6.3
comp03	72	382	5.3
comp04	79	396	5.0
comp05	54	771	14.3
comp06	108	632	5.9
comp07	131	667	5.1
comp08	86	478	5.6
comp09	76	405	5.3
comp10	115	694	6.0
comp11	30	94	3.1
comp12	88	1368	15.5
comp13	82	468	5.7
comp14	85	486	5.7

Table 5 Illustrating the statistics of the data sets regarding the unavailable periods. For each instance the number of courses ($|C|$), the total number of unavailable periods ($|U|$) and the average number of unavailable periods per course ($|U|/|C|$) is reported.

4 Conclusion

A mixed integer programming model for the curriculum-based course timetabling problem has been proposed with an underlying flow network. It has been shown that the formulation decreases the number of integer variables significantly and provides better results than a traditional three-index formulation. It is also competitive with most of the other mixed integer programming based approaches from the literature and improves one currently best known lower bound on the benchmarking instances from the second international timetabling competition. Some of the approaches from the literature are based on the original three-indexed model and it is believed that these approaches can also benefit from the proposed model.

Acknowledgements The authors would like to thank professor Stephan Røpke, Department of Management Engineering, Technical University of Denmark, and professor Carsten Thomassen, Department of Applied Mathematics and Computer Science, Technical University of Denmark for fruitful discussions on the proof in Appendix A.

A Proof of Proposition 2

For the proof of Proposition 2 we will be considering the graph $\mathcal{G}_{\text{mf}}$ as described in Section 2.1 and a given solution pair $(\overline{x}, \overline{y})$ to Model 2 and Model 3 where $\sum_{p \in P} \overline{x}_{c,p} = \sum_{r \in R} \overline{y}_{c,r} = L_c \, \forall c \in C$ due to constraints (2b) and (3c). Furthermore we will assume that $L_c \leq |P| \; \forall c \in C$. This is a fair assumption to make as the problem is otherwise infeasible. Before the proposition is proved it will be restated here for the sake of completeness.

Proposition 3 (Restatement of Proposition2)
Consider some period-room assignment pair $(\overline{x}, \overline{y})$ and let $F\left(\mathcal{G}_{mf}\right)$ denote all feasible integer flows in $\mathcal{G}_{mf}$ given this assignment. If there exists a flow $f \in F\left(\mathcal{G}_{mf}\right)$ where $v(f) \geq \sum_{c \in C} L_c$ then there exists a flow $f' \in F\left(\mathcal{G}_{mf}\right)$ where $v(f') = v(f)$ and $f^1_{c,p,r} = f^2_{c,p,r} \, \forall c \in C, p \in P, r \in R$

Algorithm 1: EqualPairMaxFlow

Input: The graph $\mathcal{G}_{\text{mf}}$
Output: An integer maximum flow f where $f^1_{c,t,r} = f^2_{c,t,r}$ $\forall c \in C$, $t \in T$, $r \in R$ if $v(f) \geq \sum_{c \in C} L_c$, otherwise nil
Initialize f as the maximum flow in $\mathcal{G}_{\text{mf}}$
if $v(f) < \sum_{c \in C} L_c$ **then**
  return nil
// Iterate over all (c, p, r)-triples to repair any violations
foreach $c \in C$ **do**
  foreach $p \in P$ **do**
    foreach $r \in R$ **do**
      /* Change the flow f by setting $f^2_{c,p,r}$ to the same value as $f^1_{c,p,r}$ */
      $f^2_{c,p,r} \leftarrow f^1_{c,p,r}$
return f

To prove Proposition 2 we will show that Algorithm 1 is correct.

Algorithm 1 starts off by finding a maximum flow f which has integer values. This can be done by some polynomial algorithm, e.g. the Labeling algorithm (Ahuja et al, 1993, proof of Theorem 6.5). If $v(f) < \sum_c L_c$ then the algorithm returns **nil** to indicate that the assignment $(\overline{x}, \overline{y})$ is infeasible which has been proved in Section 2.1 to be the case. If $v(f) \geq \sum_c L_c$ then the algorithm iterates over every triple (c, p, r) and then set the value of the variable $f^2_{c,p,r}$ to the same value as the variable $f^1_{c,p,r}$. When the algorithm is done then clearly the flow still maintains integer values and $f^1_{c,p,r} = f^2_{c,p,r}$ for every triple (c, p, r). What needs to be shown to prove that Algorithm 1 is correct is that the value of the flow is unchanged, the node balancing constraints are not violated and the capacities are not exceeded, i.e. that the the flow after the change remains a feasible flow.

Assuming that Algorithm 1 is correct we can prove Proposition 2.

Proof (Proof of Proposition 2) As Algorithm 1 is correct then Proposition 2 must be true since if $v(f) < \sum_{c \in C} L_c$ the algorithm returns that the assignment $(\overline{x}, \overline{y})$ is infeasible and if $v(f) \geq \sum_{c \in C} L_c$ the algorithm will return an integer maximum flow f where $f^1_{c,p,r} = f^2_{c,p,r}$ $\forall c \in C$, $p \in P$, $r \in R$.

To prove that Algorithm 1 is correct we will first show that when considering an integer feasible flow f where $v(f) \geq \sum_{c \in C} L_c$, if there is a violation then it is possible to redirect the flow such that the total number of violations is decreased by at least two as stated in Proposition 4. This means that if we have k violations for the flow f then applying this redirection technique at most $k/2$ times will remove all such violations where k must be less than or equal to $|C| \cdot |P| \cdot |R|$.

Proposition 4 *Consider a flow $f \in F\left(\mathcal{G}_{mf}\right)$ where $v(f) \geq \sum_{c \in C} L_c$. If there exists violations of some course-period-room triples (c, p, r), i.e. that $f^1_{c,p,r} \neq f^2_{c,p,r}$, then it is possible to redirect the flow to another flow $f' \in F\left(\mathcal{G}_{mf}\right)$ where $v(f') = v(f)$ such that the total number of violations is decreased by at least two.*

Proof (Proof of Proposition 4) Consider a flow $f \in F\left(\mathcal{G}_{\text{mf}}\right)$ where $v(f) \geq \sum_{c \in C} L_c$. Assume that there exists violations of some course-period-room triples (c, p, r), i.e. that $f^1_{c,p,r} \neq f^2_{c,p,r}$. Since we know that $\sum_{p \in P} \overline{x}_{c,p} = L_c$ for every course $c \in C$ then for every period $p \in P$ where $\overline{x}_{c,p} = 1$ there must be at least one unit of flow on the arc $(c, p) \rightarrow (v)$ otherwise all the flow from the source cannot get to the sink. Let there be a course-period-room triple (c_1, p_1, r_1) such that $f^1_{c_1,p_1,r_1} \neq f^2_{c_1,p_1,r_1}$. Since the capacity on the arc $(c_1, r_1) \rightarrow (r_1, p_1)^1$ is $\overline{x}_{c_1,p_1}$ which is a binary value and the capacity on the arc $(r_1, p_1)^2 \rightarrow (c_1, p_1)$ is one then two cases can occur:

Case 1 $f^1_{c_1,p_1,r_1} = 0 \wedge f^2_{c_1,p_1,r_1} = 1$

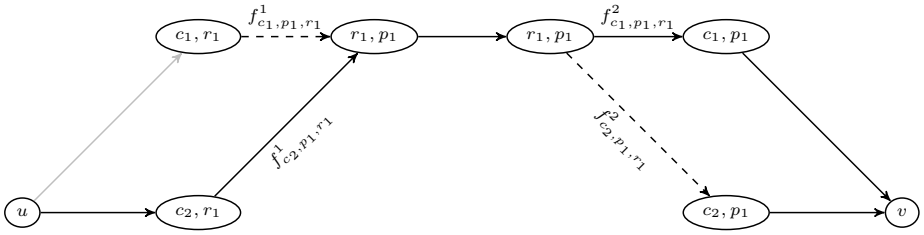


Fig. 4 Illustration of Case 1. The dashed arcs means that there is no flow. The lightly gray arcs correspond to where it is unknown whether there is any flow and the solid black arcs are where there must be at least one unit of flow.

Case 2 $f^1_{c_1,p_1,r_1} = 1 \wedge f^2_{c_1,p_1,r_1} = 0$

Consider first Case 1. This case is illustrated in Fig. 4.

Since $f^2_{c_1,p_1,r_1} = 1$ this means that $f_{r_1,p_1} = 1$ since this is the only way flow can enter the node $(r_1, p_1)^2$ meaning that node $(r_1, p_1)^1$ must be sending out one unit of flow. Since $(r_1, p_1)^1$ is sending out one unit of flow then it must mean that $f^1_{c_2,p_1,r_1} = 1$ for some course $c_2 \in C$. Furthermore since the capacity on the arc $(r_1, p_1)^1 \to (r_1, p_1)^2$ is one and this is the only arc entering $(r_1, p_1)^2$ then it cannot send any units of flow to node (c_2, p_1). This means that Case 1 must contain a triple (c_2, p_1, r_1) for which Case 2 applies. So we can prove the claim for both cases by only considering Case 2.

Consider now Case 2. Since $f^1_{c_1,p_1,r_1} = 1$ then there must be a unit of flow on the arc from $(r_1, p_1)^2$ to (c_2, p_1) for some course $c_2 \in C$. This is illustrated in Fig. 5.

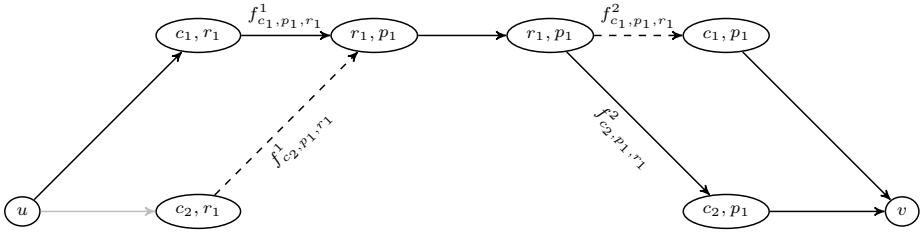


Fig. 5 Illustration of Case 2. The interpretation of the arcs corresponds to Fig. 4.

Due to the construction of the graph the capacity on the arc $(c_1, p_1) \to (v)$ is equal to the capacity on the arc $(c_1, r_1) \to (r_1, p_1)^1$ and must therefore be one since $f^1_{c_1,p_1,r_1} = 1$. This means that we could redirect the flow on the subpath $(r_1, p_1)^2 \to (c_2, p_1) \to (v)$ to the subpath $(r_1, p_1)^2 \to (c_1, p_1) \to (v)$ if there is no flow on the arc $(c_1, p_1) \to (v)$ and maintain an integer feasible flow where the total amount of flow is unchanged. However as latter mentioned the flow on the arc $(c_1, p_1) \to (v)$ must be one, i.e. $f^v_{c_1,p_1} = 1$, since $\overline{x}_{c_1,p_1} = 1$ which means that node (c_1, p_1) must receive one unit of flow from node $(r_2, p_1)^2$ for some room $r_2 \in R$. This case is illustrated in Fig. 6.

Consider the four nodes $(r_1, p_1)^2$, (c_1, p_1), $(r_2, p_1)^2$ and (c_2, p_1) in Fig. 6. As mentioned earlier the capacity on the arc $(r_1, p_1)^2 \to (c_1, p_1)$ must be one. The capacity on the arc $(r_2, p_1)^2 \to (c_2, p_1)$ is set to $\overline{x}_{c_2,p_1}$ which is also the capacity on the arc $(r_1, p_1)^2 \to (c_2, p_1)$ which must be one since there is one unit of flow on the arc $(r_1, p_1)^2 \to (c_2, p_1)$. So swapping the flow on the arc $(r_1, p_1)^2 \to (c_2, p_1)$ with the arc $(r_1, p_1)^2 \to (c_1, p_1)$ and swapping the flow on the arcs $(r_2, p_1)^2 \to (c_1, p_1)$ and $(r_2, p_1)^2 \to (c_2, p_1)$ will maintain an integer feasible flow with an unchanged amount of flow where the Case 2 violation is removed from the triple (c_1, p_1, r_1) and the Case 1 violation is removed from the triple (c_2, p_1, r_1). This swap does

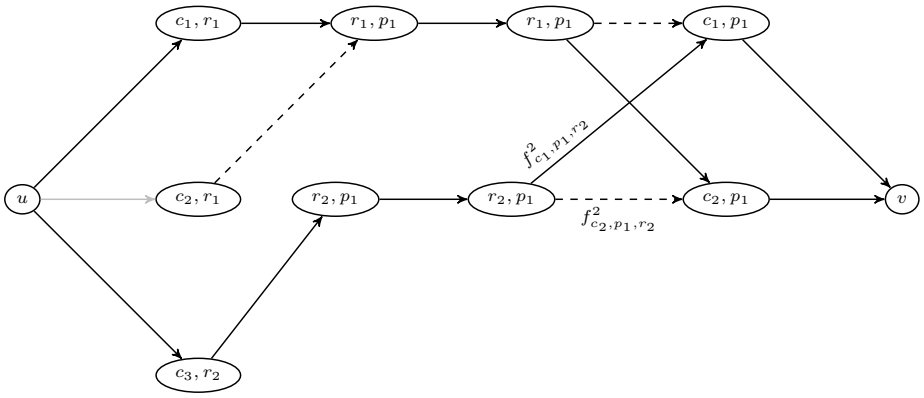


Fig. 6 Illustration of Case 2. The interpretation of the arcs corresponds to Fig. 4.

not introduce new violations when $c_1 \neq c_3$ and so we are done. However if $c_1 = c_3$ then one violation is introduced for the triple (c_1, p_1, r_2) and one for the triple (c_2, p_1, r_2) meaning that making the swaps does not change the number of violations. So we need to show that when $c_1 = c_3$ it is possible to find another place in the graph to make the swap and repair at least two violations.

Since we know that there is flow from the source to the nodes (c_1, r_1) and (c_1, r_2) then we know that $\sum_{r \in R} \overline{y}_{c_1,r} \geq 2$ meaning that $\sum_{p \in P} \overline{x}_{c_1,p} \geq 2$, i.e. that c_1 is teaching at least two lectures. This means that there must exist another period $p_2 \in P : p_1 \neq p_2$ where $\overline{x}_{c_1,p_2} = 1$. As $\overline{x}_{c_1,p_2} = 1$ implies that there is at least one unit of flow on the arc $(c_1, t_2) \rightarrow (v)$ then there must be one unit of flow on the path $(u) \rightarrow (c_4, r_3) \rightarrow (r_3, p_2)^1 \rightarrow (r_3, p_2)^2 \rightarrow (c_1, p_2) \rightarrow (v)$ for some $c_4 \in C$ and $r_3 \in R$ as illustrated in Fig. 7.

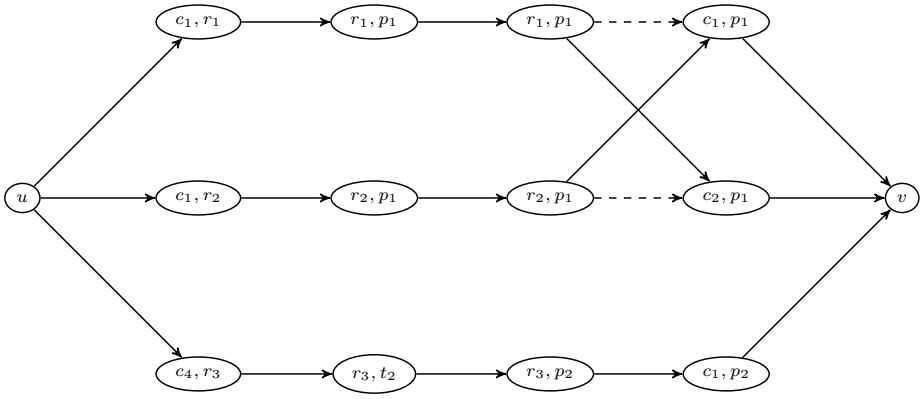


Fig. 7 Illustration of Case 2 where $c_1 = c_3$. The interpretation of the arcs corresponds to Fig. 4.

Suppose that $c_1 = c_4$. Then $\sum_{r \in R} y_{c_1,r} \geq 3$ and there must be some other period that c_1 is assigned to and we can consider that period as p_2 instead. So we must be able to find a period $p_2 \in T$ and a course $c_4 \in C$ such that $p_1 \neq p_2$ and $c_1 \neq c_4$ where there is flow on the path $(u) \rightarrow (c_4, r_3) \rightarrow (r_3, p_2)^1 \rightarrow (r_3, p_2)^2 \rightarrow (c_1, p_2) \rightarrow (v)$ for some $r_3 \in R$. This means that for the triple (c_1, p_2, r_3) there is a Case 1 violation and for the triple (c_4, p_2, r_3) there is

a Case 2 violation. This is exactly the same cases as for the triples (c_2, p_1, r_1) and (c_1, p_1, r_1) and so there must exist a course $c_5 \in C$ and a room $r_4 \in R$ where there is one unit of flow on the path $(u) \to (c_5, r_4) \to (r_4, p_2)^1 \to (r_4, p_2)^2 \to (c_4, p_2) \to (v)$. This means that we have two cases; either $c_4 \neq c_5$ and we can swap the flow on the arcs $(r_3, p_2)^2 \to (c_1, p_2)$ and $(r_3, p_2)^2 \to (c_4, p_2)$ and swap the flow on the arcs $(r_4, p_2)^2 \to (c_4, p_2)$ and $(r_4, p_2)^2 \to (c_1, p_2)$ or $c_4 = c_5$ and we can find a path $(u) \to (c_6, r_5) \to (r_5, p_3)^1 \to (r_5, p_3)^2 \to (c_4, p_3) \to (v)$ where there is one unit of flow and where $c_6 \neq c_4$ and $p_3 \neq p_2$ in the same way as we found c_4 and p_2. This is illustrated in Fig. 8.

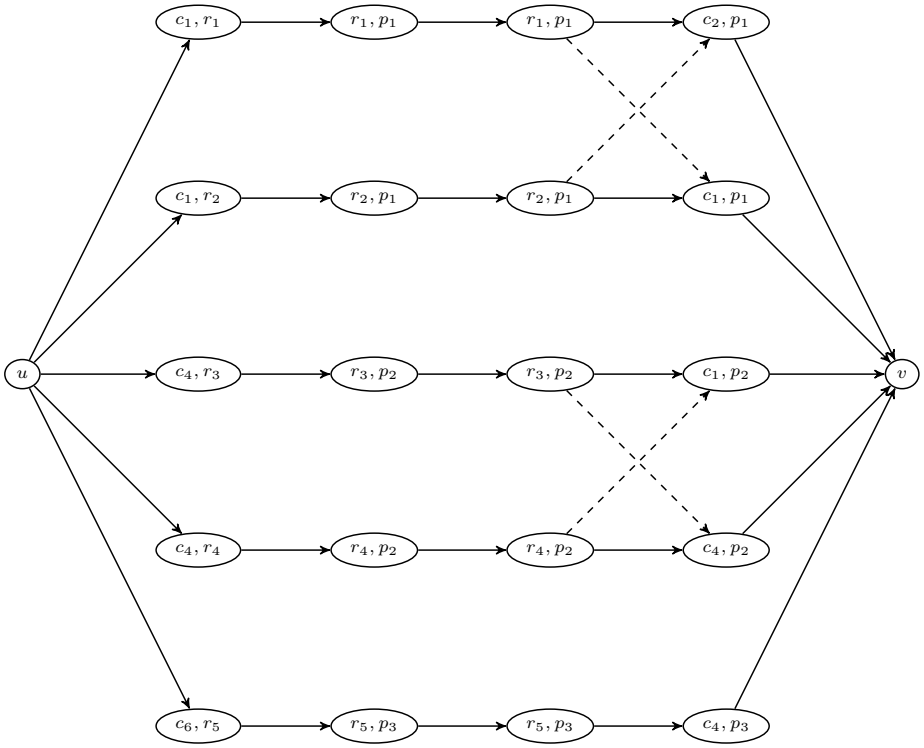


Fig. 8 Illustration of Case 2 after a couple of iterations. The interpretation of the arcs corresponds to Fig. 4.

It may be the case that $c_6 = c_1$. However this indicates that the course c_1 is teaching at least one more lecture than previously thought when we found the period p_2, so we can backtrack to the point where we found p_2 and then find another period instead of p_2 which we have not yet considered for c_1. This means that whenever we are considering a violating pair we must be able to either redirect the flow for that pair or find a new violating pair which involves a new course which we either have not yet considered before or it is a course we have been considering before and then we can backtrack to this course and consider a new period which we have not considered before for that course. Clearly all the operations can be made in polynomial asymptotic time but it needs to be shown that the total number of backtracking operations is finitely bounded to ensure that the algorithm is finite. So if the total number of backtrack operations is finite then eventually the algorithm will end up with some pair where the flow can be redirected and decrease the number of violations.

Let $T(m, n)$ be the total number of backtrack operations that our algorithm performs where $m = |P|$ and $n = |C|$. The number of times that we backtrack to the first course in our algorithm can at most be the number of lectures taught by the course since we consider

a new period not considered for the course before whenever we backtrack. Since the number of lectures is linearly bounded by m then we can at most backtrack $O(m)$ times to the first course. Every time we backtrack to the first course we have been backtracking $T(m, n-1)$ times to the remaining courses meaning that we have the following recursive relation:

$$T(m,n) = O(m) \cdot T(m,n-1)$$

Consider when $n = 2$. We can backtrack to the first course $O(m)$ times but we can never backtrack to the second course since there are no other courses to backtrack from and so we have the base case:

$$T(m,2) = O(m)$$

We will now show that the recursion leads to a finite number of total backtracking operations by making a guess of the asymptotic bound:

$$T(m,n) = O\left(m^{n-1}\right)$$

It is easy to see that it holds for the base case so we can assume that it holds for $T(m, n-1)$ and then we have:

$$\begin{aligned} T(m,n-1) &= O\left(m^{n-2}\right) \\ T(m,n) &= O(m) \cdot O\left(m^{n-2}\right) = O\left(m^{n-1}\right) \end{aligned}$$

It has now been shown by induction that the algorithm is making a finite number of backtracking operations which concludes the proof of Proposition 4.

Before proving the correctness of Algorithm 1 it should be noted that since $\sum_{r \in R} \overline{y}_{c,r} = L_c$ $\forall c \in C$ then the total capacity on the outgoing arcs of the source is $\sum_{c \in C} L_c$. This means that for the maximum flow f it must always hold that $v(f) \leq \sum_{c \in C} L_c$. Furthermore since all capacities in the graph $\mathcal{G}_{\text{mf}}$ are integers then there must be a maximum flow taking integer values (Ahuja et al, 1993, Theorem 6.5).

Proof (Proof that Algorithm 1 is correct) The proof of Proposition 4 implies that if we have a feasible integer flow $f \in \mathcal{G}_{\text{mf}}$ where $v(f) \geq \sum_{c \in C} L_c$ and if for some triple (c, p, r) we have that $f^1_{c,p,r} \neq f^2_{c,p,r}$ then there must exist some courses $c_1 \in C$, $c_2 \in C$, $c_3 \in C$, some rooms $r_1 \in R$, $r_2 \in R$ and a period $p_1 \in P$ where $c_1 \neq c_3$ (it is possible that $c_2 = c_3$) and the following holds; $f^1_{c_1,p_1,r_1} = 1$, $f^1_{c_1,p_1,r_2} = 0$, $f^1_{c_2,p_1,r_1} = 0$, $f^1_{c_3,p_1,r_2} = 1$, $f^2_{c_1,p_1,r_1} = 0$, $f^2_{c_1,p_1,r_2} = 1$, $f^2_{c_2,p_1,r_1} = 1$ and $f^2_{c_2,p_1,r_2} = 0$. Swapping the values of $f^2_{c_1,p_1,r_1}$ and $f^2_{c_1,p_1,r_2}$ and swapping the values of $f^2_{c_2,p_1,r_1}$ and $f^2_{c_2,p_1,r_2}$ will remain an integer feasible flow with the same total amount of flow and a decrease in the number of violations by at least two. So a simple algorithm is to search for these courses, these rooms and this period where the swap can be done, do the swap and then iterate. This can be implemented to run in polynomial asymptotic time instead of the exponential asymptotic time given in the proof of Proposition 4. However, since the swaps are only done on the $f^2_{c,p,r}$ variables and never on the $f^1_{c,p,r}$ variables then this means that the values of the $f^1_{c,p,r}$ variables must be feasible values for the $f^2_{c,p,r}$ and so a much simpler algorithm can be constructed by the following steps:

- Step 1 Find an integer maximum flow f. This can be done by some polynomial maximum flow algorithm, e.g. the Labeling Algorithm (Ahuja et al, 1993, proof of Theorem 6.5, section 6.5)
- Step 2 If $v(f) < \sum_{c \in C} L_c$ then return **nil**, i.e. that it is infeasible, which is correct as it has been proved that the assignment pair $(\overline{x}, \overline{y})$ cannot be feasible in this case, otherwise go to Step 3.
- Step 3 Iterate over all triples (c, p, r) and set the value of the variable $f^2_{c,p,r}$ equal to the value of the $f^1_{c,p,r}$ variable and return this new flow.

Note that these steps is exactly the description of Algorithm 1 and so the algorithm must be correct.

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA

Bettinelli A, Cacchiani V, Roberti R, Toth P (2015) An overview of curriculum-based course timetabling. TOP pp 1–37

Bron C, Kerbosch J (1973) Algorithm 457: Finding all cliques of an undirected graph. Commun ACM 16(9):575–577, DOI 10.1145/362342.362367, URL http://doi.acm.org/10.1145/362342.362367

Burke E, Marecek J, Parkes A, Rudová H (2010) Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research 37(3):582–597

Burke EK, Marec J, Parkes AJ, Rudova H (2012) A branch-and-cut procedure for the udine course timetabling problem. Annals of Operations Research 194(1):71–87

Cacchiani V, Caprara A, Roberti R, Toth P (2013) A new lower bound for curriculum-based course timetabling. Computers & Operations Research 40(10):2466 – 2477

Gaspero LD, Schaerf A, McCollum B (2007) The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. rep., School of Electronics, Electrical Engineering and Computer Science, Queens University SARC Building, Belfast, United Kingdom

Gurobi Optimization I (2015) Gurobi optimizer reference manual. URL http://www.gurobi.com

Hao JK, Benlic U (2011) Lower bounds for the itc-2007 curriculum-based course timetabling problem. European Journal of Operational Research 212(3):464 – 472

Lach G, Lübbecke M (2008) Optimal university course timetables and the partial transversal polytope. In: McGeoch C (ed) Experimental Algorithms, Lecture Notes in Computer Science, vol 5038, Springer Berlin / Heidelberg, pp 235–248

Lach G, Lübbecke M (2012) Curriculum based course timetabling: new solutions to udine benchmark instances. Annals of Operations Research 194:255–272

Scheduling, Timetabling Research Group at the University of Udine I (2015) Curriculum-based course timetabling. http://tabu.diegm.uniud.it/ctt/index.php

MISTA 2015

Appointment scheduling in hospitals

Sequencing and scheduling using timeaggregation

Sarah Kirchner · Marco Lübbecke

1 Introduction

In Germany as well as in many other countries, hospital services provided for admitted patients are settled using *diagnosis-related groups (DRGs)*. That is, patients are grouped according to their diagnosis, received services and demographic characteristics into a DRG. The hospital receives a fixed reimbursement dependent on this DRG (Institut für das Entgeltsystem im Krankenhaus , InEK). In the german DRG-system, there is a lower and an upper trim point for the length of stay of a patient with a given DRG. If the length of a patients stay is longer than the upper trim point the reimbursement is increased by a not cost-covering amount. This payment scheme provides incentives for hospitals to aim for a short hospitalization of admitted patients.

Almost all patients receive more than one medical service during their hospital stay and there may be dependencies between these services. To facilitate the requirements of these multiple dependent services, coordinated appointment calendars for the different resources of a hospital are needed. At the moment, it is common practice that medical staff at a resource sequentially assigns appointment times to incoming requests, disregarding all other services the patient may need and often without considering the impact the decision has on the length of the patients stay.

Additional to the admitted patients, many hospital units also provide ambulatory services to patients. Often they also need to consider walk-in patients that arrive at the hospital without prior notice.

The majority of contributions in literature about appointment scheduling only considers a single resource such as an operating theater (Hulshof et al, 2012). Gartner and Kolisch (2014) propose an integer programming model to schedule admitted patients. They aggregate the problem and decide for every appointment request only the day on which the appointment is scheduled. The sequencing decision on a resource is not considered.

Sarah Kirchner
RWTH Aachen University
E-mail: kirchner@or.rwth-aachen.de

Marco Lübbecke
RWTH Aachen University
E-mail: marco.luebbecke@rwth-aachen.de

Together with the software company Inform and the university hospital in Aachen, it is our goal to develop tools to support hospital staff in the decision making process, taking into account all patient groups in a hospital.

2 Problem Description

As already mentioned there are several scarce resources in a hospital. These resources can be devices like an MRT machine, rooms or persons, having regular opening hours. There are different patient groups in the hospital, which need to be treated differently in the appointment scheduling process. Admitted patients typically stay in the hospital for more than one night. Often their treatment plan is not exactly known on their arrival and services are requested during their hospital stay. These requests do not need to be answered immediately and can be batched for some time to allow for a better informed scheduling decision. Outpatients on the contrary often request one or more service via phone prior to their arrival. All of these requests should – if possible – be carried out during the same day as outpatients do not stay in the hospital overnight. Furthermore, these requests need to be answered immediately and cannot be batched. Additionally walk-in patients request services that need to be scheduled on the same day. These requests also need to be answered immediately. It may be possible to process a request on more than one resource. Therefore a date, time and resource need to be determined for every request. The objective is to minimize the average length of stay of all patients.

As a hospital is a highly dynamic environment in which new patients arrive and new appointments are made over time, the appointment scheduling problem has to be solved frequently and decisions made in the past have to be considered in future scheduling iterations.

The different requirements for response times to requests call for different solution algorithms.

3 Solution Methods

We propose to batch requests for admitted patients if possible and compute a solution for these requests during the night, when most resources are closed and only few appointment requests are made. We assume that accurate information about the exact execution time for a request is only needed on the day of execution itself. Before that, it is sufficient to know the day (or week for requests in the far future) the request is scheduled to be processed. A time-indexed IP formulation for the problem is proposed in which timeslots are aggregated for the time after the next day. The aggregated decisions are considered in the next solving iteration for consistency. We consider the sequencing decisions on resources for the next working day and make aggregated decisions only for the time after that.

For the requests of ambulant and walk-in patients we propose a greedy heuristic since a fast response is needed and therefore IP-based approaches do not seem to fit. Additionally a wait-and-see heuristic is used to fill idle time on the resources during a day.

References

Gartner D, Kolisch R (2014) Scheduling the hospital-wide flow of elective patients. European Journal of Operational Research 233(3):689 – 699

Hulshof PJ, Kortbeek N, Boucherie RJ, Hans EW, Bakker PJ (2012) Taxonomic classification of planning decisions in health care: a structured review of the state of the art in or/ms. Health systems 1(2):129–175

Institut für das Entgeltsystem im Krankenhaus (InEK) (2014) G-DRG German Diagnosis Related Groups, Version 2015, Definitionshandbuch. http://www.g-drg.de/cms/G-DRG-System_2015/Definitionshandbuch

MISTA 2015

Resource Failure Recovery in Production Scheduling

Roman Barták · Marek Vlk

1 Introduction

In real-life scheduling, manufacturing systems face uncertainty due to unexpected events occurring on the shop floor. Machines break down, operations take longer than anticipated, personnel do not perform as expected, urgent orders arrive, others are cancelled, etc. These disturbances render the ongoing schedule infeasible. In such cases, a simple approach is to collect the data from the shop floor when the disruption occurs and to generate a new schedule from scratch. Gathering the information and complete rescheduling involve excessive amount of time which may lead to a failure of the scheduling mechanism. Moreover, the recovered schedule may deviate prohibitively from the ongoing schedule.

This paper describes two algorithms for updating a schedule in response to a resource failure. The first method takes the activities that were to be processed on a broken down machine, reallocates them, and then it keeps repairing violated constraints until it gets a feasible schedule. The second method deallocates a subset of activities and then it allocates them back through Conflict-directed Backjumping with Backmarking.

2 Related Works

The field of rescheduling (predictive-reactive scheduling) has been addressed in a number of works, as surveyed for instance in [11], [14], and [9]. Most of the related works suggest generally usable approaches regardless of the particular scheduling model. Nevertheless, it is not always possible to straightforwardly use the presented methods for a selected class of scheduling problems, as it requires significant adjustments to make it applicable. In addition, if it is desired to achieve better results in terms of the speed of procedures and the modification distance of a schedule, it is suitable to tailor an algorithm for the particular class of problems.

Roman Barták, Marek Vlk
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
E-mail: {bartak, vlk}@ktiml.mff.cuni.cz

The fundamental inspiration for our methods comes from *heuristic-based* approaches, which do not guarantee to find an optimal solution, but respond in a short time. The simplest schedule repair technique is the *right shift rescheduling* [1]. This technique shifts the operations globally to the right on the time axis in order to cope with disruptions. When it arises from machine breakdown, the method introduces gaps in the schedule, during which the machines are idle. It is obvious that this approach results in schedules of bad quality, and can be used only for environments involving minor disruptions.

The shortcomings of total rescheduling and right shift rescheduling gave rise to another approach: *affected operation rescheduling*, also referred to as partial schedule repair [13]. The idea of this algorithm is to reschedule only the operations directly and indirectly affected by the disruption in order to minimize the deviation from the initial schedule.

The *Repair-DTP* algorithm proposed in [12] tackles a problem very similar to ours, however, it is designed to correct violated constraints in manually edited schedules. The model involves precedence constraints and synchronization constraints, but excludes minimum and maximum time lags. Nonetheless, in order to reduce the search space, the Repair-DTP algorithm employs *Simple Temporal Networks* (STN) [7] and *Incremental Full Path Consistency* (IFPC) algorithm [10], which incrementally maintains the *All Pairs Shortest Path* (APSP) property. If a feasible correction exists, the algorithm tries to find the most similar schedule to the initial one through only shifting activities in time. Since the Repair-DTP algorithm does not try changes in resource selection, it cannot be used to deal with machine failure. Moreover, the main shortcoming of the algorithm is searching through disjunctions, introduced by hierarchical nature of the model and by resource unarity. This leads to excessive (exponentially growing) amount of temporal networks that are inspected, which requires unacceptable amount of time.

In the methods proposed further, apart from STN and IFPC algorithm, some widely used search techniques from the field of *Constraint Satisfaction* [6] are employed, namely *Conflict-directed Backjumping with Backmarking* [8].

3 Problem Definition

The scheduling model is taken from the FlowOpt system [4], which contains a tool for designing and editing *manufacturing workflows*. Workflows in the FlowOpt model match up the structure of Nested Temporal Networks with Alternatives [3].

A scheduling problem consists of activities that are required to be processed. The time distance between two distinct activities may be limited by a *simple temporal constraint* [7]. Each such constraint can be written as a triplet (A_i, A_j, w_{ij}), and the semantics is such that $Start(A_j) - Start(A_i) \leq w_{ij}$, where $Start(A_i), Start(A_j)$ denote the start times of activities, and $w_{ij} \in \mathbb{Z}$. Note that the minimal time distance between the two activities is obtained through adding a reverse constraint (A_j, A_i, w_{ji}), which means $Start(A_i) - Start(A_j) \leq w_{ji}$, and hence $Start(A_j) - Start(A_i) \geq -w_{ji}$. Therefore, in what follows we understand temporal constraints as constraints determining the maximal distance between start times of two distinct activities.

Activities are processed on *resources*. All resources are unary, which means that each resource may perform no more than one activity at a time, i.e., activities on a resource may not overlap. This limitation is referred to as a *resource constraint*. To

make a schedule feasible, each activity requires exactly one resource from its associated *resource group* to be *selected*.

The problem we actually tackle is that we are given a particular instance of the scheduling problem along with a feasible schedule, and also with a resource that can no longer be used. The aim is to find as fast as possible a feasible schedule that is as similar to the ongoing schedule as possible. In order to simplify the description of the algorithms, let us assume that a resource fails at the beginning of the time horizon, i.e., right before the schedule execution begins.

The methods proposed in related literature cannot be straightforwardly used for the model we deal with because of the presence of simple temporal constraints that are more restrictive than frequently used precedence constraints.

4 Right Shift Affected

Right Shift Affected is a greedy algorithm aimed at changing allocation of as few activities as possible. The idea is to reallocate activities from the failed resource and then keep reallocating activities that violate some constraint until the schedule is feasible.

The algorithm works as follows. First, it goes through all activities in the model and checks whether the activity is allocated to the failed resource. If so, the activity is reallocated (seeking for an available resource after the original start time of the activity), and the activity is added to the set $\mathcal{A}$. Now, none of the activities uses the failed resource and the set $\mathcal{A}$ contains activities that have been reallocated and therefore must be checked for temporal constraint violation.

Next, the algorithm takes an activity from the set $\mathcal{A}$ and proceeds to repair all violated temporal constraints associated with the activity in question. It repairs the constraints through moving activities to the right, so that if another activity is moved, it is added into the set $\mathcal{A}$ because it must be then checked for temporal constraint violation. If the activity does no longer violate any temporal constraint, the algorithm proceeds to another one from $\mathcal{A}$.

Activities are reallocated as follows. Suppose the algorithm wants to repair a temporal constraint in such a way that an activity A should be reallocated to a time point t. Then activity A is allocated in such a way that it does not violate any resource constraint, which is achieved through seeking a time point t^* (greater than or equal to time point t) where activity A can be allocated without violating the resource constraints. Since the activities are always allocated in such a way that no resource constraints are violated, the routine checks only temporal constraints.

Violated temporal constraints are repaired as follows. When a temporal constraint $Start(A_j) - Start(A_i) \leq w_{ij}$ is violated, then the algorithm moves the activity A_i to the right starting from the minimal time point satisfying the constraint ($= Start(A_j) - w_{ij}$). When the algorithm picks an activity to be repaired, then it iterates over all temporal constraints associated with the activity being repaired until the activity satisfies all associated constraints.

As far as the order of taking activities from $\mathcal{A}$ is concerned, the best heuristic with respect to all conceivable performance measures turned out to be picking the rightmost activity, i.e., the activity with the maximum start time. The explanation is that shifting the rightmost activities rightwards makes free space for shifting the activities allocated more on the left, which would otherwise have to creep over one another.

5 STN-recovery

STN-recovery is a bit more sophisticated algorithm to tackle the resource failure. This algorithm anticipates that moving a large number of activities by a short time is preferable to moving activities a lot in time. The basic idea is to deallocate some set of already scheduled activities and then to allocate them back again.

The point of the algorithm is to allocate connected components one after another using Conflict-directed Backjumping with Backmarking (CBJBM) [8]. The allocation of an activity is carried out such that the start time of the activity is continuously incremented until an available resource at that time is found, or until the maximal possible value of the start time (which is determined with respect to the already allocated activities) is exceeded. In the former case the algorithm proceeds to allocate the next activity, in the latter case the algorithm goes back to reallocate some previous activity.

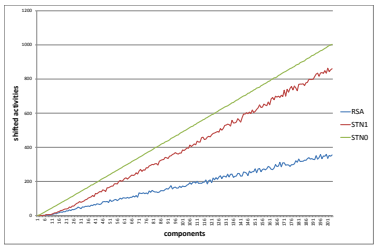
The Simple Temporal Network (STN) [7] is built from the temporal constraints in the model and the STN with the All Pairs Shortest Path property (APSP) is computed first — before the schedule execution begins. Recall that the APSP property gives consistent maximal distances between the start times of all pairs of activities.

STN-recovery itself consists of the following six steps.

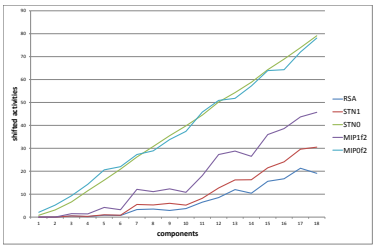
1. Find activities allocated to the failed resource and change their resource selection to an available resource, picking the resource with the lowest usage, while keeping the start times of the activities unchanged. Now some activities allocated on the same resource may overlap.
2. For each resource (to which some activity has been added in step 1) shift the activities that overlap (to the right) so as they do not overlap, and add them into the set $\mathcal{A}$. Include in $\mathcal{A}$ also activities that were not actually shifted but are allocated on the right of those shifted activities on the same resource.
3. For the sake of pruning the search space of the forthcoming allocation search, add STN constraints between the global predecessor and each activity in $\mathcal{A}$ so as to enforce that they can only start at the time they are currently allocated or later. Update the STN via Incremental Full Path Consistency [10] to preserve the APSP property.
4. For each activity A in $\mathcal{A}$, acquire the connected component the activity A belongs to, and for all activities in all acquired connected components compute their $MinStart$ values that is the maximum of (i) the current start time of the activity and (ii) its minimal distance from the global predecessor resulting from the STN.
5. Deallocate (retract from resources) all activities in all connected components acquired in step 4.
6. Take the leftmost (according to the $MinStart$ values) non-allocated component C and allocate all activities in C starting with its leftmost activity through CBJBM. The activities within a connected component are allocated in the increasing order of their $MinStart$ values. Repeat this step until all connected components are allocated.

6 Experimental Results

We performed experiments with randomly generated problems composed of 6 resources in each of two resource groups. Each connected component consists of 5 activities and



(a) Comparison for Right Shift Affected and STN-recovery.



(b) Comparison including MIP models.

Fig. 1: Modification distance: the number of shifted activities.

up to 10 temporal constraints (some may be redundant). The values of x-axes in the following figures are the number of connected components in the model. Having more resources in a group than the number of activities in a component ensures recoverability from a resource failure.

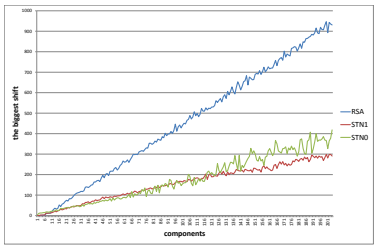
To justify the claims from the introduction, the comparison also includes what we refer to as $STN0$ that is the sixth (last) step of STN-recovery itself and thus corresponds to the rescheduling from scratch. The entire STN-recovery algorithm as described is referred to as $STN1$.

Further, we modelled the rescheduling problem as a Mixed Integer Program (MIP). Having n activities and m resources, the model involves variables shr_i and shl_i for shifts of activities to the right and to the left respectively, and binary variables r_{ij} indicating that activity i is scheduled to resource j. This is referred to as $MIP0$. Further, recall that the first five steps of the STN-recovery algorithm obtain and unschedule a set of activities that are to be allocated in the sixth step, while the other activities remain untouched. Suppose that each activity that is to remain untouched is fixed. This model is referred to as $MIP1$.

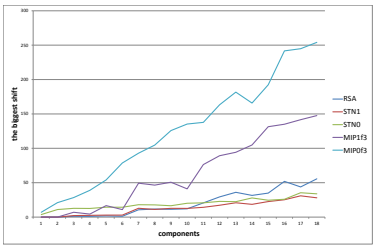
The experiments were conducted using the mosek optimizer [2] with the following settings. If an optimal solution is not found in 10 seconds, the engine outputs the first integer feasible solution found. The algorithms were running on Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz, 3701 Mhz, kernels: 4, logical processors: 8; RAM: 8,00 GB. Since the MIP models terminated only for problems containing a small number of activities, we include one figure with and one figure without the MIP models.

The results confirm the hypotheses that the Right Shift Affected algorithm is far better when minimizing the number of shifted activities (Figure 1), whereas the STN-recovery algorithm is significantly better in minimizing the biggest shift of an activity (Figure 2). As far as the total sum of shifts is concerned, STN-recovery outperforms the Right Shift Affected, but the difference is not that big.

Right Shift Affected is somewhat faster than STN-recovery (Figure 3), however, STN-recovery has the following advantage. The algorithm always allocates the leftmost connected component that has not been allocated yet, therefore, when the algorithm is allocating the connected component with the leftmost activity that has the $MinStart$ value t, the schedule is not going to be modified before the time point t. This allows

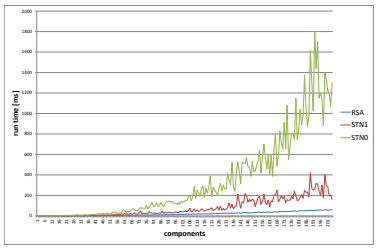


(a) Comparison for Right Shift Affected and STN-recovery.

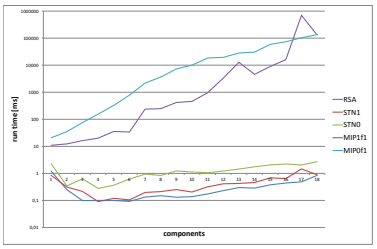


(b) Comparison including MIP models.

Fig. 2: Modification distance: the biggest shift of an activity.



(a) Comparison for Right Shift Affected and STN-recovery.



(b) Comparison including MIP models (logarithmic scale).

Fig. 3: Run times for the algorithms in milliseconds.

the system to keep executing the ongoing schedule even if it has not been completely recovered yet.

7 Conclusion and Future Goals

This paper described two different methods to handle a resource failure, i.e., a disruption when a resource suddenly cannot be used anymore by any activity. The first method is suitable when it is desired to move as few activities as possible; however, the question whether the algorithm always ends is still open. The second method is useful when the intention is to shift activities by a short time distance, regardless of the number of moved activities. The main shortcoming is that if there is no feasible recovery of the ongoing schedule, neither of the methods is able to quickly and securely report it. In real-life environments, however, the schedule recoverability from the breakdown of any particular machine is often known (for instance the minimum required number of available resources of each resource group may be obvious) or can be computed before the schedule execution begins.

This work also compared the suggested algorithms to the Mixed Integer Programming models. Solving the models using the mosek optimizer turned out to be uncom-

petitive with the two suggested algorithms. A more detailed description of the methods and the MIP models may be found in [5].

Our further task is to propose a new technique, which, in response to unforeseen events, will not only modify the allocation of already scheduled activities, but will be able to replace some activities in the original schedule by a set of other (not scheduled) activities by searching through alternative branches, i.e., to replan some (ideally the smallest necessary) subset of the schedule.

Acknowledgements This research is partially supported by SVV project number 260 224 and by the Czech Science Foundation under the project P103-15-19877S.

References

1. Abumaizar, R.J., Svestka, J.A.: Rescheduling job shops under random disruptions. International Journal of Production Research **35**(7), 2065–2082 (1997)
2. Andersen, E.D., Andersen, K.D.: The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In: High performance optimization, pp. 197–232. Springer (2000)
3. Barták, R., Čepek, O.: Nested temporal networks with alternatives. In: AAAI Workshop on Spatial and Temporal Reasoning, Technical Report WS-07-12, AAAI Press, pp. 1–8 (2007)
4. Barták, R., Jaška, M., Novák, L., Rovenský, V., Skalický, T., Cully, M., Sheahan, C., Thanh-Tung, D.: Flowopt: Bridging the gap between optimization technology and manufacturing planners. In: Luc De Raedt et al. (Eds.): Proceedings of 20th European Conference on Artificial Intelligence (ECAI 2012), pp. 1003–1004. IOS Press (2012)
5. Barták, R., Vlk, M.: Machine breakdown recovery in production scheduling with simple temporal constraints. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science) (2015). To appear
6. Brailsford, S.C., Potts, C.N., Smith, B.M.: Constraint satisfaction problems: Algorithms and applications. European Journal of Operational Research **119**(3), 557–581 (1999)
7. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial intelligence **49**(1), 61–95 (1991)
8. Kondrak, G., Van Beek, P.: A theoretical evaluation of selected backtracking algorithms. Artificial Intelligence **89**(1), 365–387 (1997)
9. Ouelhadj, D., Petrovic, S.: A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling **12**(4), 417–431 (2009)
10. Planken, L.R.: New algorithms for the simple temporal problem. Ph.D. thesis, TU Delft, Delft University of Technology (2008)
11. Raheja, A.S., Subramaniam, V.: Reactive recovery of job shop schedules – a review. International Journal of Advanced Manufacturing Technology **19**, 756–763 (2002)
12. Skalický, T.: Interactive scheduling and visualisation. Master's thesis, Charles University in Prague (2011)
13. Smith, S.F.: Reactive scheduling systems. In: D. Brown and W. Scherer (eds.), Intelligent scheduling systems, pp. 155–192. Springer US (1995)
14. Vieira, G., Herrmann, J., Lin, E.: Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of Scheduling **6**, 39–62 (2003)

MISTA 2015

Primal Heuristics for the Vehicle Routing Problem with Synchronized Visits

Sohaib Afifi · Aziz Moukrim

Abstract The aim of the paper is to solve exactly a new variant of the vehicle routing problem with time windows which includes synchronization visits. A new formulation is proposed and solved using boosting methods. We propose some dedicated primal heuristics and node feasibility checks. The results compared to a previous formulation and to a standard solver show the efficiency of this type of combination.

1 Introduction

This paper provides a new solution method for a particular variant of the vehicle routing problem (VRP). The problem is called VRP with synchronized visits (VRPTWSyn). In addition to the time windows constraints, we consider that for some customers, we need more than one visit, e.g., two visits from two different vehicles are required to complete the service. Visits associated with a particular customer need to be synchronized, i.e. having the same start time.

VRPTWSyn was first studied by Bredström and Rönnqvist [5] considering an application in home-care services for elders. The authors proposed a mathematical formulation and studied the role of the synchronization constraints. As a continuity, the same authors proposed in [4] a branch-and-price algorithm. A meta-heuristic approach based on a simulated annealing schema with some dedicated local searches has been proposed by Afifi et al. [1]. Later, Labadie et al. [7], in a preliminary work, proposed an iterative local search algorithm and presented some results on the small and medium sized instances. Rousseau et al. [9] considered a dynamic case of the problem and proposed a heuristic method with a constraint programming component. General perspectives of temporal constraints for vehicle routing can also be found in the survey [6] and for the home health care service in [8].

Sohaib Afifi · Aziz Moukrim
Sorbonne universités, Université de Technologie de Compiègne, CNRS, laboratoire Heudiasyc UMR 7253, CS 60 319, 57 avenue de Landshut 60 203 Compiègne cedex
E-mail: {sohaib.afifi, aziz.moukrim}@hds.utc.fr

2 The method

The formulation presented in Bredström and Rönnqvist [5] uses $\mathcal{O}(mn^2)$ binary variables and $\mathcal{O}(mn^2)$ constraints, where n is the number of customers and m the number of vehicles. In this paper, we propose and use a new linear formulation for VRPTWSyn adapted from the one presented in Bard et al. [3]. It uses $\mathcal{O}(n^2)$ binary variables and $\mathcal{O}(n^2)$ constraints.

Beside the use of the light formulation, our aim in this work is to boost its execution by keeping the Branch-and-bound tree small and by discovering feasible solutions earlier in the process. Primal heuristics are a very important aspect for MIPs. We developed and tested a number of heuristics dedicated to our problem and discussed their integration into the branch-and-bound process. We count three types of primal heuristics: start heuristics, diving heuristics and improvement heuristics. Most of them explore the components developed and presented in [1].

Start heuristics aim at finding a feasible solution before the beginning of the tree exploration, usually at the presolving stage or at the root node. It uses the construction operator described in [1] and is run only at the beginning of the process.

Diving heuristics aim at finding a feasible solution early in the Branch-and-Bound-process without the need to branch on all the variables following the quickly go-down strategy. One can simply change the feasibility graph according to the fixed variables at the current node and then run the construction algorithm on the new reduced problem defined by this graph.

Improvement heuristics are algorithms that aim to form a new feasible solution of better objective value out of the current incumbent. This counts four neighborhoods which are randomly executed in some nodes, TwoOpt*, OrOpt, Move and Exchange [1].

Another strategy is to keep the Branch-and-bound tree small by checking the feasibility of the current node using dedicated checkers which explore the problem specifications. In this path, we used the energetic reasoning and the clique methods proposed in [2] for the VRPTW. At every node a new problem is created considering the local constraints defined at the current node. Then either energetic reasoning or maximum clique are checked so the number of tours needed to serve all the visits respecting the current constraints should not exceed the maximum number of vehicles.

3 Experimentation

The experimentations are done on the standard instances of Bredström and Rönnqvist [5]. The benchmark, which was generated to simulate the scheduling problem in home-care services, comprises 10 data sets. Each contains three types of instances based on the size of the time windows. We implemented the branch-and-cut algorithm in C++ using the framework SCIP with IBM Ilog Cplex 12.6 as underlying LP solver. The program is compiled with GNU GCC in a Linux environment.

The heuristics presented here are able to generate feasible solutions quickly. They reduce the calculation time while keep improving the quality of the solutions. In the other hand, the feasibility check used can predict infeasibility in earlier stages and hence reducing the time used to improve the dual bound. Figure 1 shows an example on an average size instance and how introducing these techniques reduced considerably the cpu time on both sides, primal and dual solutions.

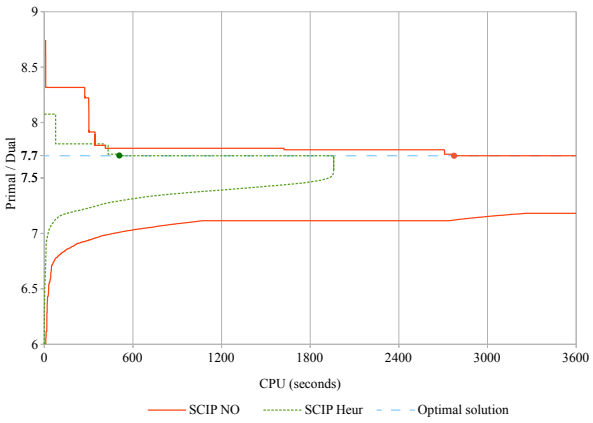


Fig. 1 Development of the primal and dual bound, if MIP processes instance 6M, with default settings (continued line) and with our heuristics (dashed).

Acknowledgements This work was partially supported by the National Agency for Research, under TCDU project, reference ANR-14-CE22-0017 and was carried out in the framework of the Labex MS2T, which was funded by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

1. Afifi S, Dang DC, Moukrim A (2013) A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In: Proc. of LION-7, Lecture Notes in Computer Science, vol 7997, pp 259–265
2. Afifi S, Guibadj RN, Moukrim A (2014) New lower bounds on the number of vehicles for the vehicle routing problem with time windows. In: CPAIOR 2014, Cork, Ireland, Springer, Lecture Notes in Computer Science, vol 8451, pp 422–437
3. Bard J, Kontoravdis G, Yu G (2002) A branch-and-cut procedure for the vehicle routing problem with time windows. Transportation Science 36(2):250–269
4. Bredström D, Rönnqvist M (2007) A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. NHH Dept of Finance and Management Science Discussion Paper
5. Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research 191(1):19–31
6. Drexl M (2012) Synchronization in vehicle routing-a survey of vrps with multiple synchronization constraints. Transportation Science 46(3):297–316
7. Labadie N, Prins C, Yang Y (2014) Iterated local search for a vehicle routing problem with synchronization constraints pp 257–263
8. Mankowska DS, Meisel F, Bierwirth C (2014) The home health care routing and scheduling problem with interdependent services. Health care management science 17(1):15–30
9. Rousseau LM, Gendreau M, Pesant G (2013) The synchronized dynamic vehicle dispatching problem. INFOR: Information Systems and Operational Research 51(2):76–83

MISTA 2015

Comparison of EATTS and XHSTT - Towards a Unified Description Language for Timetabling Problems

M. Müller · J. Ostler · P. Wilke

Abstract Researchers and users of timetabling algorithms would like to benchmark the different approaches and implementations. One approach is the usage of the same dataset or specification of the problem presuming that such a uniform description exists.

Here we compare two specification languages, EATTS and XHSTT, capable to describe all aspects of a timetabling problem, including input data, constraints and solutions. In addition we will show that the set of problems which can be specified by EATTS includes those which can be specified by XHSTT but not vice versa.

It would be quite useful if a single standardized description language would be established so we request comments on which directions further developments should go.

1 Introduction

Timetabling problems belong to the most popular classes of optimization problems and therefore, different approaches have been published which consist of descriptions of the problem and possible solution strategies. It is obvious that in order to compare such solution strategies and/or particular solutions, it is essential to use the same database. Since diverse approaches use different and probably incompatible description languages, the use of either transformations between the languages or a unified description mechanism becomes necessary.

In this work, two already existing ways of specification will be discussed and compared to each other. Furthermore, a transformation between both specification formats will be proposed with emphasis on the most significant issues encountered during this transformation's modelling.

Multi Criteria Optimisation Group
Pattern Recognition Lab
Computer Science Dept.
University Erlangen-Nuernberg, Martensstrasse 3, 91058 Erlangen, Germany
E-mail: Peter.Wilke@FAU.DE

2 Related Work

Timetabling problems originate from different (real world) situations, which result in diverse types of problem (e.g. nurse rostering, school time tabling, staff absence planning). Therefore, description languages for such frameworks develop individually and with emphasis on specific aspects of the problems. A specification language can either be designed to describe one specific type of timetabling problem, for which it is optimized. Or it can be designed to be as general as possible, and can thus be used to describe timetabling problems of almost any kind. XHSTT [Post et al(2010)Post, Kingston et al, Post et al(2014)Post, Kingston et al] is an example for the former case, specialized in high school timetabling. The timetabling framework EATTS [Ostler and Wilke(2010)], in contrast, represents the general approach.

3 Comparison of the Approaches

The main difference between EATTS and XHSTT is the specification of planning events and their handling during the making and change of assignments in the optimization phase.

3.1 XHSTT –Approach of the University of Twente

XHSTT distinguishes between two different types of planning events: those that are defined in the specification as events that are to be planned (called instance events) and those that are part of the solution (called solution events). Each instance event consists of a duration and of various slots (event resources) for resources that are required for this event; these event resources can be fixed references, i.e. pointers to existing resources, or formal descriptions of required resources which are of a given type and are to fulfil a given role within the event

In order to satisfy the instance event, a solver needs to generate one or more solution events. Each solution event, which corresponds to a given instance event, should be assigned resources according to the resource lists given in this enclosing instance event. In other words, the XHSTT approach requires the solver to create new events during the optimization phase.

Another property of this framework worth being mentioned is the discrete character of the defined time slots. By default, each time slot has the canonical duration of 1 *time unit*. In case that longer planning periods are required they are specified by the slot's membership in a corresponding resource group (called time group) which contains all the time slot resources of this period.

3.2 EATTS – Approach of the University of Erlangen

In the EATTS framework, the same format is used for problem specification (input) and solution specification (output) of problems. Whilst he specification of required resources is analogous to the one of XHSTT, events and times are handled differently. Each event which is subject to assignments in a solution or during the optimisation phase has to be defined in the specification, and no further events can be created after

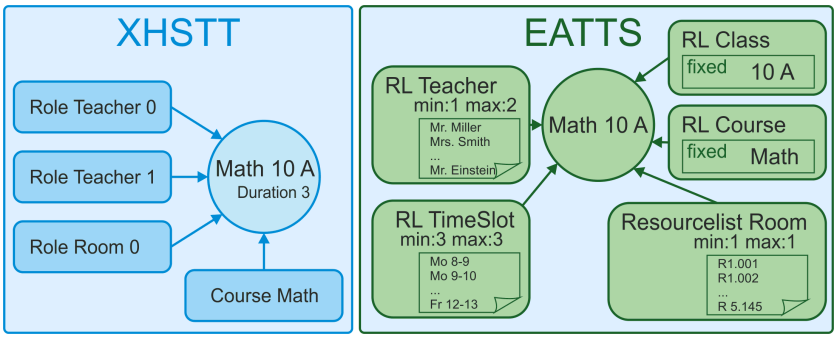


Fig. 1 Event Declaration XHSTT versus EATTS

the start of the optimisation. Each splitting of an event is implemented by assigning a set of (not necessarily consecutive) time slot resources to this event instead of creating new solution events with their own assigned times and resources.

In contrast to XHSTT, time slots in EATTS are defined by a discrete start time and duration.

Fig. 1 shows the two different ways how a high school timetabling event is specified. In both cases 2 different teachers are able to give Math for class 10A with length 3 time slots. In addition EATTS allows to specify a list of available and suitable resources (like rooms), while in XHSTT suitable resources are identified by constraints.

3.3 Comparison

The differences in both approaches cause problems to a possible computational transformation of specification files between them. They also have a non-negligible impact on the way a solver computes solutions as well as on the solution space itself. The most apparent advantage of the XHSTT format over EATTS is a direct consequence of the fact that EATTS does not allow its events to have different assignments for different time slots, but rather one single assignment for the whole event. This also implies that the assigned resources of any event with duration greater than one will be busy during all the time slots assigned to that event. In contrast, XHSTT's instance events produce solution events, with each having its own resources assigned to the resource slots specified in the instance event. This results in an increased flexibility for the overall possible assignments of resources to all existing events, and in an extended solution space for XHSTT.

A further observation made during the design of the transformation is that it can make a significant difference for the solver and the solution space, whether specific solution properties are guaranteed by the optimisation framework or controlled by constraints. Assume, for example, a teacher who does not wish to work on Wednesdays - for whatever reason. An optimisation framework can then either use an (explicit) constraint which returns a non-zero cost in case this teacher is assigned to events taking place on Wednesday, or not make such assignments at all. In the latter case, the (implicit) constraint will always be satisfied, but no solutions can be created which violate it but maybe show better overall fitness values or lead the optimisation strategy to better solutions.

4 Transformation of Specifications - Semantic hierarchy

Now we focus on the semantic power of both approaches. We will proof that:

$$L(XHSTT) \subset L(EATTS)$$

E.g. all descriptions given in the language $L(XHSTT)$ generated by XHSTT can also be expressed in EATTS. As XHSTT is tailored to specify high school timetabling problems the set of describable problems is smaller than that of EATTS, i.e.:

$$L(XHSTT) \neq L(EATTS)$$

In order to obtain a viable computational transformation between the two formats, a parser was designed and implemented, which is able to read specification file of the XHSTT format and generate files of the EATTS format, such that the semantic is preserved. This parser processes XML files and generates an internal attributed syntax tree (the data structure used by EATTS). The parser was implemented using the recursive descendent approach. Before the start of the parsing, a semantic analyser checks the XML file for possible inconsistencies.

The transformation consists of the following tasks:

1. reading in the time slots, resources and resource types,
2. converting the discrete atomic time slots to a continuous representation,
3. grouping time slots and resources,
4. reading in and preprocessing the instance events (see below),
5. grouping the events,
6. reading in the constraints.

Due to the different concepts to represent events internally, some additional preprocessing steps are necessary. In detail for each instance event in the XHSTT specification:

1. read in the instance event's duration d and required resources,
2. calculate the number n of atomic time units to represent d,
3. generate n event data structures in the EATTS framework while preserving the information about their origin.

These n events represent potential solution events in the sense of XHSTT. While the necessary data structures for the administration of events were already available in EATTS, additional procedures had to be implemented in order to provide the solution event character. For example, consider an instance event of duration 2 *time units*, requiring 1 teacher as its resource. The parser then generates 2 EATTS-events and stores them in EATTS. As soon as the solver wants to assign a time slot to the instance event, the control mechanisms picks one of the 2 generated events, assigns the time and releases the event for assignment of teacher resources.

5 Current State of the Work, possible Enhancements, and future Work

As of today, the parser successfully transforms timetabling problems from the XHSTT format to both EATTS internal representations and specification files. It is not restricted to specific problem instances, but is able to parse any problem file complying the XML documentation given by the Univ. Twente [Post(2015)].

Work in progress is the use of the above mentioned control mechanisms, which assert the event handling according to XHSTT. Tests executed so far indicate that the handling of events works as expected and accordingly to XHSTT.

An interesting topic for future work is the difference in the handling of specific constraints, as explained in sec. 3.3. It could be worth investigating the impact of implicit or explicit execution of constraints regarding runtime behaviour and solution quality.

Furthermore, it would be desirable to compare solutions of both approaches with each other. Therefore it is necessary to transform solutions to a unified format.

Acknowledgements Special thanks goes to the research group at the Centre for Telematics and Information Technology at the University of Twente.

References

Ostler and Wilke(2010). Ostler J, Wilke P (2010) The Erlangen Advanced Timetabling System (EATTS) Unified XML File Format for the Specification of Timetabling Systems. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Patat 2010 - Queen's University Belfast, pp 447–464, URL `http://www.cs.qub.ac.uk/~b.mccollum/patat10/Proceedings_patat10.pdf`

Post(2015). Post G (2015) Benchmarking project for (High) School Timetabling. URL `http://www.utwente.nl/ctit/hstt/`

Post et al(2010)Post, Kingston et al. Post G, Kingston JH, et al (2010) An XML Format for Benchmarks in High School Timetabling II. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Patat 2010 - Queen's University Belfast, pp 347–352, URL `http://www.cs.qub.ac.uk/~b.mccollum/patat10/Proceedings_patat10.pdf`

Post et al(2014)Post, Kingston et al. Post G, Kingston JH, et al (2014) XHSTT: an XML Archive for High School Timetabling Problems in Different Countries. In: Annals of Operations Research, pp 295–301

MISTA 2015

Mixed-criticality scheduling with known probabilities

Yasmina Seddik · Zdenek Hanzalek

1 Introduction

We consider n jobs $J_1, \ldots, J_n$ to be processed on a single machine. For $i = 1, \ldots, n$, job J_i has a release date $r_i \geq 0$ and a *criticality level* $L_i \in \mathbb{N}^*$: the greater is L_i, the more critical is J_i. Job J_i has L_i possible processing times $0 < p_{i,1} < \ldots < p_{i,L_i}$, as well as a deadline $\tilde{d}_i \geq r_i + p_{i,L_i}$. Moreover, each job J_i has a weight $w_i > 0$. All the parameters are integer. The actual processing time of job J_i is uncertain (it takes one of the values $p_{i,1}, \ldots, p_{i,L_i}$) and is only known at runtime: we denote it by p_i. If $p_i = p_{i,j}$ we say that j is the *execution level* of J_i, $j \in \{1, \ldots, L_i\}$.

Notice that the criticality level of a job is known in advance, while its execution level is only known at runtime. The execution level of a job is smaller than or equal to its criticality level.

In the following, we represent mixed-criticality jobs by "F-shapes" as in Figure 1.

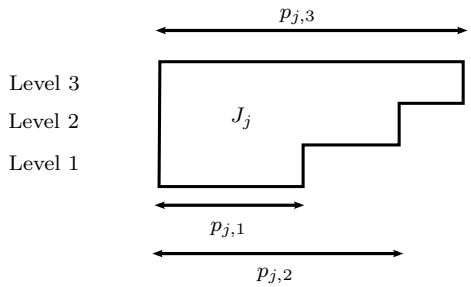


Fig. 1 Representation of a mixed-criticality job with criticality level 3

Yasmina Seddik
Czech Technical University, Prague, Czech Republic
E-mail: yasmina.seddik@fel.cvut.cz

Zdenek Hanzalek
Czech Technical University, Prague, Czech Republic
E-mail: hanzalek@fel.cvut.cz

A *schedule* S is defined as a vector $(s_1, \ldots, s_n)$ where s_i is the starting time of job J_i in S. We define a *feasible schedule* $S = (s_1, \ldots, s_n)$ as a schedule that satisfies the following conditions (cf. Figure 2):

1. Release dates and deadlines constraints are fulfilled: $\forall i \in \{1, \ldots, n\}$: $s_i \geq r_i$ and $s_i + p_{i,L_i} \leq \tilde{d}_i$;
2. At each level of criticality, jobs do not overlap. Equivalently, jobs do not overlap at their highest common level: $\forall i, j \in \{1, \ldots, n\}$ s.t. $s_i < s_j$, we have: $s_i + p_{i,k} \leq s_j$, with $k = \min(L_i, L_j)$.

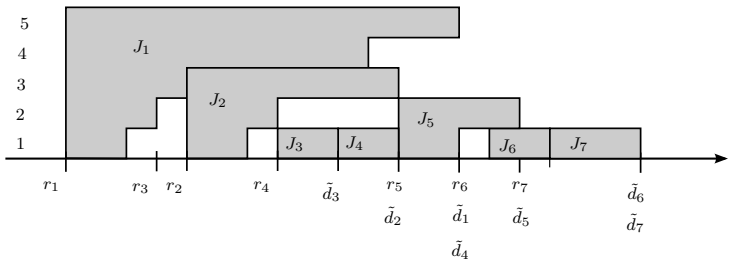


Fig. 2 A feasible schedule

At runtime, jobs with lower criticality levels can be rejected (i.e. not executed) to allow the execution of more critical jobs. For instance, on the example of Figure 2, if job J_5 is executed until time $\tilde{d}_5$, then J_6 is not executed. At runtime, each job J_i starts its execution at its starting time s_i if and only if the machine is idle at time s_i. If a job J_i is started, it is completed, whatever is its processing time at runtime.

We assume that probabilities of execution levels are known for every job, i.e. the probability of job J_i to be executed for $p_{i,j}$ time units, given that job J_i is not rejected, $i = 1, \ldots, n$, $j = 1, \ldots, L_i$. Given a feasible schedule $S = (s_1, \ldots, s_n)$, we denote by P_i the probability that job J_i is processed (not rejected). The objective function (defined in the next paragraph) of the considered problem is a function of values $P_1, \ldots, P_n$. The value of P_i, $i = 1, \ldots, n$, depends on the jobs covering J_i in S, where the "coverage" is defined as follows. Given a feasible schedule $S = (s_1, \ldots, s_n)$ and two jobs J_i, J_j in S, we say that job J_i covers job J_j iff $s_i < s_j < s_i + p_{i,L_i}$. On the example of Figure 2, job J_1 covers jobs J_2, J_3, J_4, J_5; job J_2 covers J_3 and J_4; and job J_5 covers J_6. Notice that a job that is not covered has $P_i = 1$.

In order to maximize the average weighted probability to execute the jobs, our aim is to compute a feasible schedule that maximizes the following criterion: $\sum_{i=1}^{n} w_i P_i$. In the classical three-field notation of Graham et al. [5], we denote this problem by $1|r_i, \tilde{d}_i, mc, mu| \sum_{i=1}^{n} w_i P_i$ (where mc stands for mixed-criticality and mu for match-up, see Section 2). Notice that the considered optimization criterion is non-regular, i.e. left-shifted schedules are not dominant.

2 Application and related work

Mixed-criticality framework was first introduced by Vestal [7], to schedule functionalities with different criticalities, on embedded systems. Each job J_i has a criticality level

$L_i \in \mathbb{N}$. The greater is L_i the more critical is J_i. For each job, several estimations of its processing time are considered, one for each level of criticality: each job J_i has L_i different processing time estimations. Optimistic estimations reflect the average, more realistic, behavior of the jobs. Pessimistic (and thus safer) estimations represent worst case executions. In order to achieve the tradeoff between safety guarantees and efficient resource usage, the multiple processing time estimations of a job are taken into account, following the current criticality level of the system. Initially, the criticality level of the system is 1. As soon as some job is executed for longer than its processing time estimation corresponding to the current level of the system, the criticality of the system increases by one unit. When the criticality level of the system is equal to some value l, the execution of all the jobs with criticality level at least equal to l must be guaranteed, while jobs with a lower criticality level can be rejected, in order to execute higher criticality jobs.

Most of the previous works on mixed-criticality [3] consider systems where jobs are scheduled in an event-triggered manner, i.e. fixed priority or EDF policies are applied online by the operating system scheduler. In such a context, for safety reasons, if the system switches to a higher criticality level, it does not switch back again to a lower level. For instance, if the system switches to criticality level 2, then no jobs with criticality level 1 are executed any more. This can lead to an important waste of resources, especially when the criticality level of the system is raised at the beginning. We consider here a different framework, where arrival times (release dates) and deadlines of the jobs are known in advance. We adopt the task model from Hanzalek et al. [6] that, following the idea of match-up scheduling [1,2], allows switching back to a lower criticality level, to improve the efficiency of resource usage. A practical application of such a model is scheduling of messages in automotive embedded systems. Jobs are nonpreemptive, since the particular structure of messages does not allow resuming their sending after interruption. European standards define four criticality levels, corresponding (from highest to lowest criticality) to chassis, engine, driver's assistance and infotainment components. While Hanzalek et al. [6] minimize the makespan criterion, in this work we optimize instead a new criterion, related to jobs execution probabilities.

3 Complexity and algorithms

Problem $1|r_i, \tilde{d}_i, mc, mu| \sum_{i=1}^{n} w_i P_i$ is strongly NP-hard. Indeed, even the problem $1|r_i, \tilde{d}_i, mc, mu|feasibility$ of finding a feasible sequence is strongly NP-hard, since it is a generalization of the strongly NP-hard problem "Sequencing with release times and deadlines" [4]. We show that the special case $1|r_i, \tilde{d}_i, mc, mu, ord| \sum_{i=1}^{n} w_i P_i$ where the jobs sequence order is given is weakly NP-hard. Indeed, we show that even the problem $1|\tilde{d}_i = \tilde{d}, mc, 2lev, mu, ord| \sum_{i=1}^{n} w_i P_i$ with given jobs sequence order, with two criticality levels, no release dates and a common deadline is NP-hard (reduction of Knapsack problem). Moreover, we provide a pseudopolynomial time algorithm (dynamic programming) for $1|r_i, \tilde{d}_i, mc, mu, ord| \sum_{i=1}^{n} w_i P_i$.

Finally, for problem $1|r_i, \tilde{d}_i, mc, mu| \sum_{i=1}^{n} w_i P_i$ we propose a dedicated Branch and Bound algorithm. During the search, a node can be pruned for two reasons: it does not lead to any feasible sequence, or it does not lead to an optimal schedule (w.r.t. $\sum_{i=1}^{n} w_i P_i$). The first pruning rule consists in applying elimination rules to each criticality level separately, as each criticality level corresponds to a scheduling problem with known processing times. The second pruning rule relies on an upper

bound on the payoff of a jobs sequence, which is a new dedicated bound for this problem. Each node of the search tree represents a (partial) sequence of jobs. Each node is obtained by appending an unsequenced job at the end of the jobs sequence of its parent node. At each leaf node (i.e. representing a complete sequence of the jobs), a corresponding optimal schedule is computed by solving a Mixed Integer Linear Program for the problem with given jobs sequence order. Experiments on randomly generated instances show that the algorithm solves most instances with up to 90 jobs in a reasonable time.

4 Conclusion

We considered a mixed-criticality scheduling problem with a new, non-regular, criterion. The problem is known to be strongly NP-hard, even in its feasibility version. We studied the problem where the sequence of jobs is given and showed that it is weakly NP-hard, by providing a pseudopolynomial time algorithm and by showing that the problem with given jobs order, with two criticality levels, no release dates and a common deadline is already NP-hard. Finally, we designed a dedicated upper bound for the general problem (no given jobs sequence order) as well as a MILP formulation for the special case with fixed sequence of jobs; which were used in the implementation of a Branch and Bound algorithm for the general problem.

Further work includes designing heuristic algorithms to solve larger instances. This could be done by exploiting the branch and bound structure to explore feasible sequences, while evaluating sequences at leaf-nodes with a heuristic algorithm instead of a MILP.

Acknowledgements The authors would like to thank Andrei Furtuna, student at Czech Technical University, for his implementation of the Branch and Bound algorithm. This work was supported by the Ministry of Education of the Czech Republic under the project "Support for improving R & D teams and the development of intersectoral mobility at CTU in Prague" number CZ.1.07/2.3.00/30.0034.

References

1. M Selim Akturk and Elif Gorgulu. Match-up scheduling under a machine breakdown. *European journal of operational research*, 112(1):81–97, 1999.
2. James C Bean, John R Birge, John Mittenthal, and Charles E Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.
3. Alan Burns and Rob Davis. Mixed criticality systems: A review. Technical report, Technical report, University of York, 2014.
4. Michael R Garey and David S Johnson. *Computers and Intractability: a guide to NP-completeness*. WH Freeman New York, 1979.
5. Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.
6. Zdenek Hanzalek, Tomas Tunys, and Premysl Sucha. Non-preemptive mixed-criticality match-up scheduling problem. Technical report, Technical report, submitted to Journal of Scheduling, 2014.
7. Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

MISTA 2015

Student Scheduling Problem At Université de Technologie de Compiègne

Taha Arbaoui · Abdelhadi Azouni · Jean-Paul Boufflet · Aziz Moukrim

Abstract Students at Université de Technologie de Compiègne (UTC) are allowed to design their curriculum by selecting courses from large course pools. There are today six engineering degrees (high-level master). The assignment is performed using a heuristic designed some years ago. The number of students, courses and curricula increased over the years and this heuristic reached its limit. Time-consuming manual post-processing is required to obtain the individual planning for each student not assigned by the heuristic. We propose a MIP model to maximize the number of assigned students while managing limited resources. Optimal solutions are achieved for each instance and are strictly better than those computed by the heuristic. We investigated the impact of clique inequalities on computing times.

1 Introduction

University timetabling problems are faced by universities each year. Course and exam timetabling are the most encountered problems in the literature. According to institutions, the Student Scheduling Problem (SSP) is either explicitly or implicitly included in the design of course timetabling. We refer the reader to Schaerf [6] for detailed review on course and exam timetabling.

Despite the fact that SSP covers a large variety of real situations, the core problem is to assign the maximum number of students to sections of their chosen courses while having for each assigned student an individual timetable that is conflict-free. Secondary objectives such as balancing the number of students in the sections of courses may also be encountered.

SSP is not as widely discussed as course and exam timetabling problems. Laporte and Desroches [3] provided a mathematical model that considers a number of hard and soft constraints. The constraints were grouped according to the population concerned i.e. students, university, teachers, etc. Cheng et al. [1] referred to their SSP problem as part of solving the American high school timetabling problem. The goal is

Taha Arbaoui, Abdelhadi Azouni, Jean-Paul Boufflet, Aziz Moukrim
Sorbonne universités, Université de Technologie de Compiègne, CNRS, laboratoire Heudiasyc UMR 7253, CS 60319, 57 avenue de Landshut 60203 Compiègne cedex France
E-mail: {taha.arbaoui,azouniab,jean-paul.boufflet,aziz.moukrim}@utc.fr

to respect the list of student preferences while having a conflict-free timetable. They demonstrated that SSP is an NP-hard problem and presented a multi-commodity flow formulation. Van den Broek et al. [8] presented the SSP at the TU Eindhoven and provided two problem formulations. They presented complexity results for variants of the problem. For further solving approaches and discussions on SSP, we refer the reader to [2, 5, 7].

2 Problem description

An academic year at Université de Technologie de Compiègne (UTC) is split into two semesters. There are five engineering degrees decomposed into a two-year cycle called "fundamental studies" followed by a three-year cycle called "major studies". There exists a master program with four majors and fourteen specializations. Eight departments share the same premises. Every semester, a set of timetables of about three hundred Units of Value (UV) is built and there are in average two thousand and five hundred students. A UV is composed of activities and for each activity there is a set of sections to be scheduled either weekly or fortnightly. Each section corresponds to events in the timetable and an event is usually a time slot, a room and a teacher. There are a thousand and four hundred sections in average. Timetables are built based on enrollment forecasts while taking into account constraints related to teachers and rooms. The choice of UVs from different curricula is allowed.

The set of timetables are made available for students at each beginning of semester. Each student chooses at most seven UVs and has to check that at least a conflict-free timetable exists using the set of timetables. Enrollments being known, timetables are updated, few sections are created and some others are cancelled according to actual enrollments and resource availabilities. Updates of the timetables being performed, we have no guarantee that all students can be scheduled. Today, the heuristic cannot find a schedule for all students. This heuristic operates in two steps. First a greedy algorithm tries to schedule a maximum number of students but some students are left unassigned. Second a hill climbing based stage is performed to improve the solution. Our goal is to optimally schedule a maximum number of students to one section for each activity of the chosen UVs while optimizing the usage of certain resources simultaneously required by some sections.

3 Mathematical Model

We denote $\mathcal{S}$ the set of students. The set of different activities is denoted $\mathcal{A}$, and, for a student s, $\mathcal{A}_s$ is the set of activities of chosen UVs. $\mathcal{K}$ and $\mathcal{K}_a$ denote the set of sections and the set of sections for an activity a respectively. The set of couples of sections $[k, k']$ for which k and k' require a same disjunctive resource is denoted $\mathcal{F}$. Decision Boolean variables $T_s = 1$ when student s is assigned into one section for each activity from $\mathcal{A}_s$, $Y_{sa} = 1$ if student s is assigned to a section of activity a, and $Z_{sak} = 1$ when student s is assigned to section k of activity a. Parameter p_k is the capacity of section k (i.e the maximum number of students one can assign to section k). For a student s enrolled in two sections, they are in conflict if student s cannot be assigned to both simultaneously. $m_{kk'} = 1$ if sections k and k' are in conflict,

0 otherwise. Values $m_{kk'}$ are entries of the adjacency matrix of the general conflict graph between sections.

We propose the following formulation to maximize the number of students scheduled into one section for each activity of chosen UVs:

Maximize:

$$\sum_{s \in \mathcal{S}} T_s \tag{1}$$

subject to:

$$\forall s \in \mathcal{S} \qquad |\mathcal{A}_s| T_s \leq \sum_{a \in \mathcal{A}_s} Y_{sa} \tag{2}$$

$$\forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}_s \qquad Y_{sa} = \sum_{k \in \mathcal{K}_a} Z_{sak} \tag{3}$$

$$\forall a \in \mathcal{A} \quad \forall k \in \mathcal{K}_a \qquad \sum_{s \in \mathcal{S}} Z_{sak} \leq p_k \tag{4}$$

$$\left.\begin{array}{l} \forall s \in \mathcal{S} \\ \forall a \in \mathcal{A}_s \\ \forall k \in \mathcal{K}_a \end{array}\right\} \quad \sum_{\substack{a' \in \mathcal{A}_s \\ k' \in \mathcal{K}_{a'} \\ k' \neq k}} m_{kk'} Z_{sa'k'} + \Big(\sum_{\substack{a' \in \mathcal{A}_s \\ k' \in \mathcal{K}_{a'} \\ k' \neq k}} m_{kk'} \Big) Z_{sak} \leq \sum_{\substack{a' \in \mathcal{A}_s \\ k' \in \mathcal{K}_{a'} \\ k' \neq k}} m_{kk'} \tag{5}$$

$$\left.\begin{array}{l} \forall [k,k'] \in F \quad \text{let } a, a' \text{such that:} \\ k \in K_a,\ k' \in K_{a'} \qquad \forall s \in S_a \end{array}\right\} \quad \sum_{s' \in S_{a'}} Z_{s'a'k'} + p_{k'} Z_{sak} \leq p_{k'} \tag{6}$$

$$T_s, Y_{sa}, Z_{sak} \in \{0,1\} \tag{7}$$

Equations (2) set $T_s = 1$ if student s is assigned into a unique section for each chosen activity. Equations (3) link the decision variables Y_{sa} and Z_{sak}. Section capacities are enforced using Equations (4). Conflicts between two sections k and k' are enforced using Equations (5). Resource disjunction between sections is managed using Equations (6). The model succeeds to attain optimal solutions on ten real instances of our university while managing disjunctive resources whereas the heuristic cannot deal with. The results are strictly better than those obtained by the heuristic.

To speed up the solving process we investigated the impact of clique inequalities on computation times. When there is a large number of cliques, we focus on maximal cliques. Unfortunately, since we may also have a large number of maximal cliques, adding all the maximal clique inequalities may slow down the solving process. We studied on our problem the impact of clique inequalities by fixing a threshold on the clique sizes to be considered.

4 Investigating the impact of clique valid inequality

A clique in the general conflict graph may correspond to many students. So, we use the induced conflict graph relative to a student s to build C_s: the set of maximal cliques for sections $\cup_{a \in A_s} K_a$ and we propose:

$$\forall s \in S \quad \forall c \in \mathcal{C}_s \qquad \sum_{k \in c,\ k \in K_a} Z_{sak} \leq 1 \tag{8}$$

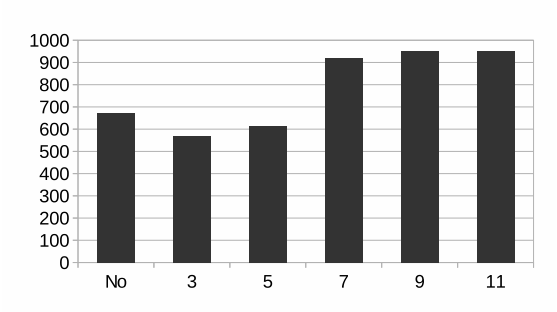


Fig. 1 Impact of clique valid inequalities

Each clique c is a set of sections that cannot be allocated to the same student at the same time. Tests were done using CPLEX 12.5 IBM (2012) MIP solver with a single thread, gcc 4.4.7, on a machine with an Intel core i7 950@3.07GHz and 24GB of RAM under linux fedora core 19. We used [4] to compute each C_s and it takes few seconds for all students. For some instances there are more than 50 000 maximal cliques, so we studied the impact of using maximal clique sizes greater than or equal to a threshold. Figure 1 shows the impact on computing times of adding all the maximal cliques larger than or equal to a threshold. A bar corresponds to the total computing time for the instances reported in seconds. Bar "No" stands for no clique inequalities. The other bars show the results obtained when using all the maximal cliques greater than or equal to 3, 5, 7, 9 and 11 respectively. The best results are achieved by adding all the maximal clique inequalities where clique sizes are larger than or equal to 3. We are investigating pretreatments to avoid infeasible configurations of sections for a student so as to reduce the number of cliques.

5 Conclusion

The Student Scheduling Problem at our university is a consequence of the large choice of courses allowed. Each semester a heuristic is used to solve the problem. We proposed a MIP formulation which permits to optimally assign students and to manage limited resources. This formulation is reinforced by clique valid inequalities extracted from the students' conflict graph between sections. We are investigating pretreatments to avoid infeasible configurations of sections for students to reduce the number of cliques. Detailed results will be presented and commented on.

Acknowledgment

This work was carried out in the framework of the Labex MS2T, which was funded by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02). We would also like to thank la Direction des Systèmes d'informations (DSI) and la Direction à la Formation et à la Pédagogie (DFP) in Université de Technologie de Compiègne (UTC) for their collaboration in this work.

References

1. E. Cheng, S. Kruk, and M. Lipman. Flow formulations for the student scheduling problem. In *Practice and Theory of Automated Timetabling IV*, volume 2740, pages 299–309. 2003.
2. R. Feldman and M. C. Golumbic. Constraint satisfiability algorithms for interactive student scheduling. In *Proceedings of the 11th international joint conference on Artificial intelligence*, volume 2, pages 1010–1016, 1989.
3. G. Laporte and S. Desroches. The problem of assigning students to course sections in a large engineering school. *Computers & operations research*, 13(4):387–394, 1986.
4. P. R. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207, 2002.
5. G. Sabin and G. Winter. The impact of automated timetabling on universities-a case study. *Journal of the operational Research Society*, 37(7):689–693, 1986.
6. A. Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127, 1999.
7. A. Tripathy. Computerised decision aid for timetabling—a case analysis. *Discrete applied mathematics*, 35(3):313–323, 1992.
8. J. Van den Broek, C. Hurkens, and G. Woeginger. Timetabling problems at the TU eindhoven. *European Journal of Operational Research*, 196(3):877–885, 2009.

MISTA 2015

Shortest k-unit Cycle in a Multiple Part-Type Robotic Cell

Vahid Eghbal Akhlaghi • Hakan Gultekin • Betul Coban

1 Introduction

Many diverse industries use robotic cells. The cells consist of an input device, a series of processing stages, an output device, and robots for material handling within the cell. If the robotic cell produces different types of parts, we refer to it as a multiple part-type cell (in contrast to single part-type cells) [1]. The amount of time needed to produce a Minimal Part Set (MPS) is called the *cycle time*. A cycle in which *k* parts are produced is called a k-unit cycle. Brauner et al. [2] prove that 1-unit cycles are not necessarily optimal for m-machine robotic cells producing single parts when $m \geq 4$ and Hall et al. [3] prove the same result is valid for multiple part type production even with m=2. The problem is then to find the shortest multi-unit cyclic schedule for the robot and the part sequence for the MPS that jointly minimize the cycle time in a multiple part-type robotic cell. They solved this problem for m=2 and m=3. Sriskandarajah et al. [4] proved this problem to be NP-hard for arbitrary m. Nevertheless, there is neither an exact mathematical model nor a heuristic algorithm in the literature, so far.

Gultekin et al. [5] propose new lower bounds for the 1 and 2-unit robot move cycles in flexible robotic cells for the flow-shop type robot move cycles. It is proved by Hurink and Knust [6] that a single gripper robot job shop scheduling problem is NP-hard. They use a Tabu Search (TS) algorithm, in the form of an extended TSP, to solve the single-machine scheduling problem. Their results illustrate that the TS algorithm finds a good solution in a short amount of time. Geismar et al. [7] propose that developing effective heuristics might be a fruitful approach to the multiple part-types robotic cell scheduling problem.

2 Problem Definition and Mathematical Model

A typical simple robotic cell contains M processing machines: M_1, M_2,... , M_m. Robot activity A_i, defined by Crama and Van de Klundert [8], consists of; unloading a part from M_i, traveling from M_i to M_{i+1} and loading the part onto M_{i+1}.

There are several pickup criteria in robotic cells. This study concentrates on free pickup criterion. Then the only restriction says that a part, which its processing time on M_i is completed, cannot be loaded onto M_{i+1} for its next processing unless M_{i+1} is unoccupied. The robot's travel time between adjacent machines M_{i-1} and M_i, equals δ, and it is additive. That is, the travel time between any two machines M_i, M_j is $|i - j|\delta$. In a k-unit cycle, the robot takes *k* parts in total,

Vahid Eghbal Akhlaghi
Middle East Technical University, Department of Industrial Engineering, Ankara, Turkey
E-mail: vahid.akhlaghi@metu.edu.tr

Hakan Gultekin
TOBB University of Economics and Technology, Department of Industrial Engineering, Ankara, Turkey
E-mail: hgultekin@etu.edu.tr

Betul Coban
TOBB University of Economics and Technology, Department of Industrial Engineering, Ankara, Turkey
E-mail: bcoban@etu.edu.tr

from the input device. Whenever all the robot activities are repeated exactly *k* times and the robot returns to its initial state of the cycle, the k-unit cycle is completed and exactly k parts are produced. The necessary time to produce k parts in this way is called *k-unit cycle time*. In the mathematical model, we assign each robot activity to a specific position *p*.

Notations:

Sets

n index of product
m index of machine
p index of position
i index of activity
l index of repetition

Parameters:

P_m^n Processing time of product n on machine m
ε Loading/Unloading time of machine
δ Travel time between consecutive machines

Variables

x_{jl}^p Binary variable: 1 if repetition *l* of activity A_j is assigned to position p
z_{jl}^n Binary variable: 1 if repetition *l* of activity A_j transport part n
y_{jlpn} Binary variable: 1 if product n is assigned to position p in repetition *l* of activity A_j
CT Cycle time
T_p Starting time of the activity assigned to position p

In the proposed model, after assigning the activities to a given number of machines and finding the starting time of the last robot activity, the minimum cycle time is achieved.

Objective *min CT*

s.t.

$$\sum_{p=1}^{n(m+1)} x_{jl}^p = 1, \qquad \forall j \in \{0,\dots,m\}, \forall l \in \{1,\dots,n\} \tag{1}$$

$$\sum_{l=1}^{n}\sum_{j=0}^{m} x_{jl}^p = 1 \qquad \forall p \in \{1,\dots,n(m+1)\} \tag{2}$$

$$\sum_{l=1}^{n}\sum_{p=0}^{n(m+1)} x_{jl}^p = n \qquad \forall j \in \{0,\dots,m\} \tag{3}$$

$$\sum_{l=1}^{n} x_{jl}^p \le 1 \qquad \forall j \in \{0,\dots,m\}, \forall p \in \{1,\dots,n(m+1)\} \tag{4}$$

$$\sum_{l=1}^{n} z_{jl}^n = 1 \qquad \forall j \in \{0,\dots,m\}, \forall n \in \{1,\dots,n\} \tag{5}$$

$$y_{jlpn} \ge x_{jl}^p + z_{jl}^n - 1 \qquad \forall p \in \{1,\dots,n(m+1)\}, \forall j \in \{0,\dots,m\}, \forall n,l \in \{1,\dots,n\} \tag{6}$$

$$x_{jl}^p \ge y_{jlpn} \qquad \forall p \in \{1,\dots,n(m+1)\}, \forall j \in \{0,\dots,m\}, \forall n,l \in \{1,\dots,n\} \tag{7}$$

$$z_{jl}^n \ge y_{jlpn} \qquad \forall p \in \{1,\dots,n(m+1)\}, \forall j \in \{0,\dots,m\}, \forall n,l \in \{1,\dots,n\} \tag{8}$$

$$T_p \ge T_r + 2\varepsilon + \delta + \sum_{l=1}^{n}\sum_{j=0}^{m}\sum_{n=1}^{n} y_{jlpn}\, P_j^n + M\left(\sum_{l=1}^{n} x_{jl}^p + \sum_{l=1}^{n} x_{(j-1)l}^r - 2\right)$$
$$\forall p,r \in \{1,\dots,n(m+1)\}: p > r, \forall j \in \{1,\dots,m\} \tag{9}$$

$$CT + T_p \ge T_r + 2\varepsilon + \delta + \sum_{l=1}^{n}\sum_{j=0}^{m}\sum_{n=1}^{n} y_{jlpn}\, P_j^n + M\left(\sum_{l=1}^{n} x_{jl}^p + \sum_{l=1}^{n} x_{(j-1)l}^r - 2\right)$$
$$\forall p,r \in \{1,\dots,n(m+1)\}: p < r, \forall j \in \{1,\dots,m\} \tag{10}$$

$$T_{p+1} \ge T_p + 2\varepsilon + \delta + \delta|j1-(j+1)| + M\left(\sum_{l=1}^{n} x_{jl}^p + \sum_{l=1}^{n} x_{j1\,l}^{p+1} - 2\right)$$
$$\forall p \in \{1,\dots,(n(m+1)-1)\}, \forall j,j1 \in \{0,\dots,m\}: j \ne j1 \tag{11}$$

$$CT \ge T_{n(m+1)} + 2\varepsilon + \sum_{j=0}^{m} \sum_{l=1}^{n} \delta(j+2) x_{jl}^{n(m+1)}$$
$$\forall n \in \{0,\dots,m\}, \forall m \in \{1,\dots,m\}, \forall p \in \{1,\dots,n(m+1)\} \tag{12}$$

$$T_0 = 0 \tag{13}$$

$$\sum_{p=1}^{n(m+1)} p x_{jl}^p \le \sum_{p=1}^{n(m+1)} p x_{j\,l1}^p \qquad \forall j \in \{0,\dots,m\}, \forall l,l1 \in \{1,\dots,n\} \tag{14}$$

$$\sum_{l1=1}^{n}\sum_{p1=p}^{r} x_{(j-1)l1}^{p1} \ge 1 - M(2 - x_{jl}^p + x_{j\,(l+1)}^r)$$
$$\forall p,r \in \{1,\dots,n(m+1)\}: r > p, \forall j \in \{1,\dots,m\}, \forall l \in \{1,\dots,n\} \tag{15}$$

$$\sum_{l1=1}^{n}\sum_{p1=p}^{r} x_{(j+1)l1}^{p1} \geq 1 - M\left(2 - x_{jl}^{p} + x_{j\,(l+1)}^{r}\right)$$
$$\forall p, r \in \{1, \dots, n(m+1)\}: r > p, \forall j \in \{0, \dots, (m-1)\}, \forall l \in \{1, \dots, n\} \quad (16)$$

Since this problem is NP-Hard [6], we proposed a heuristic approach using Tabu Search (TS) plus Genetic Algorithm (GA), which cooperatively find approximate optimal solutions.

3 The Proposed Heuristic Approach

The algorithm consists of two parts, the first a GA that generates a variety of part sequences and the second, a TS that tries to find the best robot sequence. A simple and well-known single point crossover is applied in our GA as the crossover operator. Besides, a simple swap for two randomly selected numbers is considered as the mutation operator. On the other hand, the developed TS generates the corresponding robot sequences by changing the position of a robot activity to another feasible position in the sequence. We found an upper bound for the furthest feasible move. In each iteration, the objective function value for each of these feasible and non-tabu moves are calculated. If the value is superior to the incumbent objective function value, then it will be determined as the new incumbent. Any selected move will be removed from the tabu list after three iterations.

The proposed algorithm is coded in C++. We made a computational study by an experiment design on problem parameters. The initial tests on the algorithm show that it finds high quality solutions in reasonable CPU times.

Acknowledgement

This research is based on the support of The Scientific and Technological Research Council of Turkey (TUBITAK) under grant number 213M435.

References

1. Dawande, M., Geismar, H., Sethi, S., and Sriskandarajah, C., "SEQUENCING AND SCHEDULING IN ROBOTIC CELLS: RECENT DEVELOPMENTS," J. Scheduling, vol. 8, no. 5, pp. 387–426, 2005.
2. Brauner, N., and Finke, G., "CYCLES AND PERMUTATIONS IN ROBOTIC CELLS," Mathematical and Computer Modeling, 34, 565–591, 2001.
3. Hall, N.G., Kamoun, H., and Sriskandarajah, C., "SCHEDULING IN ROBOTIC CELLS: classification, two and three machine cells," Operations Research, 45, 421–439, 1997.
4. Sriskandarajah, C., Hall, N.G., Kamoun, H., and Wan, H., "Scheduling large robotic cells without buffers," Annals of Operations Research, 76, 287–321, 1998.
5. Gultekin H, Akturk MS, and Karasan OE. "SCHEDULING IN A THREE-MACHINE ROBOTIC FLEXIBLE MANUFACTURING CELL," Computers and Operations Research; 34:24, 63–77, 2007.
6. Hurink, J. and Knust, S., "A TABU SEARCH ALGORITHM FOR SCHEDULING A SINGLE ROBOT IN A JOB-SHOP ENVIRONMENT, Discrete Applied Mathematics, Vol.119, 181-203, 2002.
7. Geismar, N., Dawande, M., and Sriskandarajah, C., "PRODUCTIVITY IMPROVEMENT FROM USING MACHINE BUFFERS IN DUAL-GRIPPER CLUSTER TOOLS," [J]. IEEE Transactions on Automation Science and Engineering, 1(8):29-41, 2011.
8. Crama, Y., and Van de Klundert, J., "CYCLIC SCHEDULING OF IDENTICAL PARTS IN A ROBOTIC CELL," Operations Research, 6, 952–965, 1997.

MISTA 2015

A branch-and-reduce exact algorithm for the single machine total tardiness problem

Federico Della Croce · Michele Garraffa · Lei Shang · Vincent T'kindt

1 Introduction

We consider the one-machine total tardiness $1||\sum T_j$ problem where a jobset $N = \{1, 2, \ldots, n\}$ of n jobs must be scheduled on a single machine. For each job j, we define a processing time p_j and a due date d_j. The problem calls for arranging the jobset in a sequence $S = (1, 2, \ldots, n)$ so as to minimize $T(N, S) = \sum_{j=1}^{n} T_j = \sum_{j=1}^{n} \max\{C_j - d_j, 0\}$, where $C_j = \sum_{i=1}^{j} p_i$.

The $1||\sum T_j$ problem is **NP**-hard in the ordinary sense [2]. It has been extensively studied in the literature and many exact procedures ([1,5,6,8]) have been proposed. The current state-of-the-art exact method of [8] dates back to 2001 and solves to optimality problems with up to 500 jobs. All these procedures are search tree approaches. On the other hand, also dynamic programming algorithms were considered. In [5] a pseudo-polynomial dynamic programming algorithm was proposed running with complexity $O(n^4 \sum p_i)$. The design of exact methods for NP-hard combinatorial optimization problems has always been a challenging issue. Here, we study their application in the context of worst-case analysis and look for best results in terms of exponential time complexity as a function of the number of variables (in our case jobs). In this context, classical search tree algorithms are more commonly defined as branch-and-reduce algorithms as typically a branch in the search tree induces the generation of two or more subproblems each with a reduced number of variables with respect to the original problem. Consider now a combinatorial optimization problem that can be represented by means of n variables/jobs. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, using notations in [10], for an integer n, we express running-time bounds of the form $p(n) \cdot T(n)$ as $O^*(T(n))$, the asterisk meaning that we ignore polynomial factors. We denote by $T(n)$ the worst case time required to exactly solve the considered combinatorial optimization problem with n jobs. Notice

F. Della Croce
D.A.I., Politecnico di Torino, Italy
E-mail: federico.dellacroce@polito.it

M. Garraffa
D.A.I., Politecnico di Torino, Italy
E-mail: michele.garraffa@polito.it

L. Shang
Université F. Rabelais de Tours, France
E-mail: lei.shang@etu.univ-tours.fr

V. T'kindt
Université F. Rabelais de Tours, France
E-mail: tkindt@univ-tours.fr

that, in this context, the dynamic programming of [5] cannot be considered due to the pseudopolynomial time complexity being function of $\sum p_i$). On the other hand, the standard technique of doing dynamic programming across the subsets [10] is applicable also to the total tardiness model and runs with complexity $O^*(2^n)$ but requires also $O^*(2^n)$ space, that is it requires both exponential time and space. To the authors knowledge, there is currently no available exact algorithm for the total tardiness problem running in $O^*(c^n)$ (c being some constant) and polynomial space. In this work we propose a branch-and-reduce exact exponential algorithm requiring $O^*((1+\sqrt{2})^n)$ time and polynomial space.

We recall (see, for instance, [4]) that, if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \leq \sum T(n-r_i) + O(p(n))$, we have $\sum T(n-r_i) + O(p(n)) = O^*(\alpha(r_1, r_2, \ldots)^n)$ where $\alpha(r_1, r_2, \ldots)$ is the largest zero of the function $f(x) = 1 - \sum x^{-r_i}$.

2 Main result

We make use of the following notation. Given the jobset $N = \{1, 2, \ldots, n\}$, let $(1, 2, \ldots, n)$ be an SPT sequence (where $i < j$ whenever $p_i = p_j$ and $d_i \leq d_j$). Let also $([1], [2], \ldots, [n])$ be an EDD sequence (where $[i] < [j]$ whenever $d_i = d_j$ and $p_i \leq p_j$). As the cost function is a regular performance measure, we know that in the optimal solution the jobs are processed with no interruption starting from time zero. Let $p(B) = \sum_{k \in B} p_k$. Let B_j and A_j be the sets of jobs that precede and follow job j in an optimal sequence. Correspondingly, $C_j = p(B_j) + p_j = p(N - A_j)$.

The main known theoretical properties are the following.

Property 1 *[3] Consider two jobs i and j, $i < j$. Then, $i \to j$ if $d_i \leq \max\{d_j, C_j\}$, else $j \to i$ if $d_i + p_i > C_j$.*

Property 2 *[5] Let job n in SPT correspond to job $[k]$ in EDD. Then, job n can be set only in position $h \geq k$ and the jobs preceding and following k are uniquely determined as $B_n = \{[1], [2], \ldots, [k-1], [k+1], \ldots, [h]\}$ and $A_n = \{[h+1], \ldots, [n]\}$.*

Property 3 *[5–7] Let $C_n(h) = \sum_{j=1}^{h} p_{[j]}$ be the completion time of job n when set in position $h \geq k$. Then, job n ($[k]$) cannot be set in such position if:*

(a) $C_n(h) \geq d_{[h+1]}$, $h < n$;
(b) $C_n(h) < d_{[r]} + p_{[r]}$, for some $r = k+1, \ldots, h$.

Property 4 *[7] For any pair of adjacent positions $i, i+1$ that can be assigned to job n, at least one of them is eliminated by Property 3.*

Similar decomposition and elimination properties hold when the smallest due date job [1] is considered (see [1]).

Consider the following branch and reduce exact algorithm TTBR (Total Tardiness Branch and Reduce) that works as follows.

1. Iteratively branch on the largest processing time job n and assign it to all possible positions $(1, ..., n)$ with n potential branches and correspondingly decompose the problem of each single branch according to Property 2, where all positions (and corresponding branches) satisfying Property 3 are eliminated.

2. If all the leaves of the search tree have been reached, STOP: the best found solution is the optimal one.

Proposition 1 *Algorithm TTBR runs in* $O^*((1+\sqrt{2})^n) \approx O^*(2.4142^n)$ *time and polynomial space.*

Proof Whenever job n is assigned to position k, two subproblems with size $k-1$ and $n-k$ are generated. Hence, in the worst case, $2n-4$ subproblems are generated considering each size $k = 2, ..., n-1$. This induces to a recursion of the type $T(n) \leq 2T(n-1)+2T(n-2)+...+2T(2)+O(p(n)) \leq 2\sum_{i=1}^{n-1} T(n-i)+O(p(n))$ corresponding to $T(n) = O^*(3^n)$. Besides, due to Property 4, for each pair of adjacent positions $i, i+1$, at least one of them must be discarded. The worst case occurs then by keeping the subproblems of sizes $n-1, n-3, n-5$ and so on and considering n to be odd. This induces to a recursion of the type $T(n) \leq 2T(n-1)+2T(n-3)+...+2T(4)+2T(2)+O(p(n)) \leq 2\sum_{i=1}^{\frac{n-1}{2}} T(n-2i+1)+O(p(n))$ that corresponds to $O^*((1+\sqrt{2})^n) \approx O^*(2.4142^n)$. □

Algorithm TTBR can be significantly improved by exploring the fact the for any given subset of k jobs assigned to the first (last) k positions of the sequence it is sufficient to retain only the dominant one. For k sufficiently small, a limited case analysis allows to discard further branches and correspondingly improve the time complexity still requiring polynomial space. The improved approach will be presented at the Conference. The proposed approach can be extended to every single machine scheduling problem that has a decomposition property similar to the Property 2: in that case, an exact exponential algorithm requiring no more than $O(3^n)$ time and polynomial space can be immediately derived along the lines of Algorithm TTBR.

References

1. F. Della Croce, R. Tadei, P. Baracco and A. Grosso (1998), "A new decomposition approach for the single machine total tardiness scheduling problem", *Journal of the Operational Research Society* 49, 1101–1106.
2. J. Du and J. Y. T. Leung (1990), "Minimizing total tardiness on one machine is NP–hard", *Mathematics of Operations Research* 15, 483–495.
3. H. Emmons (1969), "One-machine sequencing to minimize certain functions of job tardiness", *Operations Research* 17, 701–715.
4. Eppstein D., Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In Proc. Symposium on Discrete Algorithms, SODA01, 329-337 (2001).
5. E. L. Lawler (1977), "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness", *Annals of Discrete Mathematics* 1, 331–342.
6. C. N. Potts and L. N. Van Wassenhove (1982), "A decomposition algorithm for the single machine total tardiness problem", *Operations Research Letters* 5, 177–181.
7. W. Szwarc (1993), "Single machine total tardiness problem revisited", Y. Ijiri (ed.), *Creative and Innovative Approaches to the Science of Management*, Quorum Books, Westport, Connecticut (USA), 407–419.
8. W. Szwarc, A. Grosso and F. Della Croce (2001), "Algorithmic paradoxes of the single machine total tardiness problem", *Journal of Scheduling* 4, 93–104.
9. W. Szwarc and S. Mukhopadhyay (1996), "Decomposition of the single machine total tardiness problem", *Operations Research Letters* 19, 243–250.
10. Woeginger, G. J., Exact algorithms for NP-hard problems: a survey. In M. Juenger, G. Reinelt, and G. Rinaldi, (eds.) Combinatorial Optimization - Eureka! You shrink!, volume 2570 of Lecture Notes in Computer Science, 185-207, Springer-Verlag, 2003.

MISTA 2015

Maintenance planning in a stochastic job shop

Marjan van den Akker · Han Hoogeveen · Jonathan Lukkien

1 Introduction

The job shop scheduling problem is widely used to model practical production scheduling problems in a make-to-order setting. This problem is defined as follows. There are m machines, denoted by $M_1, \ldots, M_m$, which must process a set of n jobs, denoted by $J_1, \ldots, J_n$. Each machine M_i $(i = 1, \ldots, m)$ has capacity one, that is, it can handle at most one job at a time; moreover, each machine is assumed to be continuously available from time zero onward. Each job J_j $(j = 1, \ldots, n)$ consists of a *chain* of operations, which implies that the kth operation of J_j cannot start before the $(k-1)$st operation of J_j has been completed. For each operation, we know which machine M_i must execute it and how much time this requires. The goal is to minimize the makespan, which is defined as the time at which the last job has been completed.

The job shop problem is known to be $\mathcal{NP}$-hard in the strong sense; it is also known to be very hard from a computational point of view: enumerative algorithms have difficulties solving problems with more than 20 jobs and 20 machines in a reasonable amount of time. Therefore, many researchers have studied local search methods, like for example tabu search based algorithms ([6] and [5]), simulated annealing based algorithms ([7]), simulated annealing in combination with using commonalities ([3]) and hybrid genetic algorithms ([2] and [4]); all of these studies report that good results are obtained.

In practice, however, the standard assumption that the input is deterministic often does not hold. Van den Akker et al. ([1]) have studied the stochastic variant, in which the processing time of an operation is a stochastic variable with a known distribution. The goal is then to find a schedule, i.e. the order of the operations on the machines,

Marjan van den Akker
Utrecht University
E-mail: j.m.vandenakker@uu.nl

Han Hoogeveen
Utrecht University
E-mail: j.a.hoogeveen@uu.nl

Jonathan Lukkien
Utrecht University
E-mail: j.f.lukkien@gmail.com

with minimum expected makespan; here it is not allowed to make adjustments to the schedule during its execution when the realizations of the processing times become known. In our work, we consider this stochastic variant of the problem, but we assume that we have to plan maintenance on a given machine as well. This maintenance must start between a given earliest and a latest possible start time. If the maintenance operation has not been started at the latest possible start time, then it is forced to start at this time; if some operation is executed by the machine at this time, then the operation is removed, and it gets restarted to be executed from scratch as soon as the maintenance operation has been finished (the so-called preemptive repeat model). The goal is to find a schedule with minimum expected makespan. Again, we are not allowed to make adjustments to the order of the operations with the exception of the maintenance operation, which is planned during the execution of the schedule on basis of the current knowledge of the processing times.

Our contribution. We show how the local search algorithm for the stochastic job shop by [1] can be adapted to include maintenance. Our approach can be easily adjusted to deal with multiple maintenance operations and with maintenance operations of stochastic length. Similarly, our approach can be applied when rescheduling of operations during execution is allowed, but extensive replanning may lead to an excessive run time.

2 Solution approach

The core of our local search algorithm is the same as the one by van den Akker et al. ([1]) (in fact, we used the code of this program as our basis). This algorithm is based on Simulated Annealing with two neighborhoods: a swap of two consecutive operations on the longest path and a left shift operation. In each iteration, we compare the new and incumbent solution in a series of five simulation experiments; to hedge against bad luck, we keep a pool of promising solutions. Finally, we evaluate the quality of each of these solutions by running a series of 1000 discrete-event simulations to determine the expected makespan. The planning of the maintenance takes place within the simulation; it is not part of the local search.

The characteristics of the maintenance operation depend on the instance, but in general are as follows. The maintenance has to be done on exactly one given machine; for this we choose either the machine with largest or with smallest load. The maintenance requires a given amount of time that is either equal to once or twice the average processing time of the operations. The width of the interval during which we can perform the maintenance is chosen to be either two, three, or four times the length of maintenance operation.

Now we come to how to plan the start time of the maintenance within the simulation of a given schedule. Suppose that we have arrived at a possible start time of the maintenance, that is, the corresponding machine that must be maintained has just finished an operation and the current time-point falls within the possible maintenance interval, or we have arrived at the earliest possible time for maintenance and the machine is currently idle. We consider three different possible decision rules, from simple to more involved.

1. Random choice: at each such decision point there is a 50% chance to start maintenance. This is independent of the situation on the machines.

2. One-operation-lookahead: here we take the current situation of the schedule into consideration. First of all, we look at the status of the next operation that the machine must execute. This operation can be either available for processing, its job predecessor can currently be executed, or its job predecessor may have not started yet. In this latter case, we simply start the maintenance, as it is probably not worthwhile to let the machine idle that long when waiting for the next operation to become available. In the other two cases, we look at the next possibility for starting the maintenance. To that end, we compute the expected completion time of the next operation to be scheduled on the maintenance machine; if the probability that this time falls within the maintenance interval is large enough, then we currently postpone maintenance and execute the next operation first, which may introduce idle time.
3. Full look-ahead. In the one-operation-lookahead, we only take the current situation into account and check if there is time to postpone maintenance. In general, it is reasonable to postpone maintenance for as long as possible, but machine idle time should be avoided as well. To make a good choice, we simulate the future in the full look-ahead, that is, we apply a simulation within a simulation. On basis of this inner simulation, we decide whether to postpone the maintenance.

3 Computational experiments

In our computational experiments we want to find out the effect of including maintenance. To that end, we compare the schedules for the situations with and without maintenance and check by how much the makespan increases. Obviously, it is easiest to squeeze in the maintenance in the schedule when it affects the machine with the smallest load, and when the interval for performing maintenance is big. In our preliminary experiments, the one-operation-lookahead and the full-look-ahead seem to be comparable and consistently outperform the random choice rule.

References

1. J.M. van den Akker, C.H.M. van Bokland, and J.A. Hoogeveen (2013). Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela (Eds.). Symposium on Experimental Algorithms SEA 2013, Lecture Notes on Computer Science 7933, pp. 402–413.
2. J.F. Gonçalves, J.J. de Magalhães Mendes, and M.G.C. Resende (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research* 167, 77-95.
3. M.L. Kammer, J.M. van den Akker, J.A. Hoogeveen (2011). Identifying and exploiting commonalities for the job-shop scheduling problem. *Computers and Operations Research, 38*, pp. 1556–1561.
4. A. Moraglio, H. ten Eikelder, and R. Tadei (2005). *Genetic Local Search for Job Shop Scheduling Problem*, Technical Report CSM-435, University of Essex, UK.
5. E. Nowicki and C. Smutnicki (1996). A fast taboo search algorithm for the job shop problem. *Management Science* 42, 797-813.
6. E.D. Taillard (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing* 6, 108-117.
7. T. Yamada and R. Nakano (1996). Job-shop scheduling by simulated annealing combined with deterministic local search. I.H. Osman and J.P. Kelly (Eds.). *Meta-heuristics: theory and applications*. Kluwer academic publishers MA, USA, pp. 237-248.

Budgeted Internet Shopping Optimization Problem (B-ISOP)

Jakub Marszałkowski

1 Introduction

With growing importance of the Internet in business, science, but also everyday life, new applications of operations research emerged. One of the most popular of these areas is related with the Internet advertising, where the first paper supposedly was [1] and recently it proliferated in many topics concerning ads scheduling, optimizing, packing, targeting, etc. Another growing group of research papers is tailoring new 2D packing algorithms for optimization of web pages, whose all elements are obviously rectangles. This includes a pursue for better layouts of web pages (e.g. [8,10]) or just better usability of certain elements, like tag clouds (the newest survey is offered by [11]).

Internet changed a lot in the way we are shopping. The abundance of online shops not only improved the competition for the lowest prices, but also made sales of many niche products possible. More so, all these offers can be searched or even crawled and scrapped into some databases, then serving for example as a price comparator sites. A new way how such comparators could work, Internet Shopping Optimization Problem (*ISOP*) was proposed in [5] and then extended in [2,4,3]. The idea is that instead of comparing prices of single products, user creates a list of the items he wants to buy, and algorithm using database chooses for him the cheapest solution, considering both prices of items in numerous shops, but also delivery costs. This is not possible in current generation of price comparators, where for every product cheapest offerer would be chosen, in worst case causing payment of delivery costs for each item to be separate.

Jakub Marszałkowski
Institute of Computing Science
Poznan University of Technology
ul. Piotrowo 2
60-965 Poznan, Poland
E-mail: jakub.marszalkowski@cs.put.poznan.pl

Table 1 Summary of notation.

M	set of products
N	set of shops
m	number of products
n	number of shops
i	product indicator
j	shop indicator
d_j	delivery price of all products from shop j
y_j	indicator variable for shop j
p_{ij}	cost of product i in shop j
x_{ij}	indicator variable for product i in shop j
v_i	user perceived value of product i
P	budget, limit of total cost

This paper relies on ISOP as proposed in [5], however a somehow reversed version of the problem is considered. The customer still has a list of products he wants to buy, but is limited with a budget, i.e. an amount of money he can spend in total, including costs of products and delivery costs. With that, the customer will not be able to buy all of the items, but wants to maximize the total perceived value of products he will get. This perceived value can represent one of the following concepts: user preferences or priorities, ratings of the products, basic monetary value of the products, etc...

2 Mathematical formulation

The Budgeted Internet Shopping Optimization Problem (*B-ISOP*) can be formulated as follows. The customer wants to buy a set M of m products where for every product i user assigns its perceived value v_i. The customer is limited by a budget P. In a database there is gathered information on set N of n shops: for each shop j its standard delivery price is d_j and for each product i available in

shop j its cost in this shop is p_{ij}. The objective is to maximize the value of products the customer can buy from the shops within the limitation of his budget. This can be stated as:

$$\max \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}v_i$$

$$\text{s.t. } \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} + \sum_{j=1}^{n} d_j y_j \leq P,$$

Additional constraints are required that every product is chosen at most once and that if any product is selected from a shop j, this shops delivery cost will be paid. Finally, indicators are binary.

$$\sum_{j=1}^{n} x_{ij} \leq 1,\ i = 1,\ldots,m,$$

$$0 \leq x_{ij} \leq y_j,\ i = 1,\ldots,m,\ j = 1,\ldots,n,$$

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\},\ i = 1,\ldots,m,\ j = 1,\ldots,n.$$

The proposed problem is NP-hard as it will be demonstrated by proving NP-completeness of its decision counterpart - problem B1.
Problem B1: Is it possible to buy a set $M' \subset M$ of m' products with value of at least V within the budget P?

Proof B1 is in NP because NDTM can guess set M' in time $O(m{*}n)$ and verify it on time $O(2mn+2n)$. Now a reduction of problem B1 from Binary Knapsack Problem (BKP) will be demonstrated, defined as follows. There is a set of items with weights w_k and costs c_k. Is it possible to fill a knapsack with items of total weight not exceeding W and sum of costs being at least C?

$$\max \sum_{k=1}^{l} z_k c_k$$

$$\text{s.t. } \sum_{k=1}^{l} w_k z_k \leq W,$$

$$x_l \in \{0,1\},\ l = 1,\ldots,k.$$

Given an instance of BKP, following instance of B1 can be constructed. There is only one shop. Prices of products are equal to the weights of knapsack items $p_{i1} = w_k, k = i, i = 1,\ldots,m$ and similarly values are equal to costs $v_i = c_k$. Delivery price $d_1 = 0$. The budget equals knapsack size $P = W$ and the required value of items equals sum of costs $V = C$. This simple reduction is polynomial in size of the problem. BKP has a solution if and only if there exists a solution for constructed instance of the problem B1, and the selected set of items M' will be the solution for BKP as well. This proves that the problem B1 is NP-complete and thus B-ISOP is NP-hard.

3 Special case: maximize only the number of products

Special case of the problem can be considered as follows. The customer is not specifying values of the products, but rather willing to maximize the number of the products he receives. This can be achieved by using $v_i = 1$ for $i = 1,\ldots,m$ in previous formulation of the B-ISOP. For this version of the problem the earlier proof of NP-Hardness with reduction from BKP will not hold, because BKP where the costs of all the items are equal is polynomially solvable. A new proof will be constructed by proving NP-Completeness of the decision problem B2.
Problem B2: Is it possible to buy a set $M'' \subset M$ of products with cardinality m'' at least V'?

Proof B2 is in NP exactly in the same way as B1. Reduction of the problem B2 can be performed from Maximum Coverage (MC) problem [7], defined as: Given a number k and a collection of sets $S = S_1, S_2, \ldots, S_m$ find a subset $S' \subseteq S$ of sets, such that $\left|S'\right| \leq k$ and the number of covered elements $\left|\bigcup_{S_l \in S'} S_l\right|$ being at least g.

With any instance of MC instance of B2 can be constructed. Shops represent the subsets $M_j = S_l$, with prices of all products p_{ij} set in a following way. If element i is part of the set S_l price of according product is equal to zero $p_{il} = 0$. If the set S_l does not contain element i the price of according product p_{il} is a big number. All delivery prices are equal to one $p_j = 1,\ i = 1,\ldots,m,$. The budget represents the given limit of sets $P = k$ and the required number of products represents a required number of covered elements $V' = g$. Values of the products equal to one $v_i = 1$ for $i = 1,\ldots,m$ were assumed previously as distinction of this special case. The reduction can be performed in time $O(m{*}n)$ so it is polynomial again. The constructed instance of the problem B2 solves the MC. This proves that the problem B2 is NP-complete and thus the special case is NP-hard as well.

4 Relation to other problems

The proof provided in previous section shows that the proposed B-ISOP problem is a generalization of the Binary Knapsack Problem. Although there might seem to be some similarities, the problem is not a generalization of the Multiple Choice Knapsack Problem [12]. The difference is that there not only the cost of the item (here value) changes between classes, but also its weight (here price). Further, B-ISOP is a generalization of the Maximum

Coverage problem and some of the MC generalizations. This includes the Weighted Maximum Coverage (*WMC*) [7] where the elements have weights (here perceived value) and the Budgeted Maximum Coverage (*BMC*) [9] where the elements have weights and the sets have costs (here delivery price). All the above versions of MC do not have costs of the elements, which causes an algorithmic difference: with inclusion of a set (here shop) to a solution, all its elements are automatically included. This supposedly renders the approximation algorithms provided for those problems not usable for B-ISOP. One more general version of MC problem named Generalized Maximum Coverage (*GMC*) [6] was also identified, where the value of an element differs from set to set.

Although the motivation for B-ISOP comes from a field of shopping optimization, the BMC and GMC problems were proposed with other motivations. For BMC examples of facilities location problems are proposed, while GMC is described as having important applications in wireless OFDMA scheduling. This sets proposed B-ISOP formulation in wider context and shows that work on this problem might offer results also for areas of operations research other than shopping optimization.

5 Summary and future work

In this paper formulation of new optimization problem was introduced. As the problem can be well placed in context of related problems to the best of authors knowledge such formulation was not described ever before. The main contributions of this paper are a mathematical model of the problem and proofs of NP-hardness of both B-ISOP and its special case with maximization of just the number of products.

Future work on the problem might be twofold. Firstly, it can focus on algorithms for the described problem. This would include at least: an analysis of usability of greedy approximation algorithm provided by [6] for more general problem GMC, a proposal of heuristic (e.g. metaheuristic) algorithms. This would require also an approach giving optimal solutions at least to asses an optimality gap of the other algorithms on smaller instances. The proposed ILP model should allow for use of an existing solver software. Secondly, the model presented as every mathematical model is some simplification of the reality. It could be extended to capture more real world situations, e.g. from on-line shopping or cloud brokering areas. Some ideas to catch up are already proposed in the Internet shopping optimization research like with price-sensitive discounts [3,2] or dual discounting functions [4].

Areas of possible application of the presented problem are not limited to Internet shopping optimization. Similar problems can be already found in cloud brokerage, where user wants to buy the best set of cloud resources. Possibly in not so far future this might also prove useful on markets of energy, where the user wants to buy electricity from different sources, even valuating green energy more than dirty one.

Acknowledgements Jakub Marszalkowski acknowledges support by the FNR (Luxembourg) and NCBiR (Poland), through IShOP project, INTER/POLLUX/13/6466384.

References

1. Adler, M., Gibbons, P.B., Matias, Y.: Scheduling space-sharing for internet advertising. Journal of Scheduling **5**(2), 103–119 (2002)
2. Blazewicz, J., Bouvry, P., Kovalyov, M., Musial, J.: Erratum to: Internet shopping with price-sensitive discounts. 4OR **12**(4), 403–406 (2014)
3. Blazewicz, J., Bouvry, P., Kovalyov, M., Musial, J.: Internet shopping with price sensitive discounts. 4OR **12**(1), 35–48 (2014)
4. Blazewicz, J., Cheriere, N., Dutot, P.F., Musial, J., Trystram, D.: Novel dual discounting functions for the internet shopping optimization problem: new algorithms. Journal of Scheduling pp. 1–11 (2014)
5. BŁAZEWICZ, J., Kovalyov, M.Y., MUSIAŁ, J., WOJCIECHOWSKI, A.: Internet shopping optimization problem. Int. J. Appl. Math. Comput. Sci **20**(2), 385–390 (2010)
6. Cohen, R., Katzir, L.: The generalized maximum coverage problem. Information Processing Letters **108**(1), 15–22 (2008)
7. Hochbaum, D.S.: Approximation algorithms for np-hard problems. SIGACT News **28**(2), 40–52 (1997)
8. Hurst, N., Marriott, K.: Satisficing scrolls: a shortcut to satisfactory layout. In: Proceeding of the eighth ACM symposium on Document engineering, DocEng '08, pp. 131–140. ACM, New York, NY, USA (2008)
9. Khuller, S., Moss, A., Naor, J.S.: The budgeted maximum coverage problem. Information Processing Letters **70**(1), 39–45 (1999)
10. Marszałkowski, J., Drozdowski, M.: Optimization of column width in website layout for advertisement fit. European Journal of Operational Research **226**(3), 592–601 (2013)
11. Marszałkowski, J., Rusiecki, Ł., Drozdowski, M., Narożny, H.: Toward building aesthetic, useful and readable tag clouds for websites. In: Proceedings of the 11th International Conference on e-Business (ICE-B-2014), pp. 230–235 (2014)
12. Sinha, P., Zoltners, A.A.: The multiple-choice knapsack problem. Operations Research **27**(3), 503–515 (1979)

MISTA 2015

Packing-based approaches for a discrete malleable task scheduling problem

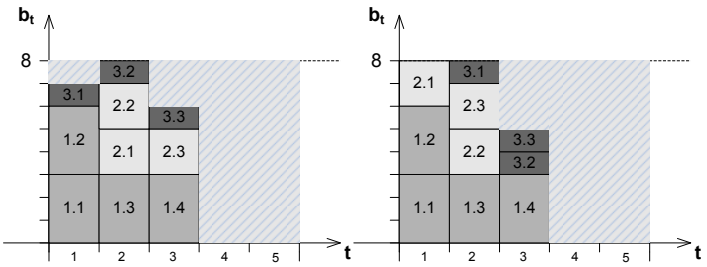
Roland Braune

1 Introduction

The subject of this work is a multi-capacitated scheduling problem inspired by a real-world scenario. Tasks involve processing of a predefined number of materials, where each material requires the same amount of time to be processed. Resource capacities are considered in an aggregate form, inducing a "big-bucket" scheduling model where a resource may process multiple tasks at the same time. More specifically, we are dealing with *discretely divisible*, *renewable* resources. Consequently, the actual resource consumption of a task at a time period t is an integer multiple of the processing time (resource units) required to handle one single material. The resulting *size* of a task may therefore vary over time, being only bounded from above by the respective resource's maximum capacity. However, task preemption is not allowed, meaning that, once started, a task has to be scheduled without interruption until it is finished. This in turn implies that at least one material has to be in process at every time instant of its scheduled time window.

The objective is to minimize a weighted form of resource idle time. It is not sufficient to just keep the resource utilization as high as possible, as commonly achieved by makespan minimization, it is rather the time instant at which idle time occurs that matters. In the corresponding real-world scenario, high resource utilization in earlier time periods is of crucial importance, mainly due to rolling horizon planning concerns. Figure 1 shows the difference between the "standard" and "weighted" idle time perspective. Each rectangle corresponds to one material and materials that belong to the same task are shaded in the same color and share the same task index (the number before the dot). The capacity of the resource is denoted by b_t, where t represents the (discrete) time index. Note that the pure total idle time is equal to 22 in both cases. However, Figure 1b depicts the preferred situation, where idle time is "shifted" to the right. Unlike other idle time or utilization related criteria, such as work continuity or resource leveling (cf., e.g., [7]), the above described objective apparently has not yet been subject to any kind of investigations in the context of multi-capacitated scheduling.

Roland Braune
Department of Business Administration,
Faculty of Business Economics and Statistics, University of Vienna,
1090 Vienna, Austria E-mail: roland.braune@univie.ac.at



(a) Unweighted idle time (b) Idle time depending on the time index

Fig. 1: Resource idle time seen from different points of view.

When looking at the capacity allocation scheme on a single resource only, it is easy to identify relationships to *malleable* task scheduling on a multiprocessor resource [5]. However, the classic notion of this problem assumes task preemptability and a continuous time line (cf., e.g., [2]). To the best of our knowledge, purely discrete variants have not been discussed in the literature so far. The recently proposed *energy scheduling problem* [1] also bears close resemblance to the described setting and can be considered as a generalization.

2 Modeling Approach and Complexity Analysis

A more formal description of the scheduling problem outlined in Section 1 can be given as follows: Let A denote the set of tasks to be processed on a single multiprocessor resource. For each task i, we are given the number of materials mc_i to be processed, and the number of resource units ρ_i required for processing one single material.

Let b_t denote the available capacity of a single multiprocessor resource at time t and let ac_{it} be the resource amount allocated by task j on that resource at time t. Note that for each task i and time index t, $ac_{it} \bmod \rho_i = 0$ has to hold (cf. Section 1). The desired idle time minimization effect, as indicated in Figure 1, can be achieved by weighting idle time with a function $g(t)$, strictly decreasing in t. We chose a straightforward linear scheme, based on the length T of the planning horizon, that is, $g(t) := T - t + 1$. Then the objective function can be written as

$$\sum_{t=1}^{T} (b_t - \sum_{i \in A} ac_{it}) \cdot (T - t + 1) \tag{1}$$

Clearly, each task can be seen as a chain of rigid unit time task "slices" (cf. the notion of evolving tasks [5]), coupled by minimum and maximum time lags with fixed values of 0 and 1, respectively. Each slice of a task i has unit processing time and requires exactly ρ_i resource units.

By adopting the slice-based perspective, it is possible to transform (1) to an expression containing a constant part and a sum of products of slice completion times and required resource amounts ρ_i, which corresponds to the classic weighted completion time objective.

In fact, the overall problem, denoted as $\mathscr{P}$ henceforth, can finally be considered as a multiprocessor scheduling problem with rigid unit-time tasks, chains with minimum and maximum time lags (constant, task independent), weights equal to the task sizes, and weighted completion time objective. In common three-field notation, this problem can be written as $P \mid chains(l); size_i; p_i = 1; w_i = size_i \mid \sum w_i C_i$.

As far as complexity is concerned, the combination of constant time lags and weights equal to the task sizes does not allow for a straightforward reduction from an existing multiprocessor scheduling problem. However, the NP-hardness proof for $P \mid size_i; p_i = 1 \mid \sum C_i$ (cf. [6]) can easily be extended by size-dependent weights. Consequently, problem $P \mid size_i; p_i = 1; w_i = size_i \mid \sum w_i C_i$, briefly denoted as $\mathscr{P}'$, is shown to be NP-hard in the strong sense. Since problem $\mathscr{P}'$ is a relaxation and thus a special case of the original problem $\mathscr{P}$, strong NP-hardness can also be deduced for the latter.

3 Lower Bounds and Exact Solution Methods

Problem $\mathscr{P}'$ ($P \mid size_i; p_i = 1; w_i = size_i \mid \sum w_i C_i$) and clearly also the original version $\mathscr{P}$ with precedence constraints can be mapped to a bin packing problem with linear usage costs (cf., e.g., [3]). This problem involves packing n items, each corresponding to a single (unit-time) task slice, into m bins, one for each time index of the original scheduling problem. For each item $i \in \{1,\ldots,n\}$ and each bin $j \in \{1,\ldots,m\}$, a binary variable x_{ij} indicates whether the item is assigned to the respective bin ($x_{ij} = 1$) or not. The bin capacities C_j are all set equal to the number of available parallel processors (P). We can then formulate the (relaxed) problem as

$$\min \quad \sum_{j=1}^{m} j \cdot \sum_{i=1}^{n} x_{ij} \cdot w_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} \cdot w_i \leq C_j, \quad \forall 1 \leq j \leq m, \tag{2}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall 1 \leq i \leq n, \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad \forall 1 \leq i \leq n, \forall 1 \leq j \leq m, \tag{4}$$

where constraints (2) ensure that bin capacities are not exceeded while constraints (3) require that each item is assigned to exactly one bin. The incorporation of precedence constraints is straightforward and omitted here due to space limitations. Note that due to the absence of fixed costs, the variant that we consider is *not* a generalization of the classic bin packing problem.

At first, we propose various lower bounds for that kind of bin packing problem. The focus is primarily on the relaxed version without precedence constraints. The idea of Martello and Toth's lower bound L_2 [8] is transferred and extended to this problem. Furthermore, we analyze the performance of three different Lagrangean relaxation schemes, based on assignment and capacity constraints, in that specific context. Specifically, we show that the well known Lagrangean relaxation for the RCPSP, initially proposed by Christofides et al. [4], is applicable to the original problem setting with precedence constraints.

The lower bounds are then embedded into a straightforward, item-based branch-and-bound algorithm to solve both problem variants to optimality. Apart from that, we propose extensions to the constraint propagation techniques introduced by Cambazard et al. [3] for linear usage cost bin packing.

Besides randomly generated problems of varying size, our final computational experiments also involve instances extracted from a real-world scenario where the objective at hand is in fact one of the optimization criteria of immediate relevance to the decision maker.

References

1. Artigues, C., Lopez, P., Haït, A.: The energy scheduling problem: Industrial case-study and constraint propagation techniques. International Journal of Production Economics **143**(1), 13–23 (2013)
2. Blazewicz, J., Kovalyov, M., Machowiak, M., Trystram, D., Weglarz, J.: Preemptable malleable task scheduling problem. IEEE Transactions on Computers **55**(4), 486–490 (2006)
3. Cambazard, H., Mehta, D., O'Sullivan, B., Simonis, H.: Bin packing with linear usage costs an application to energy management in data centres. In: C. Schulte (ed.) Principles and Practice of Constraint Programming, *Lecture Notes in Computer Science*, vol. 8124, pp. 47–62. Springer Berlin Heidelberg (2013)
4. Christofides, N., Alvarez-Valdes, R., Tamarit, J.M.: Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research **29**, 262–273 (1987)
5. Drozdowski, M.: Scheduling for Parallel Processing. Computer Communications and Networks. Springer Verlag London (2009)
6. Drozdowski, M., Dell'Olmo, P.: Scheduling multiprocessor tasks for mean flow time criterion. Computers & Operations Research **27**(6), 571–585 (2000)
7. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **207**(1), 1–14 (2010)
8. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons, New York (1990)

MISTA 2015

Multi-level Benders Decomposition for Multi-modal Outpatient Scheduling in Hospitals

Atle Riise · Leonardo Lamorgese · Carlo Mannino

1 Introduction

Most hospitals have laboratories and clinics that perform examinations or surgical procedures on outpatients. We consider the scheduling of patient appointments in such clinics. This scheduling is typically more complex than appointment scheduling in, say, primary care clinics. It often involves the scheduling of a large number of patients over a horizon of several weeks or months. Each activity may require several constrained renewable resources such as a doctor, a room, and some equipment. Some resources may have setup times. The clinic usually has several resources of each type available on each day, and so a suitable combination of resources (a *mode*) must be chosen for each appointment, in addition to the start time. Activity durations depends on the procedure, age, and medical condition of the patient, and often also on the chosen mode.

There is a substantial literature on outpatient appointment scheduling in health care; see e.g. the reviews in [7] and [12]. However, this literature mostly concerns simpler, single resource problems [1], where the goal is an optimal design of time blocks into which future patients can be scheduled. Typical research questions include the optimal choice block start times, the number of patients in each block, the estimation of service durations, the choice of scheduling heuristics to apply, etc. Often, all patients are assumed to require the same service time, or they may be grouped based on expected service time. In contrast, the problems class that we study in this paper typically arise in hospital out-patient clinics that provide more complex services for elective patients, requiring more resources and individual service times, as described above. Indeed, these problems are more similar to inpatient surgery scheduling. In particular, they resemble

Atle Riise
SINTEF ICT, Dept. of applied mathematics, and University of Oslo
E-mail: atle.riise@sintef.no

Leonardo Lamorgese
SINTEF ICT, Dept. of applied mathematics
E-mail: Leonardo.Lamorgese@sintef.no

Carlo Mannino
SINTEF ICT, Dept. of applied mathematics, and University of Oslo
E-mail: carlo.mannino@sintef.no

surgery admission planning, with the added challenge of assigning not only a day, but also a start time, to each patient. There are several recent surveys of the surgery scheduling literature, including the comprehensive overview in [6], which was updated in 2013 [9], as well as [20,11,16]. Using the taxonomy in [9], we see that very few papers consider the scheduling of outpatients, while also including choices of both start time and resources (operating rooms) [17,21,5,4].

The approach presented in this paper is inspired by the Logic-Based Benders' Decomposition (LBBD) scheme introduced in [13,15]. Over the past 15 years, different authors have successfully applied LBBD to a variety of hard discrete optimization problems such as general planning and scheduling [18,14,2], location-allocation [10], transport scheduling [8,19,23], bin packing [22], sports scheduling [24] and more. Most authors resort to a two stage approach like in LBBD or the classic Benders', with some exceptions (e.g.[3]).

In this paper, we present a three-level LBBD algorithm for the multi-model appointment scheduling problems. We are not aware of any previous work that apply logic based Bender's decomposition to this kind of scheduling problems. A case study based on real world data from a Norwegian hospital provides empirical evidence for the efficiency of the proposed algorithm, and for its superior performance compared to a corresponding two-level LBBD algorithm. We offer some insight into why this is so, and finally indicate some directions for future research.

2 A three-level Logic Based Benders Decomposition algorithm

The multi-modal appointment scheduling problems that we consider lend themselves naturally to LBBD. Firstly, the most important objective typically is waiting time, relative to priority based due dates.

This objective depends only on the day at which each patient is scheduled, and it is therefore naturally to formulate our master problem based on the allocation of patients to days (Problem I). Often, as in our case study, the remaining sub problem is a feasibility problem. Furthermore, the problem has a clear periodic (daily) structure, where the schedule on each day is independent of the schedules for other days. The original problem, which may be very large due to the long planning horizon, can therefore be decomposed into a set of much smaller problems, one for each day. These daily sub problems constitute the slave problems in our first decomposition. However, we show that, while effective, this decomposition is not in itself enough to tackle real life problem instances, which may contain several hundred patients. We therefore apply a second decomposition of each daily sub problem into a master *mode problem*

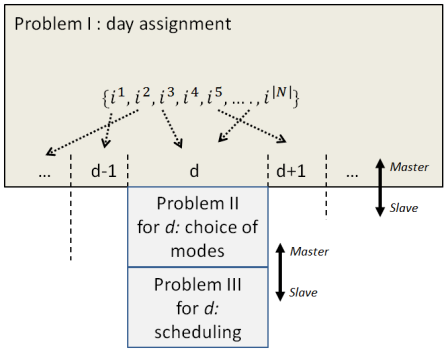


Fig. 1 The 3-level logic based bender's decomposition for a set of appointments, N. The d's enumerate the days.

(Problem II), where the mode for each appointment is chosen, and a slave *scheduling problem* (Problem III). The scheduling problem determines if there exists a feasible start time for each activity, given the mode choices as they are fixed in the master solution. See Fig. 1 for an illustration of the relationship between these problems.

3 Case study

We present computational results from a case study at the gastroenterology laboratory at the University Hospital of North Norway, a fairly typical Norwegian hospital outpatient clinic. Based on real resource and patient data from both daily and monthly (bulk) planning situations, we show that the three-level LBBD algorithm is vastly superior to the corresponding two-level version. Indeed, while the two-level algorithm cannot solve the monthly scheduling problems within the one hour time-out, the proposed three-level LBBD algorithm solves them to optimality within 5 minutes. We discuss how the various aspects of our decomposition contributes to this increased performance.

Acknowledgements This work has been supported by the Research Council of Norway, under grant no. 219335/O30.

References

1. Batun, S., Begen, M.A.: Optimization in healthcare delivery modeling: Methods and applications, pp. 75–119. Springer (2013)
2. Beck, J.C.: Checking-up on branch-and-check, pp. 84–98. Springer (2010)
3. Benini, L., Lombardi, M., Mantovani, M., Milano, M., Ruggiero, M.: Multi-stage benders decomposition for optimizing multicore architectures. In: M. Perron, M. Trick (eds.) Fifth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Oprimization Problems, pp. 36–50 (2008)
4. Cardoen, B., Demeulemeester, E., Beliën, J.: Optimizing a multiple objective surgical case sequencing problem. International Journal of Production Economics **119**(2), 354–366 (2009). 0925-5273
5. Cardoen, B., Demeulemeester, E., Beliën, J.: Sequencing surgical cases in a day-care environment: An exact branch-and-price approach. Computers & Operations Research **36**(9), 2660–2669 (2009). 0305-0548
6. Cardoen, B., Demeulemeester, E., Beliën, J.: Operating room planning and scheduling: A literature review. European Journal of Operational Research **201**(3), 921–932 (2010). 0377-2217
7. Cayirli, T., Veral, E.: Outpatient scheduling in health care: A review of literature. Production and Operations Management **12**(4), 519–549 (2003)
8. Correia, I., Captivo, M.: Bounds for the single source modular capacitated plant location problem. Computers & Operations Research **33**, 2991–3003 (2006)
9. Demeulemeester, E., Beliën, J., Cardoen, B., Samudra, M.: Operating Room Planning and Scheduling, *International Series in Operations Research & Management Science*, vol. 184, book section 5, pp. 121–152. Springer New York (2013)
10. Fazel-Zarandi, M., Beck, J.: Using logic-based benders decomposition to solve the capacity and distance constrained plant location problem. INFORMS Journal on Computing **24**, 399–415 (2012)
11. Guerriero, F., Guido, R.: Operational research in the management of the operating theatre: a survey. Health Care Management Science **14**(1), 89–114 (2011)
12. Gupta, D., Denton, B.: Appointment scheduling in health care: Challenges and opportunities. IIE Transactions **40**(9), 800–819 (2008)
13. Hooker, J.N.: Logic-based Methods for Optimization. Wiley (2000)
14. Hooker, J.N.: Planning and scheduling by logic-based benders decomposition. Operations Research **55**(3), 588–602 (2007)

15. Hooker, J.N., Ottosson, G.: Logic-based benders decomposition. Mathematical Programming **96**(1), 33–60 (2003)
16. Hulshof, P.J.H., Kortbeek, N., Boucherie, R.J., Hans, E.W., Bakker, P.J.M.: Taxonomic classification of planning decisions in health care: a structured review of the state of the art in or/ms. HS **1**(2), 129–175 (2012)
17. Huschka, T.R.: Bi-criteria evaluation of an outpatient procedure center via simulation. In: Simulation Conference, 2007 Winter, pp. 1510–1518 (2007)
18. Jain, V., Grossmann, I.: Algorithms for hybrid milp/cp models for a class of optimization problems. INFORMS Journal on Computing **13**, 258–276 (2001)
19. Lamorgese, L., Mannino, C.: Optimal train dispatching by benders'-like decomposition. Transportation Science **to appear** (2015)
20. May, J.H., Spangler, W.E., Strum, D.P., Vargas, L.G.: The surgical scheduling problem: Current research and future opportunities. Production and Operations Management **20**(3), 392–405 (2011)
21. Pham, D.N., Klinkert, A.: Surgical case scheduling as a generalized job shop scheduling problem. European Journal of Operational Research **185**(3), 1011–1025 (2008). 0377-2217
22. Pisinger, D., Sigurd, M.: Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. INFORMS Journal on Computing **19**(1), 36–51 (2009)
23. Raidl, G., Baumhauer, T., b., H.: Speeding up logic-based benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In: Springer (ed.) Hybrid Metaheuristics (2014)
24. Rasmussen, R., Trick, M.: A benders approach for the constrained minimum break problem. European Journal of Operational Research **177**(1), 198–213 (2007)

MISTA 2015

Variable Neighborhood Search for a Rich Production Planning Problem

Michael Schilde · Karl Schneeberger · Karl F. Doerner

1 Introduction and problem description

We present a solution approach based on variable neighborhood search (VNS, [4]) for a real-world inspired rich production planning problem for dairy products. Our method can be used by practitioners to perform the detailed planning for daily production as well as to determine the consequences of possible future developments (e.g., increasing demand for existing products, introduction of new products, or installing additional equipment). Especially the latter is a task that currently is often based on rough estimates and gut instincts. Our approach covers the three main problem components (lot-sizing, sequencing, and scheduling) and all aggregate levels simultaneously and thus provides practically feasible solutions.

The underlying problem covers the production of several final products on a set of heterogeneous filling machines. Each final product consists of one or more components that need to be made available in filling tanks before the filling machines can start filling them into bins (e.g., yoghurt cups, PET bottles, etc.). These components are produced by mixing blended milk with different ingredients in a mixer, followed by a certain soaking time in a mixing tank. After this, the product is denoted as base mass and needs to be heated in a heater, fermented in a fermentation tank, chilled in a cooler and stored in a filling tank for the filling machine. If the next aggregate is not ready, the product can remain inside each of the tanks for a limited amount of time. Some products do not require to be processed on every aggregate level and can thus skip one or more of them.

Each aggregate is connected to a set of ingoing and outgoing product pipes that are used to transfer products between aggregates. Each pipe can be connected to several aggregates on both ends, but a transfer between two aggregates always blocks the pipe

M. Schilde (✉) · K. Schneeberger · K.F. Doerner
University of Vienna, Department of Business Administration,
Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria
E-mail: michael.schilde@univie.ac.at

K.F. Doerner
University of Vienna,
Christian-Doppler-Laboratory for Efficient Intermodal Transport Operations,
Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

for all other aggregates during this time. Each used pipe and aggregate also causes a certain amount of product loss when being cleaned after processing a product. A cleaning in place (CIP) pipe and a sterilization in place (SIP) pipe are attached to each aggregate. Each CIP/SIP-pipe can be connected to more than one aggregate, but only one of these aggregates can be serviced (cleaned or sterilized) at a time. Furthermore, a cleaning interval can be given for each aggregate to specify how much time may pass between two cleaning procedures, without causing an additional intermittent cleaning. The requirement for and the duration of a setup, a cleaning, and a sterilization are sequence-dependent. There are product-dependent limits on the time between finishing fermentation and starting to cool the product, and on the maximum time any product may spend inside a tank. Each aggregate is assigned a certain processing speed for each product it can process and each tank has a minimum and a maximum capacity.

Gellert et al. [2] studied a related problem with respect to the cleaning intervals as well as the cleaning and sterilization of aggregates, but they only considered the filling lines, assuming that they are the most limiting factor for the entire production system. Their approach will lead to practically infeasible solutions if the aggregate levels before the filling lines can not provide the required base mass in time due to resource conflicts. The main novelty of our approach is the fact that we consider the complete, realistically sized, multi-stage production system including all resource blocking constraints, whereas the mentioned authors focus solely on the filling lines. For an overview of the recent literature covering perishability issues in production see Amorim et al. [1]. Lütke-Entrup et al. [3] presented a mathematical model formulation covering some of the aspects included in our problem setting, but by far not all of them. In another work, we extended their position based model to match our problem setting and developed a fix&optimize based solution approach for very small test cases [5]. This work is based on the same problem definition.

2 Solution approach

Our solution approach consists of an adaptive VNS and a new heuristic to generate an initial feasible solution for the problem. The initial solution is generated by first grouping the final products by base mass (as several products may require the same base mass). Within each group, the products are sorted using a 3-opt procedure, such that the required time for setup and CIP/SIP within the group is minimized. Then, the product groups are assigned to filling machines based on a load balancing constraint and sequenced in a way that again the total time needed for setup and CIP/SIP between the groups on each machine is minimized. After that, the production lots for the remaining aggregate levels, transfer pipes, and CIP/SIP pipes are created for each of the product groups. On each aggregate level, the algorithm again aims for similar loads on all aggregates. After creating the production lots on all aggregate levels, a tailored sequencing and scheduling algorithm is used to determine a feasible timing for this solution.

The VNS-based improvement method aims to eliminate potential resource conflicts on the different aggregate levels and pipes. Therefore the method uses a move neighborhood operator and a swap neighborhood operator to change the assignment of production lots to aggregates or pipes. The move operator assigns one or more production lots to a different aggregate. Hereby the lots to be moved are selected based on how much the remaining lots on this aggregate would benefit from removing the

corresponding lot (e.g., because the lot prevents another lot from starting earlier or blocks a required CIP or SIP pipe). The aggregates to which the lots are moved are selected pseudo-randomly based on the same criterion (aggregates with lower impact are preferred). The swap operator works just as the move operator, but exchanges two or more lots between two or more aggregates based on the described criterion. The number of lots to be moved/swapped defines the neighborhood size of the operators. The selection of the neighborhood operators and their sizes is performed on a random basis. Initially, all operators and sizes have the same probability of being selected. After each iteration, the probabilities are adapted depending on the obtained solution (i.e., if an operator/size combination finds an improving solution, the chance of being selected is increased, otherwise it is decreased).

3 Test instances and solutions

Our solution approach was evaluated using real-world test instances from a large European dairy producing company. These instances each contain demand information for 101 different final products during one week as well as information about 41 aggregates, 178 product pipes, 5 CIP pipes, and 6 SIP pipes. The results obtained using our solution method show, that we are able to find solutions of similar quality as the ones used in practice within 15 minutes of calculation time even without applying the improvement method. Compared to the fix&optimize based solution approach we developed for this problem setting, our metaheuristic solution approach is competitive for very small instances (larger instances cannot be solved using the fix&optimize method).

By applying the metaheuristic, the results can be improved even further (depending on the calculation time granted to the search algorithm). The solutions used in practice are manually created and thus creating them can take several working days. Furthermore the risk of planning errors (e.g., missing production volume or overlooked conflicts on the product pipes, CIP or SIP pipes) can be drastically reduced by using an automated method. We are thus confident that our method will be a viable alternative when used as a planning support tool in practice.

Acknowledgements Financial support from the Austrian Research Promotion Agency (FFG, Bridge) under Grant #838560 is gratefully acknowledged.

References

1. P. Amorim, C.H. Antunes, and B. Almada-Lobo. Multi-objective lot-sizing and scheduling dealing with perishability issues. *Industrial & Engineering Chemistry Research*, 50:3371–3381, 2011.
2. T. Gellert, W. Höhn, and R.H. Möhring. Sequencing and scheduling for filling lines in dairy production. *Optimization Letters*, 5:491–504, 2011.
3. M. Lütke-Entrup, H.-O. Günther, P. van Beek, M. Grunow, and T. Seiler. Mixed integer linear programming approaches to shelf life integrated planning and scheduling in yogurt production. *International Journal of Production Research*, 43:5071–5100, 2005.
4. N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
5. K. Schneeberger, M. Schilde, and K. F. Doerner. Solving a rich position-based model for dairy products with a fix&optimize based solution approach. Technical Report UNIVIE-PLIS-2015-01, University of Vienna, Department of Business Administration, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria, January 2015.

MISTA 2015

Scheduling of jobs in the continuous casting stage of steel production

Oliver Herr · Asvin Goel

Extended Abstract

In this contribution we study a problem motivated by a practical problem arising in the continuous casting stage of steel production. In the practical problem, a continuous caster is fed with ladles of liquid steel. Each ladle contains a certain steel grade and has orders allocated to it that determine a due date. Whenever two ladles of different steel grade are processed consecutively, a setup process is required. However, no setup is required when the steel grade is not changed. The liquid steel is produced from hot iron supplied by the blast furnace with a constant rate. The sequence of ladles, including setups between ladles of different steel grades, is not allowed to consume more hot metal then supplied by the blast furnace.

The problem studied in this contribution is a variant of a single machine family scheduling problem with the goal of minimizing total tardiness, see e.g. Gupta and Chantaravarapan (2008) and Schaller (2007). In the problem, each job has a given processing time, a due date, and belongs to a given family. The machine can only process one job at a time and each job must be processed without preemption. A setup task has to be conducted between jobs belonging to different families and during this setup the machine cannot process any job. Furthermore, each jobs requires a certain amount of a common resource that is supplied at a constant rate. Each job consumes a given amount of the resource at a constant rate. At any time, the cumulative consumption must not exceed the cumulative supply. Therefore, jobs may have to wait due to an insufficient availability of the resource.

As illustrated in Figure 1, the fundamental difference of this problem to the case without resource constraints is, that it may be necessary that the machine is idle because the required resource for the next job is net yet available, whereas in the case without resource constraints the machine is only idle for the time of the setups that may be required. In the case without resource constraints the optimal duration of any

Oliver Herr
Jacobs University, Bremen, Germany
E-mail: o.herr@jacobs-university.de

Asvin Goel
Kühne Logistics University, Hamburg, Germany
E-mail: asvin.goel@the-klu.org

subsequence of jobs is always the sum of all processing times and the required setups. With resource constraints, however, the optimal duration may be larger because the machine has to wait for the resource required.

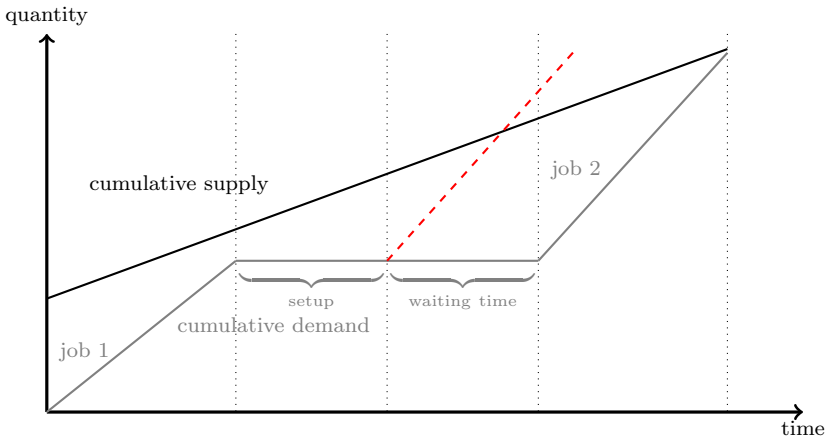


Fig. 1 Additional waiting time may be required because of the resource constraint.

Due to operational requirements in continuous casting, waiting times must only be scheduled when a setup is conducted. Therefore, a complete subsequence of jobs with the same steel grade may be delayed because of the resource constraint or an additional setup has to be conducted, possibly of larger duration than the required waiting time.

In this contribution we present a formulation of the problem as a mixed integer program. As solving the problem using a general purpose MIP-solver is in general too time consuming for problem instances with larger numbers of jobs, we present a heuristic approach based on different operators modifying the sequence of jobs. The approach iteratively selects a neighbourhood operator to obtain a new sequence of jobs. In order to reduce the number of decisions to be made when applying the operators we focus our solution representation on the sequence of jobs without explicitly considering where in the current solution a setup is conducted. In order to evaluate a sequence of jobs, however, the exact timing of setups and their duration has to be determined. Central to our approach is a labelling method for determining where setups have to be conducted - and for how long - in order to minimize total tardiness for any sequence of jobs.

Our approach is evaluated on artificially generated instances having similar characteristics as the real-life problem that motivated this research. Computational experiments are conducted comparing the performance of our approach with a state-of-the-art commercial mixed integer programming solver. These experiments demonstrate that the proposed solution approach finds optimal or near optimal solutions for most of the small sized instances with less than 15 jobs. For instances with 15 jobs or more, the commercial mixed integer programming solver was not able to optimally solve the instances within a run time limit of one hour. For these larger instances our approach outperforms the commercial solver both in terms of solution quality of the best found solution as well as the time required to find this solution. The run time required by our

heuristic only grows moderately so that good solutions can be obtained within only a few minutes for instances with a problem size of practical relevance.

References

J. N. D. Gupta and S. Chantaravarapan. Single machine group scheduling with family setups to minimize total tardiness. *International Journal of Production Research*, 46(6):1707–1722, 2008. ISSN 0020-7543. doi: 10.1080/00207540601009976. URL `http://www.tandfonline.com/doi/abs/10.1080/00207540601009976`.

J. E. Schaller. Scheduling on a single machine with family setups to minimize total tardiness. *International Journal of Production Economics*, 105(2):329–344, Feb. 2007.

MISTA 2015

The polynomial solvability of a bicriteria linear combination on parallel identical machines with release dates

Hari Balasubramanian • John Fowler• Ahmet Keha

1 Introduction

Certain deterministic NP-hard scheduling problems in the parallel machine environment have been proved to polynomially solvable when the processing times are assumed to be equal. $P|r_j|\sum w_jC_j$ and $P|r_j|\sum T_j$ are NP hard, but $P|r_j,p_j=p|\sum w_jC_j$ and $P|r_j,p_j=p|\sum T_j$are polynomially solvable by a linear programming based algorithm proposed in Brucker and Kravchenko [3]. Further, the equal processing time assumption is well suited to many multi-resource non-preemptive scheduling problems in practice. For example, even though appointment durations tend to be stochastic, during the booking process office based medical practices offer equal length appointment to patients who call in at different times of the day.

In a recent review focused on equal processing time problems, Kravchenko and Werner [1] noted that problems involving both deadlines and weights, $P|r_j,p_j=p,D_j|\sum w_jC_j$ and $P|r_j,p_j=p|\sum w_jT_j$, still remain open. In a paper that reports new results for preemptive scheduling problems in the same environment, Prot et al. [2] report that $P|rj,pj=p,prmp|\sum w_jT_j$ also remains open.

We tackle the bicriteria linear combination $\alpha\sum w_jC_j+(1-\alpha)\sum T_j$, $0\le\alpha\le1$, in the $P|r_j,p_j=p|-$ non-preemptive setting. Note that this is an objective function that includes both weighted completion time as well as total tardiness. We demonstrate that the LP-based algorithm Brucker and Kravchenko [3] can be used to solve the bicriteria linear combination, as long as:

$$\frac{1}{1+w_{min}}\le\alpha\le1, \text{ where } w_{min}=\min\{w_i-w_j|J_i,J_j,w_i>w_j,d_i>d_j\} \quad (1)$$

Hari Balasubramanian
University of Massachusetts, Amherst
E-mail: hbalasubraman@ecs.umass.edu

John Fowler
Arizona State University
E-mail: john.fowler@asu.edu

Ahmet Keha
Arizona State University (now at Exxon Mobil Research and Engineering Company)
E-mail: ahmet.keha@asu.edu

In other words, w_{min} is the smallest difference in weights for any pair of jobs for which a clear dominance cannot be established. A dominance between a pair of jobs cannot be established when the weight of one job is greater than the other but its due date is later. Without loss of generality we can assume that the weights are integral, so the lowest value that w_{min} can take is 1. Therefore the transformation will always work for $\alpha \geq 1/2$. Therefore, when the weights for both total weighted completion time and total tardiness are equal, the bicriteria linear combination is polynomially solvable. If $w_{min} = 2$, then bicriteria linear combination is solvable for $\alpha \geq 1/3$. Thus, the larger the value of w_{min}, the wider is the range of α values for which polynomial solvability of the bicriteria linear combination in the $P|r_j, p_j = p| -$ setting is possible. In light of the fact that $P|r_j, p_j = p, D_j | \sum w_j C_j$ and $P|r_j, p_j = p | \sum w_j T_j$ are open problems, our bicriteria result provides an intriguing middle ground.

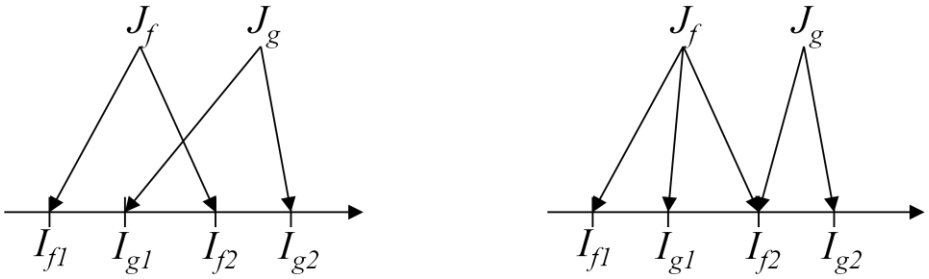
What drives this special condition on w_{min}? To explain this, we provide an outline of the linear programming based algorithm proposed in Brucker and Kravchenko [3].

2 LP Based Algorithm

Brucker and Kravchenko [3] use a linear programming algorithm to solve $P|r_j, p_j = p| \sum w_j C_j$ and $P|r_j, p_j = p| \sum T_j$. We use the same algorithm to demonstrate the polynomial solvability for the bicriteria linear combination involving total weighted completion time and total tardiness. The LP formulation can be described as follows. Since processing times are equal, the number of time intervals in which the jobs can be scheduled can be pre-calculated. For a non-decreasing objective function, the jobs can either start on their own release date or after other jobs have completed. This means that $r_j + kp, \forall j = 1 \dots n$, where k is an integer, describes all possible start times. Therefore, there are O(n^2) possible start times and as many intervals. The main decision variable in the linear program, $x_{j,i}$ assigns a portion of job *j*'s total processing time *p* to interval *i*. Constraints ensure that: (1) the sum total of job *j* assigned across all intervals adds to *p*; (2) no more than *m* jobs are scheduled simultaneously; and (3) no portion of a job's processing time is assigned before the job's release date. The objective function to be minimized is the bicriteria linear combination. Note that the optimal solution to this linear program may not be feasible, since the processing time of a job may be assigned to multiple intervals. In Brucker and Kravchenko (2008) the LP optimal solution x^* for $P|r_j, p_j = p| \sum w_j C_j$ is *transformed* to a new solution x^{**} which, in turn, is used to assign jobs feasibly and optimally to *n* "marked" intervals (these marked intervals, determined in a separate algorithm, identify where the jobs will be scheduled).

Similarly, to ensure that $\alpha \sum w_j C_j + (1 - \alpha) \sum T_j$, $0 \leq \alpha \leq 1$ can be feasibly and optimally solved, the transformation from x^* to x^{**} as described in Brucker and Kravchenko (2008) has to be achieved. In particular, the original LP solution x^* may have a pair of "mixed jobs" as shown in Figure 1 below. In other words, there may exist two jobs say J_f and J_g and four intervals I_{f1}, $I_{g1,}$ I_{f2}, and I_{g2} (which are located in that order in time) such that the variables $x_{f,f1}$, $x_{f,f2,}$ $x_{g,g1}$are all non-zero. Such a pair of jobs are called mixed jobs. To eliminate mixed jobs in the schedule, an amount min ($x_{g,g1,}$ $x_{f,f2}$) is transferred/exchanged between the intervals g_1 and f_2. Figure 1 shows the transformation when $x_{g,g1} < x_{f,f2}$.

Figure 1. Jobs are mixed (left) and after transformation, no longer mixed (right). An amount min ($x_{g,g1}$, $x_{f,f2}$) is exchanged between the intervals I_{g1} and I_{f2}. In this case $x_{g,g1} = \min(x_{g,g1}, x_{f,f2})$



An important condition is that these transformations to eliminate mixed jobs should leave the objective function unchanged.

We show that that for minimizing $\alpha \sum w_j C_j + (1 - \alpha) \sum T_j$, $0 \leq \alpha \leq 1$ the objective function remains unchanged under this transformation for all $\frac{1}{1+w_{min}} \leq \alpha \leq 1$. This condition becomes relevant when neither of the jobs, J_f and J_g, dominates the other: i.e. when the weight of one job is greater than the other, but the due date is later. The rest of the steps to prove feasibility and optimality are identical to Brucker and Kravchenko (2008). For a full and more comprehensive derivation and implications which are not discussed in this abstract, please see the working paper we have posted online [4].

References

1. Kravchenko, S. A., & Werner, F. (2011). Parallel machine problems with equal processing times: a survey. *Journal of Scheduling*, *14*(5), 435-444.
2. Prot, D., Bellenguez-Morineau, O., & Lahlou, C. (2013). New complexity results for parallel identical machine scheduling problems with preemption, release dates and regular criteria. *European Journal of Operational Research*, *231*(2), 282-287.
3. Brucker, Peter, and Svetlana A. Kravchenko. "Scheduling jobs with equal processing times and time windows on identical parallel machines." *Journal of Scheduling* 11.4 (2008): 229-237.
4. Balasubramanian, H., Fowler, J., and Keha, A.. "The polynomial solvability of selected bicriteria scheduling problems on parallel machines with equal length jobs and release dates" Working paper: http://people.umass.edu/hbalasub/BalasubramanianetalJOS.pdf

MISTA 2015

LAHC applied to The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Janniele A. Soares · Haroldo G. Santos · Davi D. Baltar · Túlio A. M. Toffolo

1 Introduction

The Project Scheduling Problem (PSP) consists in to schedule jobs over time in such a way that precedence relations are satisfied and resource consumption limits are respected [5]. In this paper a comprehensive variation of the PSP is considered: the Multi-Mode Resource Constrained Multi-Project Scheduling Problem (MMRCMPSP). In this problem jobs can be processed in different modes, with varying execution speeds and consuming different amounts of resources. The objective function also considers project delays.

As a generalization of the PSP, the MMRCMPSP is NP-Hard and can be used to model many problems in several areas, such as project management in information technology companies, scheduling instructions to processor architecture, civil engineering, ingot production scheduling, among others. Even the generation of an initial feasible solution for the MMRCMPSP is also NP-Hard, a feasible selection of modes most be selected, which corresponds to solving the multi-dimensional knapsack problem.

The model of MMRCMPSP [6] has a set P projects, where each $p\ inP$ consists of a set $J_p = 1, ..., |J_p|$ jobs. Each project p has a start time, where jobs can be initiated. The beginning and the end of a project are delimited by fictitious jobs.

Furthermore, the MMRCMPSP comprises a set of constraints, related with the precedence between jobs J and the consumption of resources. These resources, can be renewables R, in way local or global (shared with another projects), and non-renewable K, may to deplete throughout each project $p \in P$. Each job $j \in J$ can be performed in a certain mode $m \in M$, it determines the time taken for the execution d_{jm} and the amount of renewable and non-renewable resources consumed, respectively v_{rjm} and u_{kjm}. This consumption can not exceed the amount available of renewable q_r and non-renewable o_k resources. All relations of precedence $Pred$ and $Pred_j$ must be guaranteed.

The objective function adopted in this paper, refers to the default in MISTA 2013 Challenge [1], having two components: the TPD, main goal, which is the sum of the

J. A. Soares[1] · H. G. Santos[2] · D. D. Baltar[3] · T. A. M. Toffolo[4] · [1,3] Computing and Information Systems Department, Federal University of Ouro Preto, Brazil · [2,4] Computing Department, Federal University of Ouro Preto, Brazil · E-mail: janniele@decsi.ufop.br, haroldo@iceb.ufop.br, davibaltarx@gmail.com,tulio@toffolo.com.br

delays of the projects in relation to the duration of the critical path and the TMS, secondary objective, which is the difference between the maximum completion time of a project and the beginning minimum time of a project.

In this work, we propose and evaluate computationally a solver based in the Late Acceptance Hill-Climbing (LAHC) [4] metaheuristic. The techniques implemented consider as input specific instances of multi-projects available on the MISTA2013 Challenge site, these instances are composed of specific projects from PSPLib.

2 Solution Approach

Our approach, differently from most approaches used in the MISTA 2013 competition, always navigates in the search space of feasible solutions. The generation of different feasible mode sets is accomplished by the solution of a series of multi-dimensional knapsack problems and the use of indirect solution representations. After building an initial solution pool, several LAHC threads start local search around those solutions.

To increase diversity, we first build a pool of different mode sets which satisfy the consumption of non-renewable resources. Since the selection of processing modes also determines the duration of jobs, a greedy strategy to prioritize the selection of fast processing modes is employed. One must observe, however, that it does not guarantees a smaller TPD, because renewable resources constraints can delay the starting time of jobs.

The binary program to built this initial set of modes considers: J jobs with respective processing times p_{jm} and N non renewable resources. Each job has a set M_j of possible modes and the non-renewable resource n consumption of job j in mode m is denoted as r_{jmn}. Thus, the binary program is solved to select which mode m job j will be allocated, considering its respective decision variables x_{jm} and resource availability q_n for each non renewable resource. Which corresponds to the NP-Hard problem of the 0-1 multidimensional knapsack problem.

The local search procedures which will be described later operate over an indirect solution representation: a solution is stored in a (Γ, Π) ordered pair of vectors where $\Gamma_j \in M_j$ indicates the selected mode for job j and $\Pi_j \in \{1, \ldots, |J|\}$ indicates the desired position for job j in the sequence of allocations. Γ always respects non-renewable resources consumption. As most of the processing time of this algorithm is spent in the local search phase, the ability to quickly decode a (Γ, Π) pair is a fundamental aspect in the performance of the algorithm.

We implemented all optimizations proposed in [2], such as prefix detection and early exploration of resource insufficiency. Differently from [2] we do not guarantee a valid topological sort in Π. This speeds up the generation of valid movements, since some validations may be disabled but the cost saved is moved to the decoding phase. To transform the sorting in Π into a valid topological sorting, at each new allocation one has to check the available job with highest priority (smallest desired position), which yields an $O(n^2)$ algorithm to decode Π. Fortunately we devised a simple heap to speed up this to $O(n \log n)$. Initially, all jobs are inserted into the heap with priority $|S_j^-| \times |J| + \Pi_j$.

The complete neighborhood structure $\mathcal{N}(s)$ is composed by fourteen types of movements: Change One Mode (COMS), Change Two Modes (CTMS), Change Three Modes (CTRMS), Change Four Modes (CFMS), Invert Subsequence (INVS), Shift Jobs (SJS), Swap Jobs (SWJS), Compact Project (CP), Shift Project (SPS), Swap

Two Projects (SWPS), Mutation One Extreme (MOE), Move Projects (MP), Swap Jobs FILS (SJF), Insert Jobs FILS (IJF).

The Late Acceptance Hill-Climbing (LAHC) is a new metaheuristic proposed by [3], which consists of an adaptation of the classic Hill-Climbing method. To accept or not a new candidate, its cost is compared with some cost previous belonging the list. It stores a list c of size l with values of cost, this size is the number of the previous iterations i. The list c is initialized to the cost of the initial solution s received as a parameter. At each iteration i, a candidate solution s' is generated and its cost is compared to a virtual position in the c. The solution s' is accepted if their cost is smaller than the contents of the virtual position on c, or is smaller than the cost of current solution s. If accepted, the solution s will be updated, if even better than the best solution found so far $s*$, it will also be updated. Subsequently the contents of the virtual position of c will be updated with the new cost of s

To prevent stagnation and increase diversification, we store how many movements were performed for each job j. Whenever number 1000 of non-improvement iterations is reached, diversification is activated and a series of diversification movements is performed. The selection of these movements uses an adapted objective function which incentives the selection of movements involving jobs which were not commonly selected in previous iterations.

3 Computational Experiments

All algorithms were coded in C++ and the binary programming models were solved by CPLEX 12.6. The code was compiled with GCC 4.7.1 using flag -O3. All tests ran on a computer with an Intel(R) Core(TM) i7-4960X CPU @ 3.60GHz processor and 32 Gb of RAM, running OpenSUSE Linux 13.2. The developed method ran in parallel using 4 threads.

Table 1 shows the best results found by the proposed approach, as well the average and standard deviation, after 10 runs within 300 seconds of runtime, with 4 threads with l size of 500. The last column of the table represents the gap obtained by dividing the cost of the best known solution B for the cost of the best solution obtained C by approach. Instances were the obtained results were better or equal than the best results known in the literature are emphasized. It is important to emphasize that the results presented by the technical report [2] were obtained in 2500 different executions for each instance, while the results presented by MISTA2013 Challenge were obtained with only 10 different executions, the same as the results of the present work.

4 Conclusions

In this work we presented the application of LAHC metaheuristic for MMRCMPSP. Integer programming is used to build an initial feasible solution. The LAHC algorithm was enhanced to perform informed diversification using long term memory. Efficient algorithms to decode indirect solution representations were also implemented. Our solver was able to improve two best known solutions for instances used in the MISTA 2013 Challenge and provided solutions very close to the best known ones for the remaining instances.

Table 1 Best and average results after 10 runs of the algorithm sided with [1] and [2]

Inst.	Best		Avg.		Std.Dev.		Mista		Report [2]		B/C
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	**1**	**23**	**1.00**	**23.00**	**0**	**0**	**1**	**23**	**1**	**23**	**1.00**
A-2	**2**	**41**	**2.27**	**41.13**	**0**	**0**	**2**	**41**	**2**	**41**	**1.00**
A-3	**0**	**50**	**0**	**50**	**0**	**0**	**0**	**50**	**0**	**50**	**1.00**
A-4	**65**	**42**	**67.50**	**43.10**	**0.81**	**1.58**	**65**	**42**	**65**	**42**	**1.00**
A-5	157	105	169.00	109.40	6.03	2.06	153	105	150	103	0.96
A-6	145	96	159.00	99.60	9.26	3.38	147	96	133	99	0.92
A-7	608	195	630.90	206.80	11.63	5.74	596	196	590	190	0.97
A-8	286	152	308.90	155.20	10.99	1.99	302	155	272	148	0.95
A-9	207	126	220.50	129.50	6.25	2.91	223	119	197	122	0.95
A-10	896	312	943.10	320.70	18.45	3.66	969	314	836	303	0.93
B-1	352	125	366.70	130.80	6.96	2.68	349	127	294	118	0.84
B-2	444	167	460.90	168.70	8.09	2.41	434	160	431	158	0.97
B-3	557	213	579.30	213.10	10.87	0.83	545	210	526	200	0.94
B-4	1276	281	1355.70	291.10	31.37	4.64	1274	289	1252	275	0.98
B-5	845	250	871.90	257.40	16.06	4.20	820	254	807	245	0.96
B-6	911	226	938.20	229.60	12.22	3.17	912	227	905	225	0.99
B-7	807	233	863.00	240.50	23.49	4.15	792	228	782	225	0.97
B-8	**2974**	**541**	**3081.20**	**545.60**	**54.66**	**4.36**	**3176**	**533**	**3048**	**523**	**1.02**
B-9	4630	851	4730.30	852.30	52.19	7.24	4192	746	4062	738	0.88
B-10	**3050**	**448**	**3106.90**	**450.20**	**34.59**	**5.33**	**3249**	**456**	**3140**	**436**	**1.03**
X-1	393	143	426.50	148.50	13.38	4.10	392	142	386	137	0.98
X-2	367	166	394.20	171.40	12.16	2.94	349	163	345	158	0.94
X-3	325	191	342.30	193.50	10.59	2.50	324	192	310	187	0.95
X-4	915	208	958.10	211.90	21.16	3.39	955	213	907	201	0.99
X-5	1786	377	1862.80	382.50	36.02	5.20	1768	374	1727	362	0.97
X-6	700	234	745.00	239.50	23.86	3.56	719	232	690	226	0.99
X-7	861	236	902.20	236.80	15.65	5.90	861	237	831	220	0.97
X-8	1218	286	1277.50	292.40	25.46	2.97	1233	283	1201	279	0.99
X-9	3475	706	3520.90	697.60	27.89	7.79	3268	643	3155	632	0.91
X-10	1652	399	1695.90	396.90	18.29	4.81	1600	381	1573	383	0.95

Acknowledgements The authors thank CNPq and FAPEMIG for supporting this research.

References

1. Wauters, T. Kinable, J. Smet, P. Vancroonenburg, W. Berghe, G.V. and Verstichel, J. : Mista 2013 challenge (2013). URL `http://allserv.kahosl.be/mista2013challenge/`. Access date: 2 jan. 2015
2. Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E., Parkes, A.J.: Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, technical report. Tech. rep., University of Nottingham, School of Computer Science (2013)
3. Burke, E., Bykov, Y.: A late acceptance strategy in hill-climbing for exam timetabling problems. PATAT 08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
4. Burke, E., Bykov, Y.: The Late Acceptance Hill-Climbing Heuristic. Tech. rep., University of Nottingham, School of Computer Science (2012)
5. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research **174**(1), 23 – 37 (2006)
6. Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Berghe, G.V., Verstichel, J.: The multi-mode resource-constrained multi-project scheduling problem. Journal of Scheduling (2014)

MISTA 2015

Automated Design of the Developmental Approach for Solving the Examination Timetabling Problem

Nelishia Pillay

Keywords: hyper-heuristics, automated design, timetabling

1 Introduction

The design of optimization techniques for timetabling and scheduling problems can be time consuming requiring several man hours. The research presented forms part of a larger initiative investigating the automated design of timetabling and scheduling systems using hyper-heuristics [3]. As an initial attempt, the developmental approach, which was created to solve the examination timetabling problem [2], is automatically designed using an evolutionary algorithm hyper-heuristic (EAHDA). The following sections present the developmental approach, the EAHDA, results and discussion and the conclusion and future work respectively.

2 Developmental Approach (DA)

The developmental approach was initially applied to the Toronto benchmark set and later placed as one of the finalists for the examination timetabling track for the second international timetabling competition (ITC2007). This approach takes an analogy from cell biology to develop an organism, representing a solution to the problem, by mimicking the processes of cell creation, cell interaction and cell migration. Figure 1 depicts the manually designed algorithm for the developmental approach for timetabling [6]. In the context of examination timetabling each organism represents a timetable and each cell a period in the timetable. The algorithm begins by sorting the examinations according saturation degree [9] with the highest cost used to break ties. The highest cost heuristic is the sum of the soft constraint cost over the feasible periods the examination can be scheduled in. An examination with a higher highest cost is given priority. If the organism has at least two cells, cell migration is firstly performed. Cell migration essentially changes the position of a cell in the organism. As in nature, this process is stimulus driven and in this context the stimulus is a reduction in soft constraint cost. The developmental approach employs two migration operators, *migrate* and *migraten*. The *migrate* operator changes the position of two randomly selected cells while *migraten* changes the position of an existing cell to a position not as yet allocated to a cell. The algorithm then performs cell interaction which moves an examination to a new cell while still maintaining feasibility and resulting in a reduction in soft constraint cost. All three operators are attempted for 50 iterations and if an improvement in soft constraint cost is not found no change is made. In later work chaos was incorporated into the developmental approach to better emulate the process of development in nature [7]. Noise operators were used for this purpose. Each noise operator introduces randomness by generating random numbers for i iterations. The

developmental approach is used to create *p* organisms and the fittest organism is returned as a solution. A value of 250 has generally been used for *p*.

```
Sort examinations to be scheduled in list
Create a cell and allocate a randomly selected position
Allocate the first examination in list to the cell
while (there are still examinations to schedule)
begin
 if(there are at least two cells in the organism)
  perform cell migration
 for 1 to no_of_cells
  begin
   if (a feasible cell exists for the examination)
    Allocate the exam to the cell with minimum soft constraint cost
   else if (maximum number of cells has not been reached)
    Perform cell creation
   else
    Allocate the examination to a randomly selected cell
  endfor
  Perform cell interaction
  Update low-level heuristics for the examinations in list
  Resort list
endwhile
end
```

Figure 1. Developmental approach algorithm

3 Evolutionary Algorithm Hyper-Heuristic for Developmental Approach Design (EAHDA)

This section describes the evolutionary algorithm hyper-heuristic employed to design the development approach algorithm. The evolutionary algorithm implements the generational algorithm. Each chromosome represents the developmental approach algorithm and is a string comprised of numbers representing the different biological processes: 0 (cell allocation), 1 (*migrate*), 2 (*migraten*), 3 (cell interaction), 4 (invoke noise for *i* iterations, 10 <= i <= 50). The length of each chromosome is a parameter value (*max_len)* and the best value to use is problem dependent. If *max_len* is set to a value of 5, an example of a chromosome is *10223*. This chromosome represents an algorithm that performs *migrate* followed by cell allocation, *migraten* twice and finally cell interaction. If the organism does not have at least two cells, *migrate*, *migraten* and cell interaction has no effect. The fitness of a chromosome is calculated by using it solve the problem, in this case constructing an examination timetable. Suppose that *03210* is a chromosome. Then the processes of cell allocation, cell interaction, *migraten*, *migrate* and cell interaction are performed iteratively in this order until all the examinations are scheduled. The fitness is the hard constraint cost plus one multiplied by the soft constraint cost. Tournament selection is used to select parents to create offspring for the next generation. Mutation and crossover are used to create offspring for each generation. Mutation replaces the number at the mutation point with a randomly selected number in the range 0 to 4. Crossover crosses over two parents at the crossover point to produce two offspring.

4 Results and Discussion

The EAHDA was applied to the Toronto benchmark set [9]. The hard constraint for the benchmark set is that no student must be scheduled to sit more than one examination at the same time, i.e. there must be no clashes. The soft constraint cost is a measure of the spread of examinations. The parameter values for the evolutionary algorithm are depicted in Table 1.

These parameter values were determined by performing trial runs using three problem instances, namely, *hec-s-92*, *sta-f-83* and *ute-s-92*. The algorithms were implemented in Java and simulations were run using a multi-core architecture. Two systems were used for this. There first is an HP workstation with 512 gigabytes of RAM, running Windows 8 and the second a Linux Sun cluster made available by the national center of high performance computing. Fifty cores were used for all experiments.

Table 1: Evolutionary algorithm parameter values

Population size	300
Maximum number of generations	50
Tournament size	4
Initial maximum length (*max_len*)	15
Mutation application rate	90%
Crossover application rate	10%

Each algorithm evolved by the EAHDA was used to create just one organism and not a population of *p* organisms as in previous applications of the developmental approach. Due to the stochastic nature of evolutionary algorithms and the developmental approach ten runs, each with a different random number generator seed, were performed for each problem instance. The algorithms evolved by the EAHDA were able to produce feasible solutions for all ten runs. The minimum soft constraint cost, average soft constraint cost and average runtimes over the ten runs is tabulated in Table 2 for each problem instance. The table also lists the evolved algorithm producing the minimum soft constraint cost over the ten runs for each problem instance. The runtimes of EAHDA vary from 16 minutes for smaller problem instances to 13 hours for the larger data sets. However, if you consider the number of man hours usually needed to develop the algorithm, try different operators, and the time taken to perform simulations to test the different algorithm configurations, this is not much.

Table 2: Toronto benchmarks

Problem	**Minimum Soft Constraint Cost**	**Average Soft Constraint Cost**	**Average Runtime**	**Developmental Approach Algorithm Producing Minimum Cost**
car-f-92	3.98	4.01	11 hrs	113023311324433
car-s-91	4	4.48	9 hrs	244332123021211
ear-f-83	34	34.3	2 hrs	313443312230113
hec-s-92	10.64	10.71	16 mins	103211144113342
kfu-s-93	13.53	13.63	8 hrs	240312142421423
lse-f-91	10.29	10.35	6 hrs 20 mins	123441032111313
rye-s-93	8.85	8.95	13 hrs	221140421114423
sta-f-83	157.12	157.16	33 mins	211410230330134
tre-s-92	8.01	8.10	3 hrs	031413131011132
uta-s-92	3.17	3.19	12 hrs	201332123331343
ute-s-92	25.64	25.85	48 mins	141001221421143
yor-f-83	36.86	37.15	1 hr 12 mins	320411213323114

Table 3 compares the performance of algorithms evolved by EAHDA to the standard developmental approach algorithm [6] and the later variation using noise operators. The best soft constraint cost obtained is highlighted in bold. It is evident from Table 3 that the DA algorithms evolved by EAHDA have performed better than the previous versions of EAHDA, despite creating only one organism instead of a population of organisms as in the case of the standard DA and DANO. These algorithms have produced better results than the DA for all problem instances and DANO for eleven of the twelve problem instances.

Table 3: Performance comparison with previous DA versions

Problem	EAHDA	DA [6]	DANO [7]
car-f-92	**3.98**	4.1	3.99
car-s-91	**4**	4.8	4.8
ear-f-83	**34**	34.97	34.14
hec-s-92	**10.64**	10.99	10.66
kfu-s-93	**13.53**	13.89	13.55
lse-f-91	**10.29**	10.6	10.29
rye-s-93	**8.85**	9.08	8.94
sta-f-83	**157.12**	157.22	157.12
tre-s-92	**8.01**	8.26	8.05
uta-s-92	**3.17**	3.24	3.22
ute-s-92	**25.64**	26.23	25.82
yor-f-83	36.86	38.38	**36.52**

The performance of the evolved algorithms were also empirically compared to approaches producing the best results for the Toronto benchmark set, recent approaches applied to this benchmark set, and recent hyper-heuristics used to solve the set of problems. Table 4 lists the soft constraint costs of the EAHDA and the following approaches: (1) sequential construction method with backtracking applied [5]; (2) honey bee optimization [10]; (3) adaptive decomposition and ordering approach [1]; selection perturbative hyper-heuristic [4]; (5) selection constructive hyper-heuristic in [8]; (6) selection constructive hyper-heuristic [11]; (7) generation perturbative hyper-heuristic in [12].

Table 4: Performance comparison with state-of-the-art and recent approaches

Problem	EAHDA	(1)	(2)	(3)	(4)	(5)	(6)	(7)
car-f-92	3.98	6.0	3.9	4.74	4.31	4.22	4.7	4
car-s-91	4	6.6	4.79	5.17	5.37	4.95	5.14	4.62
ear-f-83	34	29.3	34.69	40.91	35.79	35.95	37.86	34.71
hec-s-92	10.64	9.2	10.66	12.26	11.19	11.27	11.90	10.68
kfu-s-93	13.53	13.8	13	15.85	14.51	14.12	15.3	13
lse-f-91	10.29	9.6	10	12.58	10.92	10.76	12.33	10.11
rye-s-93	8.85	6.8	10.97	10.11	-	9.23	10.71	10.79
sta-f-83	157.12	158.2	157.04	158.12	157.18	157.69	160.12	158.02
tre-s-92	8.01	9.4	7.87	9.3	8.49	8.43	8.23	7.9
uta-s-92	3.17	3.5	3.10	3.65	3.44	3.33	3.88	3.12
ute-s-92	25.64	24.4	25.94	27.71	26.7	26.95	32.67	26
yor-f-83	36.86	36.2	36.15	43.98	39.47	39.63	40.53	36.2
Average	26.34	26.08	26.51	28.70	28.85	27.21	28.61	26.60

These methods are described in section 2. As can be seen from Table 4 the algorithms evolved by EAHDA have performed comparatively to the state-of-the-art, recent and hyper-heuristics approaches, used to solve the Toronto benchmark set of problems. In order to get an idea of how well these approaches have performed over the set of problems the average cost over the problem instances has been calculated for each approach. Here again the algorithms evolved by EAHDA have performed well obtaining the second best average.

5 Conclusion and Future Work

The research presented in this paper examines the use of hyper-heuristics to design the developmental approach for solving the examination timetabling problem. The soft constraint costs of the timetables produced were found to be comparable to that produced by state-of-the-art, recent and hyper-heuristic approaches used to solve the Toronto benchmark set of problems. The evolved algorithms were found to be disposable, with a different algorithm producing the best result on each run for a problem instance. Given the success that EAHDA

has had with the Toronto benchmark set, the EAHDA is currently being tested on benchmark set of the second international timetabling competition (ITC 2007). Future work will also examine including additional operators such as cell depletion and cell division.

Acknowledgements This work is based on the research supported in part by the National Research Foundation of South Africa for the Grant CSUR13091742778. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF does not accept any liability in this regard.

References

1. Abdul-Rahman, S., Burke, E. K., Bargiela, A., McCollum, B. and Ozcan, E., A Constructive Approach to Examination Timetabling Based on Adaptive Decomposition and Ordering, Annals of Operations Research, Vol. 218, 3-21 (2014).
2. Banzhaf, W., Pillay, N., Why Complex Systems Engineering Needs Biological Development, Complexity, Vol. 13, No. 2, 12-21 (2007).
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R., Hyper-heuristics: A Survey of the State of the Art. Journal of the Operational Research Society, 64, 1695-1724 (2013).
4. Burke, E.K., Qu, R., Soghier, A. Adaptive Selection of Heuristics for Improving Exam Timetables, Annals of Operations Research, Vol. 218, 129-145 (2014).
5. Caramia, M., Del Olomo, P. and Italiano, G. F., Novel Local-Search-Based Approaches to University Examination Timetabling, INFORMS Journal of Computing, Vol. 20, No. 1, 86-99 (2008).
6. Pillay, N., The Revised Developmental Approach for the Uncapacitated Examination Timetabling Problem, in Proceedings of SAICSIT 2009, 187-192 (2009).
7. Pillay, N., A Study of Noise Operators in the Developmental Approach for the Examination Timetabling Problem, in Proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS 2011), November 2011, Vol. 3, 534-538, Guangzhou, China, IEEE Press (2011).
8. Pillay, N., Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem, Journal of the Operational Research Society, 63, 47-58 (2012).
9. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y., A Survey of Search Methodologies and Automated System Development for Examination Timetabling, Journal of Scheduling, 12(1), 55–89 (2009).
10. Sabar, N.R., Ayob, M., Kendall, G. and Qu, R., A Honey-Bee Optimization Algorithm for Educational Timetabling Problems, European Journal of Operational Research, Vol. 216, Issue 3, 533-543 (2012).
11. Sabar, N. R., Ayob, M., Qu, R. and Kendall, G., A Graph Colouring Hyper-Heuristic for Examination Timetabling Problems, Applied Intelligence, Vol. 37(1), 1-11 (2012).
12. Sabar, N. R. Ayob, M, Kendall, G. and Qu, R., Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems, IEEE Transactions on Evolutionary Computation, 17(6), 840-861(2013).

MISTA 2015

Scheduling Time Variant Jobs on a Time Variant Resource

Geir Horn

In remembrance of Professor Robert Dobinson (1943-2004)

1 Introduction

Classical scheduling originated in manufacturing disciplines and considers the problem of *assigning* a set of n jobs onto m machines. The job j is assumed to have a known processing time on machine i. It should be noted that classical scheduling only implicitly considers the resources provided by the machine, *i.e.* the capacity of the machine is reflected in the time it takes to complete the job on that machine. The situation considered here is different in that the machine provides time variant resources, and the scheduling problem is to start the time variant jobs according to the resource availability on the machine. The jobs are *continuous* and once a job has started it will have to run to completion,*i.e.* the problem is a nonpremptive single-machine scheduling problem. In contrast to classical scheduling problems, the machine may start two or more jobs with overlapping execution periods if the machine's resources allow this. Initially, we will consider the *relaxed* form of the problem where it is possible to acquire additional resources for the machine to run the jobs, albeit at significantly higher cost than in its standard configuration. This will guarantee that the problem has a solution, and transform the problem to find the schedule that minimises the cost of the additional resources. The core results are presented in this extended abstract with proofs to be found in the full paper.

The problem at hand is motivated from energy related problems and the introduction of renewable energy sources whose energy production depend on weather conditions. The scheduling of electrical appliances is normally referred to as *demand side management* (DSM) [2] within the context of the next generation "smart" energy systems [3]. Consider for example the electricity produced by a photovoltaic panel whose electricity production depends on various factors like the solar radiation at the particular time of the year and the amount of clouds. We would like to use this resource to run electrical appliances with time variate energy consumption. If a cloud passes after after an appliance has started, we have to compensate for the reduced energy production by buying electricity from the grid. The goal is to minimise this external transaction. However, despite this practical motivation, we believe that the problem is generally applicable as a new class of scheduling problems.

Dr. Geir Horn
University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail: Geir.Horn@mn.uio.no

2 Basic Concepts

The schedule is fundamentally a vector $\mathbf{s} \in \mathbb{R}^n$ that assigns a *start time* s_j to each of the n jobs. Each job has a *duration* Δ_j. Since the jobs are nonpremptive, the continuous interval for which a job j consumes resources is referred to as the job's *consumption interval* and it is denoted $\mathbb{I}_j = [s_j, s_j + \Delta_j] \subset \mathbb{R}$.

One or more other jobs can be started within a job's consumption interval. Thus, in the extreme, all consumption intervals will overlap and there will be only one consumption *period*. In the general case, however, there will be several disjoint consumption periods where one or more jobs are executed in each period. Formally, a given schedule $\mathbf{s}$ will partition the job index set, $\mathbb{J} = \{1, \ldots, n\}$, into disjoint index sets $\mathbb{P}_{k|\mathbf{s}}$ such that

$$\bigcup_k \mathbb{P}_{k|\mathbf{s}} = \mathbb{J}$$

Each of these consumption periods correspond to a consumption interval $\mathbb{I}_{k|\mathbf{s}}$ being the union of the consumption intervals of the jobs in the consumption period.

Each job considered here is a *continuous* function whose value represent the amount of resources consumed by the job at a given time, *i.e.* $J_j^0(t)$ for $t \in [0, \Delta_j]$. With no loss of generality, we assume that the resource consumption is cumulative, and the amount of resources consumed between two time stamps $0 \leq t_1 \leq t_2 \leq \Delta_i$ will be $J_j^0(t_2) - J_j^0(t_1) \geq 0$. It should be noted in passing that this is in particular useful for applications when the job consumes energy, as many energy meters report only the cumulative energy consumption.

Assigning a start time s_j to a job will have the effect of shifting its resource consumption in time. There will obviously not be any resources consumed before the job starts, and no further resources consumed when the job finishes. Hence, the job's cumulative resource consumption given the schedule $\mathbf{s}$ is

$$J_j\left(t|\mathbf{s}\right) = \begin{cases} 0 & t < s_j \\ J_j^0\left(t - s_j\right) & s_j \leq t \leq s_j + \Delta_j \\ J_j^0\left(\Delta_j\right) & s_j + \Delta_j < t \end{cases} \tag{1}$$

Based on this, it is straight forward to define the total resource consumption in a consumption period as the sum of the resource demands of the jobs scheduled for that interval. Hence,

$$J_{\mathbb{I}_{k|\mathbf{s}}}(t) = \sum_{j \in \mathbb{P}_{k|\mathbf{s}}} J_j\left(t|\mathbf{s}\right) \tag{2}$$

In order to schedule the jobs according to the available resources, *i.e.* assigning a start time to each job, it is necessary to have some knowledge of the resource provisioning. The resource availability is therefore also assumed to be known as a continuous and cumulative function, $R(t)$. Again, this implies that the function is convex with the property that $R(t_1) \leq R(t_2)$ if $t_1 \leq t_2$.

3 The Scheduling Problem

The forgoing discussion indicates that the problem is fundamentally to find a schedule such that the total resource demands of the jobs in a consumption interval as given by (2) remains below the total resources available over the same consumption interval. It is sufficient to

look at one consumption interval at the time since a job can only belong to one consumption period for any given schedule.

It is therefore natural to assume that a feasible schedule $\mathbf{s}$ satisfies

$$J_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R(t) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right] \leq 0 \quad \text{for all } t \tag{3}$$

The last term in the bracket reflects the fact that the resource function $R(t)$ is cumulative, and only the resources available over the consumption interval can be used to execute the jobs scheduled for this interval. It is therefore necessary to subtract the cumulative amount of resources made available up to the start of the consumption interval, implying that the expression in the bracket is zero at the beginning of the consumption interval.

The assumption (3) is an absolute requirement if it is not possible to change the provisioning of resources. It is a *goal* in situations where additional resources can be obtained, albeit possibly at a high cost. One such example is the aforementioned energy scheduling where the resources may represent the energy produced by renewable sources like a wind mill, but where it is still possible, albeit undesirable, to buy electricity from the grid.

One could therefore face situations where assumption (3) momentarily does not hold. In these cases it would be desirable for the total resource consumption over the entire interval not to exceed the available resources over the interval. In other words, one would integrate assumption (3) over the whole consumption interval:

$$\int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} \left[J_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R(t) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right]\right] \mathrm{d}t \leq 0 \tag{4}$$

The scheduling problem is therefore a non-linear mathematical programming problem [1] aiming at minimising the resource consumption:

$$\min_{\mathbf{s}} \sum_{k} \int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} \left[J_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R(t) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right]\right] \mathrm{d}t \tag{5}$$

subject to any constraints on the starting time for each job.

It is important to note that despite the fact that the value of the integral (4) only depends on the consumption interval for which it is evaluated, the actual schedule $\mathbf{s}$ is common to all consumption intervals, and the schedule $\mathbf{s}$ defines the partitioning of the job set. The problem is therefore not separable, *i.e.* one cannot interchange the sum and the minimisation to solve the problem as a sum of smaller minimisation problems. Even though the optimisation problem cannot be separated, the functional form (5) can be significantly simplified rendering the problem tractable for numerical solution.

4 Simplifications

Lemma 1 (Job consumption and resources) *The resource consumption of the jobs scheduled in a consumption interval can be considered independent from the resources available over the same interval.*

Lemma 2 (Independent job consumptions) *The different jobs in a consumption interval can be considered independently.*

These results allows a simplification of the minimisation problem to be solved to find the best schedule, with the surprising result that the objective function is independent from the jobs' temporal resource consumption. Only the total resource consumption of each job matters, and the temporal availability of resources.

Theorem 1 (Objective function) *The minimisation problem (5) is independent of the jobs' temporal consumption of resources, and the non-linear mathematical programme can be written as*

$$\min_{\mathbf{s}} \left[\sum_{k} \sum_{j \in \mathbb{P}_{k|\mathbf{s}}} J_j^0\left(\Delta_j\right) \left[\max \mathbb{I}_{k|\mathbf{s}} - \left(s_j + \Delta_j\right)\right] + \sum_{k} \left(R\left(\min \mathbb{I}_{k|\mathbf{s}}\right) \left[\max \mathbb{I}_{k|\mathbf{s}} - \min \mathbb{I}_{k|\mathbf{s}}\right] - \int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} R(t)\, \mathrm{d}t \right) \right] \tag{6}$$

and solved subject to any constraints on the start times.

5 Conclusion

In many situations there is a need to schedule time variant jobs on a time variant resource. This topic has received little attention in the scheduling communities. It has been shown in this paper that the problem is independent of the temporal resource consumption by the jobs, and that the schedule found my non-linear mathematical programming will depend only on the temporal resource availability.

Acknowledgements

This paper is dedicated to late Prof. Robert Dobinson from CERN in gratitude for our friendship and lengthy discussions on how to approach the problem of matching distributions of stochastic variables.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant number 608806 CoSSMic.

References

1. David G. Luenberger, Yinyu Ye: Linear and Nonlinear Programming, 3rd edn. Springer (2008)
2. Linas Gelazanskas, Kelum A. A. Gamage: Demand side management in smart grid: A review and proposals for future direction. Sustainable Cities and Society **11**, 22–30 (2014). DOI 10.1016/j.scs.2013.11.001
3. Xi Fang, Satyajayant Misra, Guoliang Xue, Dejun Yang: Smart grid — the new and improved power grid: A survey. IEEE Communications Surveys Tutorials **14**(4), 944–980 (2012). DOI 10.1109/SURV.2011.101911.00087

MISTA 2015

Time-based Decomposition Strategies for the Traveling Umpire Problem

Túlio A. M. Toffolo · Tony Wauters · Greet Vanden Berghe

1 Introduction

The Traveling Umpire Problem (TUP) is an optimization problem introduced by Trick et al. (2012) that considers assigning n umpires (or referees) to games in a double round robin tournament. The tournament schedule is given beforehand, and consists of $4n - 2$ rounds in which the $2n$ teams play twice against each other; once in their home venue and once away. The objective is to minimize the total travel distance of the n umpires. In order to ensure fairness in the schedule, five hard constraints are imposed:

a) every game in the tournament must be officiated by one umpire;
b) every umpire must work in every round;
c) every umpire must visit the home of every team at least once;
d) every umpire must not visit the same venue more than once in any q_1 consecutive rounds;
e) every umpire must not officiate a game with the same team more than once in any q_2 consecutive rounds (this constraint is similar to the previous one, but also takes the 'away team' into consideration);

The values for q_1 and q_2 range from 0 to n and 0 to $\lfloor \frac{n}{2} \rfloor$, respectively.

In this abstract, we present a brief overview of two time-based decomposition algorithms to the TUP. We consider a simple decomposition scheme within two strategies: (*i*) embedded in a parallel branch-and-bound framework and (*ii*) as part of a constructive heuristic. The first strategy, coupled with a matching propagation technique, resulted in stronger lower bounds than any previous ones. This enabled optimally solving all instances with up to 14 teams and, in a few occasions, instances with 16 teams. The second strategy aims at larger instances, containing 18 to 32 teams. New best solutions were obtained during preliminary experiments.

Túlio A. M. Toffolo[a,b] · Tony Wauters[a] · Greet Vanden Berghe[a]
Emails: {tulio.toffolo, tony.wauters, greet.vandenberghe}@cs.kuleuven.be
[a] KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC - Belgium
[b] Federal University of Ouro Preto, Department of Computing - Brazil

2 Parallel branch-and-bound approach

The first approach to the TUP consists of a parallel decomposition-based branch-and-bound algorithm. Starting from the first or from the last round, the algorithm assigns games to umpires, one at a time and round after round, until all games are assigned. Whenever multiple games can be assigned to one umpire in one round, the assignment incurring the smallest increase in the travel distance is chosen. In case of ties, the lexicographic order of games is considered. When no valid assignment can be found for an umpire in a certain round, the procedure backtracks and selects the game with the second smallest travel distance. If the resulting solution is feasible, its total distance is used as an upper bound. Both top-bottom (from the first to the last round) and bottom-up (from the last to the first round) strategies are executed in parallel.

In order to speed up the branch-and-bound, the assignments of the first or last rounds are fixed beforehand (Yildiz, 2008) and some additional pruning rules are taken into account. A simple local search procedure (Wauters et al., 2014) is also applied to the solutions obtained by the branch-and-bound.

In order to provide strong lower bounds, a time-based decomposition algorithm is considered. The problem is initially decomposed into $4n+2$ subproblems, such that each subproblem deals with exactly one round. These initial subproblems consist of finding the trip that each umpire will make to officiate a game in each round. Subproblems can be easily solved with an assignment algorithm, and their solutions provide an initial lower bound. To strengthen the bounds, the decomposition is changed by iteratively incrementing the size of the subproblems. Subproblems with multiple rounds are harder to solve and are tackled by recursively applying the same branch-and-bound, taking advantage of the previously calculated lower bounds.

The subproblems are independent and can be solved in parallel. This enables approaching instances with 18 teams, despite the exponential time complexity of the algorithm.

3 Constructive heuristic approach

The branch-and-bound approach cannot handle instances with more than 18 teams. We introduce a time-based decomposition approach within a constructive algorithm. Initially, the problem is decomposed into m subproblems considering the time horizon (rounds). The value of m is a critical parameter of the algorithm, and varies according to the size of the instance. The m subproblems are solved sequentially, each taking into account the solution of the previous subproblems. If a feasible solution is obtained after solving the m subproblems, local search is applied. Otherwise, if infeasibility is detected while solving one of the subproblems, the procedure backtracks and a different solution of the previous subproblem is selected.

The main challenge in this approach is to avoid backtracking. Different objective functions for the subproblems are considered in order to minimize the backtracking fraction of the algorithm. The main idea is to penalize solutions that are likely to cause infeasibilities. This allows early detection of infeasilibites and therefore considerably speeds up the algorithm.

4 Experiments and results

The algorithms were coded in Java 8 and experiments were executed on an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz computer with 16 processors running Ubuntu Linux 12.04 LTS. The benchmark instances from Michael Trick's website[1] were considered.

The parallel branch-and-bound algorithm was shown to be very powerful. Optimal solutions for all the instances with 14 teams and for some instances with 16 teams were quickly obtained. Nevertheless, the algorithm did not perform well when dealing with larger instances.

The time-based constructive heuristic, on the other hand, obtained good results on larger instances. During preliminary experiments, improving solutions over the best known were produced in less than 3 hours of runtime (Table 1). Complete test results will be presented at the conference.

Table 1: Preliminary results obtained with the heuristic

Instance	q_1, q_2	Best known	Constructive Heuristic	
			Objective	Total Time
umps26	5, 5	354134	**352011**	1457.60 s
umps28	5, 5	398101	**396478**	3255.30 s
umps30	5, 5	450919	**449824**	3159.20 s
umps32	5, 5	502890	**501108**	7851.10 s

Acknowledgements Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be).

References

de Oliveira L, de Souza CC, Yunes T (2014) Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. European Journal of Operational Research 236(2):592 – 600

Trick MA, Yildiz H, Yunes T (2012) Scheduling major league baseball umpires and the traveling umpire problem. Interfaces 42(3):232 – 244

Wauters T, Van Malderen S, Vanden Berghe G (2014) Decomposition and local search based methods for the traveling umpire problem. European Journal of Operational Research 238(3):886 – 898

Yildiz H (2008) Methodologies and applications for scheduling, routing & related problems. PhD thesis, Carnegie Mellon University

[1] http://mat.gsia.cmu.edu/TUP/

MISTA 2015

Heuristics and Algorithms for Data Center Optimization

Per-Olov Östberg · Barry McCollum

1 Introduction

Infrastructure-as-a-Service (IaaS) cloud computing is characterized by automated provisioning of compute resources through self-service APIs. For leveraging of economy of scale effects, cloud compute resources are typically aggregated in large-scale data centers highly optimized for cost and energy efficiency. However, while cloud computing has revitalized the data center paradigm and found substantial commercial success, the average resource utilization of data center resources remains comparatively low [1] and much recent work has been directed towards cloud data center infrastructure optimization.

Due to the wide applicability of cloud-based resource provisioning models, Cloud applications span a very wide range of software applications including, e.g., monolithic legacy applications, scientific simulation and data processing applications, distributed tiered applications, and cloud native applications. Due to this heterogeneity, data center infrastructure optimization is a complex task that requires modeling of cloud applications and their behavior as well as consideration of multiple factors such as application composition, deployment configuration, and workload behavior patterns. The approach to data center infrastructure optimization taken here emphasizes the task of controlling the deployment (placement, scheduling, and horizontal elasticity of applications), as well as capacity as resource assignments (vertical elasticity) of cloud applications and components so that a) infrastructure resources maintain acceptable levels of resource utilization while b) meeting application performance and quality of service (QoS) requirements.

In particular, we address a current need for improvement of the predictability and resilience of cloud data center management tools. As the field of cloud computing transitions into deployment of more mission critical systems such as power and telecommunications infrastructure in heterogeneous data center environments, greater emphasis

Per-Olov Östberg
Dept. of Computing Science, Umeå University, Sweden
E-mail: p-o@cs.umu.se

Barry McCollum
School of EEECS, Queens University of Belfast, United Kingdom
E-mail: b.mccollum@qub.ac.uk

must be placed on application Quality of Service (QoS), stability and predictability of platforms, and development of more advanced control and optimization of infrastructures. Towards this end, we define application and component models designed to capture the behavior or cloud applications in data centers, and investigate design and formulation of heuristics-based optimization techniques for cloud data center optimization.

In this paper we perform an analysis of the different types of data center resource management mechanisms in current use, investigate how optimization and scheduling techniques can be used to improve the effiency of these, and propose a set of heuristics-based optimization techniques for cloud data center resource management optimization. The main contributions of the paper are identification and linking of opportunities for infrastructure optimization, definition of heuristics functions that can be used as cost and evaluation functions in cloud infrastructure optimization, and formulation of a set of algorithms that make up the foundation of the optimization engine in the CACTOS toolkit [2].

2 Cloud Resource Management

In this work we take the perspective of an IaaS cloud provider, and consider the joint problems of cost, energy, and operations optimization for cloud data centers. We assume that the data center infrastructure incorporates some form of virtualization, where the virtualization provides a level of indirection between applications (including application components) and physical resources, and model the optimization actions we can take on this indirection.

To simultaneously capture application behavior and the impact application component load places on infrastructure resources, we here combine two types of models that interlinked model the main interactions of applications and resources in cloud data centers: application models and component models. To give perspective on why this type and level of modeling is used, we here give a brief introduction to the type of infrastructure optimization that is targeted in this work, as well as an overview of each type of model, before going into detail about the heuristics.

2.1 Terminology

In this work we use the following topological definitions: A cloud *application* is a distributed software system where one or more application *components* (software services or subsystems) are deployed in a cloud data center. Components are typically (but not necessarily) deployed in *virtual hosts* using some form of virtualization technology, e.g., hardware supported virtualization (virtual machines), process groups, or software containers, which in turn are mapped onto physical data center (hardware) *resources* using some kind of *deployment constraints*. Deployment constraints constitute rules for the scheduling and placement of virtual hosts on physical resources, and can include, e.g., affinity or anti-affinity constraints that regulate whether or not components can be co-hosted on the same physical resource, or constraints specifying limitations on the amount of hardware resources that can be assigned to components.

2.2 The Role of Virtualization

Many current cloud offerings are based on virtualization techniques such as virtual machines (VMs), process containers, or platform-specific virtualization techniques such as Microsoft .Net Common Language Runtimes or Java Virtual Machines. While all these by definition provide a degree of separation between the code of an application and the physical execution host, we note that fundamental differences in capabilities and technical performance may impact cloud resource management strategies. For example, VMs such as KVM or Xen are designed to provide full operating system isolation and are often capable of being migrated in runtime (so called live migration). Process containers such as Docker are on the other hand designed to be more lightweight and faster to instantiate using image differentiation techniques.

The field of virtualization technology is very fast evolving and as virtualization lies at the core of cloud data centers, development of new technolgies can have a disruptive effect on established trade-offs in data center performance. For example, development of subsecond VM cloning has made VMs a more attractive type of technology in data processing systems for malleable applications. To keep our work indepent of technology-based trade-offs, we here consider virtualization mainly as a source of a level of indirection between applications and resources, and aim to construct parameterizable methods where heuristics can be updated over time to reflect changes brought on by technology developments.

For simplicity in terminology, we here use the term *virtual host* to denote a virtualization technology providing a level of indirection between an application and the physical host the application is executing on, regardless of what type of virtualization technology is used to realize this indirection.

2.3 Application Model

Cloud applications are typically made up by distributed systems that are deployed in data centers as sets of components (or subsystems) hosted in virtual hosts. Commonly, these systems are deployed using deployment tools that describe the makeup and interrelationships between components, often modeled as component graphs in cloud deployment descriptors. To capture the structural relationship between cloud application components, we define a cloud application model that defines communication links between components. As illustrated in Figure 1, this model describes component relationships in a graph format, and aims to capture how load propagates between different components in an application.

2.4 Component Model

In addition to modelling application structure, the quantiative aspects of load propagation must also be taken into account in data center optimization. Coarse-grained scheduling and initial placement of (virtual hosts of) components can be done using deployment information, e.g., virtual machine sizes and component classifications. For adaptive control and prediction of resource use however, more fine-grained modeling of the translation between incoming request patterns and internal as well as outgoing load is needed. Towards this end we define a simple component models designed to

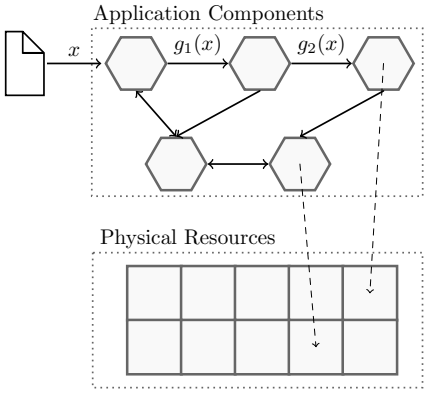


Fig. 1 Application model. Applications are modeled as connected graphs of components in the virtual layer, and mapped to physical compute resources in the physical layer.

quantify the relationships between incoming request patterns and a) internal load in the dimensions of CPU, RAM, I/O, and storage; and b) outgoing request patterns (illustrated as $f(x)$ and $g(x)$ respectively in Figure 2).

In this modeling we further make the observation that for each of these modeled entities there are significant differences between different types of applications in their load patterns. There are for example some applications that are load-wise driven directly by external requests, e.g., web servers where the needed resource capacity directly correlates to the type and amount of incoming requests (web servers are mostly idle when not processing HTTP requests), and other types of applications that are primarily driven by internal load factors, e.g., batch-oriented data processing applications with high data to compute ratios (i.e. that receive a few requests and then spend large amounts of time and resources performing computations that are not directly correlated to the incoming request patterns in any externally visible way). For this reason we also further decompose our component model to encompass the notion of foreground (load directly driven by incoming requests) and background (load not driven directly by incoming requests) load.

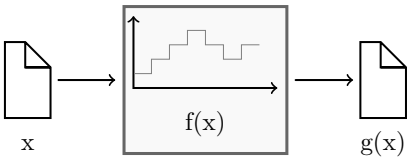


Fig. 2 Component model. Load impact and propagation functions detail how component requests result in internal virtual host load, as well as how requests are propagated to other components.

2.5 Optimization Problem

From a high level infrastructure provider perspective, cloud resource management can be seen as a complex dynamic assignment problem with a producer-consumer dynamic where demand exceeds supply. The challenge of the optimization then becomes to determine what amount of physical resource capacity (e.g., CPU, RAM, I/O, storage, network, etc.) is needed where (i.e. in what virtual host), and to control the scheduling, placement, and hardware resource allocations of virtual hosts to optimize the infrastructure efficiency. We address this optimization problem using a heuristics-based approach: we design heuristics functions that capture the behavior and impact of cloud applications and components for use as cost and evaluation functions, we formulate high-level objective functions to be used as guiding policies in automated infrastructure optimizations, and develop optimization algorithms that address scheduling and placement from a joint robustness-optimality perspective. Further information about the perspective of cloud infrastructure optimization used in this work is available in [2].

2.6 Resource Management Actions

Key to data center optimization is resource management efficiency. In our approach we structure resource management actions in four categories; placement, migration, vertical elasticity, and horizontal elasticity; and outline the high-level policies we map towards these actions. The main focus of this paper lies on the first two, but brief descriptions of the elasticity aspects are included for completeness.

2.6.1 Placement (Scheduling)

Scheduling, or placement as it is commonly called in the cloud domain, here refers to the initial assignment of virtual hosts to physical resources. As such, scheduling is an action that can have great impact on the efficiency of a data center, and is typically performed in accordance with a high-level policy that determines the objecive function of the placement. Illustrative examples of scheduling policies include, e.g., load balancing (spreading workloads more or less evenly over resources), consolidation (compacting as many workloads onto as few resources as possible), or energy efficiency (minimizing the energy footprint of the resources used to host the workloads).

As many traditional scheduling problems, cloud placement can be modeled as a parameterized bin packing problem with heuristics functions that capture costs and effectiveness of different scheduling actions. A key consideration here is the execution timeframe of the optimizations: Direct modeling of fitting functions (e.g., coarse-grained models that capture the CPU and RAM requirements of virtual hosts) allows formulation of fast greedy algorithms or even search-based methods for first or best fit algorithms that can be used for on-demand optimization of placement. More advanced models can employ, e.g., mixed integer-linear or constraint programming methods to find higher quality solution in continuous (e.g. over night) optimizations. Common to all of these approaches is that they require heuristics to model and evaluate different scheduling actions in optimization.

2.6.2 Migration (Rescheduling)

A key difference between cloud environments and traditional cluster environments is that some types of virtual hosts, e.g., virtual machines, are capable of being migrated. This can be done during runtime (live migration) or using a suspend-transfer-resume semantic (cold migration), and can have great impact on the efficiency of a data center. To illustrate how, consider another key difference between traditional cluster environment jobs and cloud services: the execution lifetime. Cluster jobs are typically scheduled using a batch scheduler and have well-define lifetimes. Cloud services on the other hand are typically deployed in virtual hosts that remain active until shut down (i.e. do not have traditional scheduling deadlines or even lifetimes that are known at the time of scheduling). Due to this, the ability of being able to migrate workloads within data centers can be greatly useful to allow placement decisions to be revisited to, e.g., use rescheduling to adapt to changed circumstances or data center events such as maintenance or hardware failures.

Migration does not come without a cost however. Cold migration incurs service interruption and a live migrated virtual host typically consumes hardware resources at both the source and destination physical resource during the migration. Both will also place a substantial (yet predictable) load on the network links between the resources. To be able to effectively use migration in data center optimization requires models that accurately capture the performance penalties and costs (in terms of resource capacity such as CPU, network bandwidth, and memory consumption) involved in the migration. This modeling is done based on knowledge of the migration technology used. For example in pre-copy live migration the virtual host memory pages are transfered to the destination host in advance, leaving the active (dirtied) pages to be retransmitted until only a core set remains and the process is quickly paused on the source and resumed on the destination (after the core set have been transfered). Similarly, if a post-copy live migration approach is used the control and core memory set is transfered and the process is resumed at the destination directly, and the remaining memory is pulled in (in order) as needed.

As a number of recontextualization issues (e.g., network connection management) also need to be handled, and the internal state of the virtual host depends on what the virtual host is currently doing, migration is at its most effective when properly scheduled (e.g., during a period of less activity). Effective scheduling of migration requires heuristics that accurately capture not only the direct costs of the migrations, but also the risks involved.

2.6.3 Vertical Elasticity

In addition to placement and migration, which essentally determine where a virtual host resides, cloud environments also typically provide dynamic control of how much hardware capacity a virtual host is assigned (vertical elasticity, from scaling up or down virtual hosts). Optimization of vertical elasticity is commonly done to facilitate overbooking of resources (i.e. fitting extra virtual hosts on a resource) by careful monitoring of application QoS metrics and the activity levels of (and interference between) the co-located virtual hosts. The term vertical elasticity is also sometimes used to denote the ability to change the size of the virtual hosts (e.g., changing the number of virtual CPUs in a virtual machine, rather than how many physical CPUs that can be used by the virtual CPUs).

2.6.4 Horizontal Elasticity

Finally, cloud environments also commonly entail a concept known as horizontal elasticity (from scaling out or in), which refers to the ability of adaptively changing the number of components or virtual hosts for a specific part of an application. This is a very powerful means of scaling a cloud application and requires whitebox knowledge of the application internals to, e.g., know to get an extra instance of a database server when the transaction load is too high on the existing instances. In this work we primarily target scheduling and rescheduling of virtual hosts, are thus focus on placement and migration of virtual hosts.

3 Conclusion

This abstract has outlined the various potential models and background issues that should be taken into consideration during the processes of scheduling and optimisation in cloud data centers. The intention of the work is to employ heuristics gleaned from real work environments and incorporate this knowledge within a meta/hyper heuristic framework to help dynamically schedule and reschedule resource within the allocation process. A classification of approaches taken within the literature will be presented for discussion at the conference along with an overview of the optimisation framework being developed within the CACTOS project i.e. CactoOPT.

The overall intention of this work is to raise awareness of cloud scheduling and optimsiation within the scheduling community which will help inform on 'best of breed' approaches and how these approaches can contribute to this increasingly important area. Importantly this is a real world application area that presents an opportunity for the scheduling community to become involved and trial techniques on large scale data sets. The intention is to use the conference as a starting point from which to build a body of work including an online repository of real world datasets and compilation of approaches and results. The authors are also interested in establishing a series of competitions which has proven to be successful in furthering the research in other areas involving optimising resource allocation i.e. Timetabling.

Acknowledgements The authors would like to ackowledge Jakub Krzywda and Ahmed Ali-Eldin for work related to the project. This work is funded by the European Union's Seventh Framework Programme under grant agreement 610711 (CACTOS).

References

1. A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, R. Subbiah, Worth their Watts? - An Empirical Study of Datacenter Servers. High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, pp. 1-10, 2010.
2. P-O. Östberg, H. Groenda, S. Wesner, J. Byrne, D. Nikolopoulos, C. Sheridan, J. Krzywda, A. Ali-Eldin, J. Tordsson, E. Elmroth, C. Stier, K. Krogmann, J. Domaschka, C. Hauser, PJ. Byrne, S. Svorobej, B. McCollum, Z. Papazachos, L. Johannessen, S. Rüth, and D. Paurevic. The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation. The 6th IEE International Conference on Cloud Computing Technology and Science (CloudCom 2014), pp. 26-31, 2014.

MISTA 2015

Variable neighbourhood search for rich personnel rostering problems

Pieter Smet · Greet Vanden Berghe

1 Introduction

Assigning shifts to employees is a problem solved daily in various organisations in health care, logistics, manufacturing, etc. Despite the many academic advances in the field of automated personnel rostering, this task is often still done manually. One reason for this research-application gap is that academic models for personnel rostering often fail to capture characteristics that cannot be neglected in practice.

The present contribution introduces a variable neighbourhood search (VNS) algorithm for solving feature-rich personnel rostering problems. The general object model introduced by Smet et al (2014) presents a flexible approach for modelling a large variety of rostering problems. This model was developed as part of a research project, and has been implemented in a commercial software system for staff scheduling.

Variable neighbourhood search is a well known metaheuristic which alternates between two phases to balance intensification and diversification (Mladenovic and Hansen, 1997). Bilgin et al (2012) applied VNS to a real world nurse rostering problem, illustrating the algorithm's flexibility and robustness on instances with varying characteristics. The present contribution introduces new neighbourhoods for the general model of Smet et al (2014), and presents new best results for a benchmark dataset.

2 General object model

Through reusable components, the model addresses common, yet complex problem characteristics which are often neglected in academic models. The model can represent problems in the class ASCNI|RVNO|PL, according to the $\alpha|\beta|\gamma$ notation of De Causmaecker and Vanden Berghe (2011).

Shift types are characterised by a start and end time, a net job time and a minimum rest time required before and after an assignment of the shift. Due to breaks, the difference between the start and end time often does not correspond to the actual job time. Therefore, the net job time is specified for each shift, which is used when counting

Pieter Smet, Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds - ITEC
E-mail: {pieter.smet, greet.vandenberghe}@cs.kuleuven.be

the total number of hours an employees has worked. According to the $\alpha|\beta|\gamma$ notation, the number of shifts is variable (N), and they can be overlapping (O).

Employees are characterised by a set of skill types, contracts and requests. The number of skills for which an employee is qualified and the organisation's skilling structure can vary significantly between different application contexts. Typically, one employee has multiple skill types, sometimes coupled with a level of experience. The general model allows the definition of such problems, modelling the complex interactions between employees that occur, for example, in a hierarchical skilling structure. According to the $\alpha|\beta|\gamma$ notation, the number of skills is variable (N), and can be defined individually (I).

Contracts consist of a number of time-related constraints and a period in which the contract is valid. The rules making up the contract are considered soft constraints, and determine the cost of an individual's roster through a weighted sum of violations (personnel objectives (P) in the $\alpha|\beta|\gamma$ notation). The threshold of each constraint is defined as a range, i.e. both a minimum and maximum value are specified. The time-related constraints are classified into three general types: counters, series and successive series (availabilities (A) and sequences (S) in the $\alpha|\beta|\gamma$ notation).

- **Counters** restrict the number of specific roster items in a certain period, defined by a start date and a number of days. This *counter period* does not need to match the scheduling period. Seven subjects can be restricted by counters: *hours worked*, *shift types worked*, *days worked*, *days idle*, *weekends worked*, *weekends idle* and *domain*.
- **Series** restrict consecutive occurrences of specific roster items. These constraints are valid in the period corresponding to the period of the contract. They are defined for five subjects: *shift types worked*, *days worked*, *days idle*, *weekends worked* and *weekends idle*.
- **Successive series** restrict two consecutive series; if the first series appears in a roster, it must be immediately succeeded by the second series. Similar to series, a successive series' period corresponds to the contract's period. Five successions of series are considered: *days worked* $\rightarrow$ *days idle*, *days idle* $\rightarrow$ *days worked*, *shift types worked* $\rightarrow$ *days idle*, *days idle* $\rightarrow$ *shift types worked* and *shift types worked* $\rightarrow$ *shift types worked*.

The time-related constraints are restricted to specific parts of the roster using *domains*. A domain is defined by a day set, shift type set and skill set, each delineating part of the problem in a different dimension. For example, the domain of the *maximum number of consecutive shift types worked* constraint is a restricted shift type set to indicate which shift types should be constrained. The domain counter constraint is included in the model as a very general type of counter constraint, which uses domains to identify the roster items whose occurrence is limited.

Apart from the time-related constraints, different scheduling constraints further restrict the assignments. This class of constraints contains three types: coverage requirements, collaboration preferences and training constraints. Coverage constraints express the number of employees required for each shift on each day as a range (fluctuating (V) and ranged coverage (R) in the $\alpha|\beta|\gamma$ notation). The model allows for a flexible definition of these constraints through skill type requirements and an associated minimum level of experience. Collaboration preferences model situations in which it is desirable to have a minimum or maximum number of employees from a subset of employees to be working together. Finally, training constraints allow for expressing guidelines regarding the training of personnel, e.g. employees who are less experienced

in a skill should work together with more experienced employees. These latter two constraints are chaperoning constraints (C) in the $\alpha|\beta|\gamma$ notation. All scheduling constraints have a weight, so that their violations can also be included in the weighted sum objective function (coverage objectives (L) in the $\alpha|\beta|\gamma$ notation).

Finally, the model includes elements which enable consistent constraint evaluation at the boundaries of the scheduling period. For all counter constraints, start and end values are defined for each employee individually. To allow consistent evaluation of the other constraint types, assignments from other scheduling periods can also be included.

3 Solution approach

Due to the large variety of soft constraints in the model, there are many neighbourhoods that need to be explored in order to reduce the number of constraint violations. As standard local search algorithms provide little support for multiple neighbourhood definitions, VNS was chosen to manage the different move operators.

In the first phase, Variable Neighbourhood Descent (VND) is used to reach locally optimal solutions. Several VND neighbourhoods are proposed which make both small and significant changes to the current solution. These neighbourhoods include commonly used ones, such as swapping one or multiple consecutive assignments between employees. However, due to the constraint definitions in the model, additional neighbourhoods are required that delete assignments or make new assignments. In the spirit of the original formulation of VNS, the order in which these neighbourhoods are explored is linked with their complexity.

The shaking phase attempts to escape local optima by perturbing the current solution. The neighbourhoods used in this phase make large changes to a solution, which cannot be achieved by only applying the VND move operators. Strong perturbations are obtained by swapping a single or all assignments between employees, and by applying a ruin-and-recreate heuristic. These neighbourhoods are ordered in a nested structure.

A computational study is performed to determine suitable parameter settings for the algorithm. Furthermore, the contribution and relative importance of the VND and shaking neighbourhoods are analysed. Computational results on instances based on real world data will be presented at the conference.

Acknowledgment: This research was carried out within the IWT 110257 project.

References

Bilgin B, De Causmaecker P, Rossie B, Vanden Berghe G (2012) Local search neighbourhoods for dealing with a novel nurse rostering model. Annals of Operations Research 194(1):33–57

De Causmaecker P, Vanden Berghe G (2011) A categorisation of nurse rostering problems. Journal of Scheduling 14(1):3–16

Mladenovic N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24(11):1097 – 1100

Smet P, Bilgin B, De Causmaecker P, Vanden Berghe G (2014) Modelling and evaluation issues in nurse rostering. Annals of Operations Research 218(1):303–326

MISTA 2015

A Hybrid Swarm Algorithm for Post Enrollment Course Timetabling

Cheng-Weng Fong • Hishammuddin Asmuni • Way-Shen Lam• Barry McCollum• Paul McMullan• Graham Kendall

Abstract The hybridization of metaheuristic approaches in solving optimization problems is increasingly the subject of many research projects. An important objective of hybridizing two or more metaheuristic approaches is to attain a balance between global exploration and local exploitation within the overall search process. This paper presents such an approach in terms of hybridising the population based approach Artificial Bee Colony (ABC) algorithm with a Hill Climbing (HC) local search and applying the resultant algorithm to the post enrollment course timetabling problem. In addition, the global best concept inspired from Particle Swarm Optimization (PSO) is used to improve the exploration of the basic ABC algorithm. The proposed hybrid approach is tested on Socha's course timetabling datasets and is compared against state-of-the-art approaches from the scientific literature. Experimental results

Cheng-Weng Fong
Department of Computer Sciences and Mathematics, Faculty of Applied Sciences and Computing, Tunku Abdul Rahman University College, Johor Branch Campus, 85000 Segamat, Johor, Malaysia
E-mail: fongcw@adu.tarc.com.my

Hishammuddin Asmuni
Software Engineering Research Group, Software Engineering Department, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
E-mail: hishamudin@utm.my

Way-Shen Lam
Software Engineering Research Group, Software Engineering Department, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
E-mail: lwshen007@gmail.com

Barry McCollum
Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom
E-mail: b.mccollum@qub.ac.uk

Paul McMullan
Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom
E-mail: p.p.mcmullan@qub.ac.uk

Graham Kendall
ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham and University of Nottingham Malaysia Campus, NG8 1BB, United Kingdom
E-mail: Graham.Kendall@nottingham.ac.uk

demonstrate that the proposed hybrid approach outperforms the basic artificial bee colony algorithm and is comparable to other previously presented approaches.

Keywords: Artificial bee colony algorithm, hill climbing local search, particle swarm optimization, post enrollment-based course timetabling

1 Introduction

University timetabling problems are complex optimization problems that have long attracted the attention of researchers from the areas of Artificial Intelligence and Operational Research. Various approaches have been proposed and trialed in addressing these problems. The earlier approaches (approximately pre 1990) were based on graph coloring heuristics (i.e. largest degree, largest enrollment, least saturation degree, largest weighted degree and largest coloured degree). These approaches were inspired by the similarities between the tasks of timetabling and graph colouring i.e. graph vertices represent events, colors represent time slots and edges between vertices correspond to conflicts between events [1, 2].

In the last two decades or so, metaheuristic (single solution and population based) approaches have been utilised in tackling university timetabling problems. Examples of single solution based approaches that have been successfully applied are simulated annealing [3, 4], iterative improvement [5], the great deluge algorithm [3, 6, 7] and variable neighborhood search [8]. Examples of population based approaches include genetic algorithms [9], honey bee mating algorithms [10] and harmony search [11]. Previous surveys and overviews regarding the application of metaheuristics in solving university timetabling problems are available in [12-15].

Hybridization of metaheuristic approaches have also been shown to be effective in addressing university timetabling problems. Abdullah *et al.* [16] developed a hybrid approach which combines variable neighborhood search with a genetic algorithm in addressing the university course timetabling problem. By referring to a recent comprehensive survey paper, Qu *et al.* [15] states that "*There are many research directions generated by considering the hybridization of meta-heuristic methods particularly between population-based methods and other approaches*". Hence, this paper presents a hybrid approach comprising the combination of Artificial Bee Colony (ABC) algorithm, Particle Swarm Optimization (PSO) and Hill Climbing (HC) in solving the post enrollment course timetabling problem. The inclusion of the latter two approaches is motivated by the creation of a better balance within the ABC approach between exploration and exploitation.

This paper is structured as follows. Section 2 introduces the university course timetabling problem. The components that employed for the hybrid algorithm are presented in Section 3 and the proposed approach is discussed in Section 4. Simulation results are presented in Section 5 and Section 6 provides concluding remarks and a discussion on possible future work and approaches.

2 Problem Description

The university course timetabling problem involves the allocation of courses or lectures into a limited set of time slots (normally on a weekly basic) and rooms while satisfying a set of predefined constraints [17]. Constraints can be categorized into two types, i.e. hard constraint and soft constraints. Hard constraint must be satisfied (feasible solution) under all circumstances, while violation of soft constraints should be minimized as far as possible, and this reflects the quality of the solution produced. A timetable solution that satisfies all soft constraints is an optimal solution. The quality of a timetable solution is evaluated based on the closeness of the timetable solution in terms of approaching the optimal [7].

University course timetabling can be divided into two types, i.e. post enrollment-based and curriculum-based course timetabling problems. The former is investigated in this paper. The main aspect that differentiates these two problems is that the post-enrollment course timetabling focuses on students' preferences while curriculum course timetabling concentrates on constructing timetables providing maximum flexibility within the curriculum structure [18]. The post enrollment course timetabling problem datasets investigated in this paper were introduced by Socha *et al.* [19]. This dataset includes 11 instances consisting of 5 small, 5 medium and 1 large instance.

The goal is to allocate all courses into timeslots and rooms into a weekly timetable subject to a set of hard constraint(s) and soft constraint(s). The problem description described here is adopted from Socha *et al.* [19] and the characteristics of the dataset can be seen in TABLE 1.

- *M*, number of students;
- *C*, number of courses;
- *T*, a set of predefined timeslots ($t_n \ldots T$, where T=45, n=1…T);
- *R*, number of rooms ($r \ldots R$, r=1,2,3…,R);
- *F*, a set of room features.

The goal of this problem is to allocate courses *C* into a number of timeslots *T* and rooms *R* permitted such that all the stated hard constraints discussed below are satisfied:

Hard constraints
- All students are required to attend only one course at one time.
- A course must be allocated to a room that satisfies feature required for that course.
- A course must be allocated into a room that can serve students that attend for that course.
- Only one course can be scheduled into a room at any timeslot.

In addition to satisfying the hard constraints stated above, the timetable solutions should minimize the violation on following soft constraints:

Soft constraints
- No student is requested to attend only one course in a day.
- No student is requested to attend three or more consecutive courses in a day.
- No student is requested to attend a course that scheduled in the last timeslot of a day.

The penalty cost calculation is based on the violation of each soft constraint per student. A penalty cost of 1 will be assigned for the violation of any soft constraint per student.

TABLE 1
Characteristics of post enrollment-based course timetabling problem

	Small	Medium	Large
Number of courses	100	400	400
Number of rooms	5	10	10
Number of timeslots	45	45	45
Number of features	5	10	10
Approx features per room	3	3	3
Percent feature use	70	80	90
Number of students	80	200	400
Max events per student	20	20	20
Max students per event	20	50	100

3 Hybridization Components

3.1 Artificial Bee Colony Algorithm

This section presents the idea of the ABC algorithm that was introduced by Karaboga [20] which is inspired by the foraging behavior of honey bees when they are exploring food sources (solutions) to collect the nectar (fitness value). Artificial bee colony has been successfully applied in addressing various optimization problems [21-24]. There are three phases in the ABC algorithm i.e. employed, onlooker and scout bee phases. The number of employed bees and onlookers are the same and they cooperate in terms of improving quality of food sources of the population in the chosen search region.

In the employed bee phase, employed bees explore for food sources in the search region. The food source searching process is represented by neighborhood search. Each bee searches for a neighborhood food source of the current food source and will accept that food source if the quality is better than current food source.

In the onlooker bee phase, the onlooker will seek for a better food source than the food sources found in employed bee phase. The food source selection is based on a probability scheme (roulette wheel selection) where food sources with better quality of nectar will have a higher chance of being chosen. The search process is repeated until all the onlookers perform the search process on selected food sources.

Employed bees will turn into scout bees if they are unable to find better food sources after an identified period. They will discard current food sources (exhausted food sources) and look for a new food source in the search region.

3.2 Particle Swarm Optimization and Hill Climbing

Particle swarm optimization is a population-based approach that was introduced by Kennedy and Eberhart [25]. The idea of this approach is based on the food searching behavior of birds or fishes. In PSO, one of the solution searching process is guided by global information [26]. Global information refers to the best solution found so far in a population (global best concept). This concept can improve the exploration ability of the search process since the search is focused on exploring promising search regions.

Hill climbing is a simple single solution based local search approach. During the search, the approach evaluates the neighborhood solutions iteratively until a local optima

solution obtained based on greedy selection scheme. This scheme accepts only improved solutions which often make the search process entrapped in local optima and poor resultant solution are subsequently obtained [7]. However, this problem can be addressed by hybridizing with other approaches. For example, a hybrid method that consists of constraint programming, simulated annealing and HC proposed by Merlot *et al.* [27] has proven to be effective when applied to the closely related examination timetabling problem.

4 Global Best Concept Artificial Bee Colony Algorithm

Global best concept artificial bee colony (GBABC) algorithm is an approach that is based on hybridization of the global best concept of PSO and HC local search. The pseudo code for the proposed approach is shown in Fig. 1. This algorithm starts by generating initial solutions using a hybrid construction approach proposed by Chiarandini *et al.* [28] and Landa-Silva and Obit [29]. The hybrid construction consists of a largest degree heuristic phase followed a by neighborhood search and tabu search phase if no feasible solution is generated initially.

Employed Bee Phase.

In this phase, all the employed bees explore for food sources using the global best concept (Section 3.2). As discussed, this is with the aim of leading the search process exploration toward promising search region rather than blindly exploring toward unknown search regions. The global best concept is achieved by generate new offspring that inherit the best solution found so far using haploid crossover [30]. From this process, the offspring generated subsequently contains certain information of the best solution and will therefore accordingly intensify the search on the best solution region. Details on the implementation of haploid crossover is presented in author's previous works [31].

Onlooker Bee Phase.

Rather than exploiting only the selected food source in the onlooker bee phase of the basic ABC algorithm, HC local search (Section 3.2) is incorporated in this phase to fine-tune (seek for local optima solution) all the food sources explored within the employed bee phase. This is carried out with the aim of improving the exploitation ability in searching local optimal solutions. Two neighborhood structures used to generate tentative solutions are employed as below:

- **N1:** Randomly selects a course and moves into another feasible time slot and room.
- **N2:** Randomly selects two courses and swap their time slots and rooms. The feasibility of the timetable solution is maintained at the same time.

During the execution of HC local search, N1 or N2 is used to generate neighborhood solution. The selection of N1 or N2 is based on probability selection. Prior to the selection of N1 or N2, a random number within the 0 to 1 range will be generated. If the random number generated falls within the range of 0 to 0.5, N1 will be selected to generate the neighborhood solution. Otherwise, N2 will be selected.

Scout Bee Phase.
In scout bee phase, scout bees serve as explore colonies to search for new food sources. Hence, the entire solutions in the population are discarded after improved (fine-tuned) by the Onlooker bees. New set of solutions are generated randomly to replace the discarded solutions.

```
Initialization:
Set population size (SN), initialize the population;
Calculate fitness value of each solution, f(sol);
Identify global best solution, sol_BS;
Set number of iterations for ABC, NumItr_ABC;
Set number of iterations for HC, NumItr_HC;

Improvement:
For t = 1 to NumItr_ABC do
     For i = 1 to SN do  //Employed Bee Phase
          Incorporate information of best solution,
          sol_BS into sol_i based on global best concept
          using haploid crossover;
     End For i

     For i = 1 to SN do  //Onlooker Bee Phase
          For h = 1 to NumItr_HC do  //HC local search
               Select solution sol_i and generate sol_i'
               by performing neighborhood search;
               If f(sol_i') ≤ f(sol_i)
                    Sol_i ← sol_i';
               End If
          End For h;
          If f(sol_BS) < f(sol_i)
               sol_BS ← sol_i;
          End If
     End For i

     //Scout Bee Phase
     All the solutions (sol) in the population are
     abandoned and new solutions (sol_new) are generat-
     ed randomly, sol ← sol_new;
     Calculate fitness value for each new solutions,
     f(sol);
End For t
```

Fig. 1. Pseudo code for global best concept artificial bee colony algorithm

5 Experimental Results

The proposed algorithm was coded using C++ programming and simulations were carried out on the Intel Core i7 2.2 Ghz laptop. In addition, the proposed approach was tested against the post enrollment-based course timetabling problem (Section 2) with 30 test runs for each instance. The experiments are carried out with 50 population size and 2000 iterations for

HC local search. Note that the experiments carried out were terminated when the iteration for artificial bee colony algorithm equaled 10000. The computational times used are different and depend on the size of each instance. For the basic ABC algorithm and proposed approach, the computational time for each instance took 15 minutes to 2 hours and 1 to 8 hours, respectively. These computational times are acceptable since in real world, the timetables generation process is carried out a few months before the actual use of the resultant timetable solution [8].

TABLE 2 illustrates the Friedman test and provides a comparison between results obtained by the basic ABC algorithm, the proposed approach and published results from the literature. In general, the proposed approach outperformed the basic ABC algorithm in all instances and manages to produce competitive results with other published approaches. It can be seen that the proposed approach is capable of producing feasible timetables for all instances. Besides that, the proposed approach manages to obtain optimal solutions with no soft constraints violation for all small instances and performed well on the remaining instances. In addition, among the approaches compared with, results of medium 01 and medium 04 ranked first, followed by medium 02 and medium 05 ranked in second place, respectively. Based on the Friedman test (ranking) conducted as shown in Table 2, it can be seen that Al-Betar *et al*. [32] ranks first, followed by the approached proposed here, Shengxiang and Jat [33], Abdullah *et al*. [34], Abdullah *et al.* [8], Landa-Silva and Obit [35] and the basic ABC algorithm in decending order.

TABLE 2
Friedman test and result comparison

Instance	Basic ABC	(GBABC)	Shengxiang and Jat [33]	Landa-Silva and Obit [35]	Abdullah *et al.* [8]	Abdullah *et al*. [34]	Al-Betar *et al*. [32]
small01	25	**0**	**0**	**0**	**0**	**0**	**0**
small02	32	**0**	**0**	1	**0**	**0**	**0**
small03	23	**0**	**0**	**0**	**0**	**0**	**0**
small04	15	**0**	**0**	**0**	**0**	**0**	**0**
small05	24	**0**	**0**	**0**	**0**	**0**	**0**
medium01	393	**83**	139	126	221	98	99
medium02	436	74	92	123	147	113	**73**
medium03	483	154	**122**	185	246	123	130
medium04	422	**96**	98	116	165	100	105
medium05	415	73	116	129	130	135	**53**
large	908	665	615	821	529	610	**385**
ranking	7.000	2.909	3.181	4.454	4.363	3.363	2.727

Fig. 2, 3 and 4 represent the convergence graph of ABC and the proposed approach in searching solutions for instances small 02, medium 01 and large. It can be observed that the slope for the proposed approach is relatively steeper than basic ABC algorithm (for all graphs). This is due to a large improvement in term of solution quality (using HC local search) at the early stage of the search process for the proposed approach. In addition, the convergence of the search process associated with the proposed approach is relatively better than the basic ABC algorithm. However, the increment of quality of solution slows down and decreases as the increase of number of iteration for both basic and proposed approaches.

Box plots which demonstrate the distribution of best, first quartile, average, third quartile and worst solution qualities generated by proposed approach can be seen at Fig. 5. It can be observed that there is a close gap for best and average values which indicates that proposed approach is stable in searching for solutions for all instances.

TABLE 3 presents the statistical analysis (*t*-test) on the performance of the basic ABC over the proposed approach at the level of significant of 0.05. The null hypothesis (H_0) is defined as there is no different between the performances of both approaches, whereas, the alternative hypothesis (H_1) is defined as the performance of proposed approach is better than basic ABC. The *p*-values for all instances are smaller than 0.05 which indicates that there exists strong evidences to support the claim on H_1 and reject H_0. From this experiment, it can be concluded that the performance of the proposed approach is significantly better than basic ABC in solving the Socha course timetabling problem.

From the experimental results, it is believe that by introducing the global best concept in the ABC within the employed bee phase it is possible to enhance the exploration ability of the search process and accordingly leads the search process to more promising search regions. Furthermore, it is also believe that by hybridizing with an HC local search approach the approach is capable of improved the local exploitation of the ABC algorithm in fine-tuning the solutions explored within the employed bee phase to local optima points.

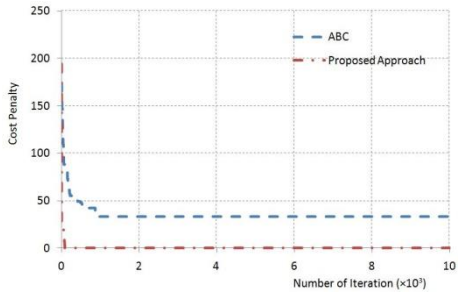


Fig. 2. Convergence graph of basic ABC algorithm and proposed approach for instance small02

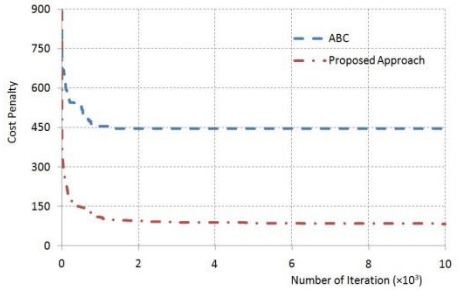


Fig. 3. Convergence graph of basic ABC algorithm and proposed approach for instance medium01

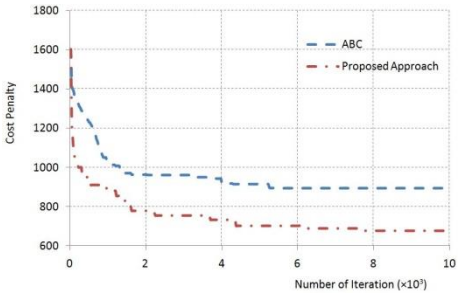


Fig. 4. Convergence graph of basic ABC algorithm and proposed approach for instance large

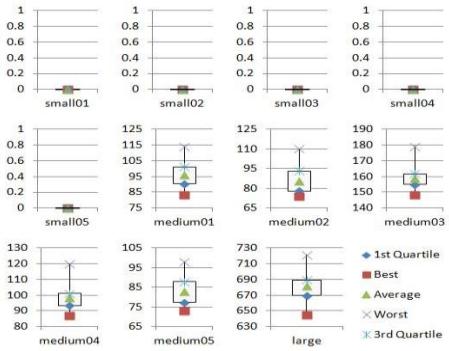


Fig. 5. Box plots of Socha course timetabling dataset

TABLE 3
Statitical Analysis for basic ABC and proposed approach on Socha dataset

Instance	Basic ABC	GBABC	*t*-test
	Average	Average	*p*-value
small01	31.03	0	< 0.05
small02	36.70	0	< 0.05
small03	26.57	0	< 0.05
small04	20.83	0	< 0.05
small05	28.67	0	< 0.05
medium01	433.67	95.87	< 0.05
medium02	470.50	85.17	< 0.05
medium03	499.80	158.90	< 0.05
medium04	443.93	98.23	< 0.05
medium05	455.73	2.67	< 0.05
large	991.67	681.33	< 0.05

6 Conclusion

This paper proposes an algorithm that is based on the basic ABC algorithm to tackle the university post enrollment-based course timetabling problem, where three types of bees are cooperating in searching for food sources within the search region. Each bee represents a feasible solution in the population and the search process is carried out by employed, onlooker and scout bees. The searching behavior of the employed bee is based on the global best

concept that is inspired from PSO where solutions are explored around the best region. For onlooker bees, a HC local search is used as an improvement approach to fine-tune the solutions search region (explored by employed bee) in the population and followed by a solution abandoned process carried out by scout bees where the solutions in the population are replaced by new solutions. The key feature of the proposed approach is the complement of the strengths of population-based and single solution based approaches that significantly enhanced both exploration and exploitation abilities of the ABC algorithm. With the exploration, exploitation and solution replacement behaviors in each cycle, the proposed algorithm has proven to be effective in tackling Socha course timetabling problems (post enrollment-based). The proposed approach can be applied in solving curriculum based course timetabling that was introduced as part of the 2nd International Timetabling Competition in 2007 and this is subject to future work.

References

1. D. J. Welsh and M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *The Computer Journal*, 10(1), 85-86 (1967).
2. D. Wood, A technique for colouring a graph applicable to large scale timetabling problems, *The Computer Journal*, 12(4), 317-319 (1969).
3. E. K. Burke, Y. Bykov, J. Newall and S. Petrovic, A time-predefined approach to course timetabling, *Yugoslav Journal of Operations Research*, 13(2), (2003).
4. J. Thompson and K. Dowsland, A robust simulated annealing based examination timetabling system, *Computers & Operations Research*, 25(7), 637-648 (1998).
5. S. Abdullah, E. K. Burke and B. McCollum, Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem, *Metaheuristics*, 153-169, Springer, (2007).
6. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke and S. Abdullah, An extended great deluge approach to the examination timetabling problem Dublin, (2009).
7. P. McMullan, An extended implementation of the great deluge algorithm for course timetabling, *Computer Science 7th International Conference Part I, Proceedings ICCS 2007, Lecture Notes in Computer Science*, Pages: 538 - 545 Springer, (2007).
8. S. Abdullah, E. K. Burke and B. McCollum, An investigation of variable neighbourhood search for university course timetabling, *The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, New York, USA, July, pp. 413-427 (2005).
9. S. Abdullah and H. Turabieh, Generating university course timetable using genetic algorithms and local search, IEEE Computer Society, (2008).
10. N. R. Sabar, M. Ayob, G. Kendall and R. Qu, A honey-bee mating optimization algorithm for educational timetabling problems, *European Journal of Operational Research*, 216(3), 533-543 (2012).
11. M. A. Al-Betar and A. T. Khader, A harmony search algorithm for university course timetabling, *Annals of Operations Research*, 194(1), 1-29 (2012).
12. E. K. Burke, K. Jackson, J. Kingston and R. Weare, Automated university timetabling: The state of the art, *The Computer Journal*, 40(565-571 (1997).
13. M. W. Carter and G. Laporte, Recent developments in practical course timetabling, *Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*, 3-19, Springer, Toronto, Canada (1998).
14. R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, *OR Spectrum*, 30(1), 167-190 (2008).
15. R. Qu, E. K. Burke, B. McCollum, L. T. Merlot and S. Y. Lee, A survey of search methodologies and automated system development for examination timetabling, *Journal of Scheduling*, 12(1), 55-89 (2009).

16. S. Abdullah, E. K. Burke and B. McCollum, A hybrid evolutionary approach to the university course timetabling problem, *IEEE Congress on Evolutionary Computation, 2007*, pp. 1764-1768 (2007).
17. K. Socha, M. Sampels and M. Manfrin, Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, *Applications of Evolutionary Computing*, 334-345, Springer-Verlag, Essex, UK (2003).
18. S. Abdullah, H. Turabieh, B. McCollum and P. McMullan, A hybrid metaheuristic approach to the university course timetabling problem, *Journal of Heuristics*, 18(1), 1-23 (2012).
19. K. Socha, J. Knowles and M. Sampels, A MAX-MIN ant system for the university course timetabling problem, *Ant Algorithm 3rd International Workshop, ANTS 2002, Lecture Notes in Computer Science*, 1-13, Springer, Brussels, Belgium (2002).
20. D. Karaboga, An idea based on honey bee swarm for numerical optimization, Erciyes University, (2005).
21. W. Gao and S. Liu, Improved artificial bee colony algorithm for global optimization, *Information Processing Letters*, 111(17), 871-882 (2011).
22. F. Kang, J. Li and Q. Xu, Structural inverse analysis by hybrid simplex artificial bee colony algorithms, *Computers & Structures*, 87(13–14), 861-870 (2009).
23. M. F. Tasgetiren, Q.-K. Pan, P. N. Suganthan and A. H. L. Chen, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, *Information Sciences*, 181(16), 3459-3475 (2011).
24. G. Zhu and S. Kwong, Gbest-guided artificial bee colony algorithm for numerical function optimization, *Applied Mathematics and Computation*, 217(7), pp. 3166-3173 (2010).
25. J. Kennedy and R. Eberhart, Particle swarm optimization, *IEEE International Conference on Neural Networks, 1995* Perth, WA, Australia, pp. 1942-1948 vol.4 (1995).
26. J. Kennedy and R. C. Eberhart, A discrete binary version of the particle swarm optimization, *In: Proceedings of the World Multiconference on Systemics. Cybernetics and Informatics*, IEEE Service Center, Piscatway. NJ., pp. pp. 4104-4109 (1997).
27. L. G. Merlot, N. Boland, B. Hughes and P. Stuckey, A hybrid algorithm for the examination timetabling problem, *Selected Papers from 4th International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*, 207-231, Springer, Gent, Belgium (2003).
28. M. Chiarandini, M. Birattari, K. Socha and O. Rossi-Doria, An effective hybrid algorithm for university course timetabling, *Journal of Scheduling*, 9(5), 403-432 (2006).
29. D. Landa-Silva and J. H. Obit, Great deluge with non-linear decay rate for solving course timetabling problems, *4th International IEEE Conference Intelligent Systems, 2008*, pp. 8-11-8-18 (2008).
30. H. A. Abbass, MBO: Marriage in honey bees optimization-A haplometrosis polygynous swarming approach, *Proceedings of the 2001 Congress on Evolutionary Computation*, Seoul, pp. 207-214 (2001).
31. F. C. Weng and H. Asmuni, An Automated Approach Based On Bee Swarm in Tackling University Examination Timetabling Problem, *International Journal of Electrical and Computer Sciences*, 13(02), 8-23 (2013).
32. M. A. Al-Betar, A. T. Khader and M. Zaman, University Course Timetabling Using a Hybrid Harmony Search Metaheuristic Algorithm, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(5), 664-681 (2012).
33. Y. Shengxiang and S. N. Jat, Genetic algorithms with suided and local search strategies for university course timetabling, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(1), 93-106 (2011).

34. S. Abdullah, K. Shaker, B. McCollum and P. McMullan, Incorporating great deluge with kempe chain neighbourhood structure for the enrolment-based course timetabling problem, *Rough Set and Knowledge Technology*, 70-77, Springer Berlin Heidelberg, (2010).
35. D. Landa-Silva and J. Obit, Evolutionary non-linear great deluge for university course timetabling, *Hybrid Artificial Intelligence Systems, 4th International Conference, Proceedings HAIS 2009, Lecture Notes in Computer Science*, 269-276, Springer, Salamanca, Spain (2009).

MISTA 2015

An integrated simultaneous approach for pilots and copilots re-scheduling problem

Atoosa Kasirzadeh · François Soumis

Abstract Various sources of unpredicted disturbances may affect the planned schedules of airline crew members. These disruptions may result in delayed/canceled flights and will affect the crew schedules. In this paper, we solve the simultaneous recovery problem for pilots and copilots by using an integrated approach to re-optimize both the pairings and personalized monthly plans. We propose a set-partitioning model for this problem. Column generation solution approach is used to solve such problem.

1 Extended Abstract

Due to complexity and unpredicted perturbations, airline decision making procedure is mainly divided into *planning* and *recovery*. Because of the very large size of the airline planning procedure, a global single-step optimization of this problem is very challenging and so far hardly achievable. As a result, the airline planning procedure is frequently divided into smaller optimization problems involving *flight scheduling*, *fleet assignment*, *aircraft maintenance and routing*, and *crew scheduling*. Due to the very large size of the problem, crew scheduling is itself divided into *crew pairing* and *crew assignment*. The crew pairing problem builds a minimum cost set of pairings based on the scheduled flights such that the collective agreements and rules are respected. The crew assignment problem combines the pairings, vacations, pre-assigned activities, and rest periods in order to construct and assign a set of monthly schedules to crew members while respecting the regulations and collective agreement rules. This problem is either solved as *bidline* or *personalized*. Through bidline approach, anonymous monthly schedules are constructed

First Author
GERAD & École Polytechnique de Montréal, Department of Mathematics and Industrial Engineering
E-mail: atoosa.kasirzadeh@polymtl.ca

Second Author
GERAD & École Polytechnique de Montréal, Department of Mathematics and Industrial Engineering
E-mail: francois.soumis@polymtl.ca

and assigned to crew members. Personalized assignment problem is solved either as *rostering* or *seniority-based*. Rostering approach aims at maximizing the global satisfaction. Seniority-based assignment aims at prioritizing the maximization of satisfaction of the more senior crew members. Traditionally, the crew pairing and crew assignment problems have been solved sequentially. However, more recently some researchers attempt in integrating the two steps.

On the day of operation, external and/or internal perturbations may occur which result in delayed or canceled flights directly affect the planned crew schedules. To adjust the planned decisions of airlines, often airline recovery steps are treated by a sequential approach; that is, respecting the disruptions, flights are re-scheduled, aircraft re-planning is solved, crew schedules are recovered, and finally the passenger recovery problem is addressed.

In this study, we focus on airline crew recovery problem, as the cost of crew members is the second largest cost of airlines (after aircraft fuels). The majority of the mathematical methods and solution approaches for solving the crew recovery problems are similar to the methods applied for planning purposes. However, there are five major differences between the crew recovery and crew planning problem. Firstly, crew recovery problem cannot be separated in two steps: crew pairing and crew assignment. The reason is that the updated pairings have to fit well whithin the planned monthly schedules; in other words, it is an update for the monthly schedules. Therefore, the integration of constructing pairing and adjustment of monthly schedules is necessary. Secondly, the pilots and the copilots are needed to be treated simultaneously. In fact, the pairings are required to be common for pilots and copilots, as much as possible, to maintain the robustness of the solution. When the pairings are different between pilots and copilots, a flight perturbation can disturb two different pairings which will disturb more flights and will result in disturbance propagation throughout the monthly schedule. Thirdly, the crew recovery problem is very time restrictive; that is, quick optimized solutions should be offered in response to the unpredicted disturbances whereas crew planning problem is not very time restrictive and is solved several weeks prior to the planning month. Fourthly, the crew planning problem is a scheduling optimization for a planning period (frequently one month) whereas crew recovery re-optimizes the schedules locally (within the recovery window of few days); therefore, the dimension of the recovery optimization problem is reduced. Fifthly, the objectives of crew planning problem are mainly defined in terms of cost minimization and efficient crew utilization whereas several objectives during airline recovery procedure are in conflict with each other. An example of such conflicting objectives are minimizing the delay of crew members and minimizing the cost of the recovery operations. Because recovery problem is a fast real-time problem, its dimensions are needed to be small to be solved in reasonable time. However, re-optimization domain considered for crew recovery problem must be sufficiently large in order to allow finding feasible schedules taking into account the re-scheduled tasks. The main crew recovery concern is to cover, in a most cost efficient way, the set of given flights while remaining as close as possible to the original crew monthly schedules. The crew recovery process consists in applying various actions to repair at the minimum cost, the original crew schedules while minimizing the number of flights that cannot be operated due to lack of sufficient crew on board. Crew recovery actions include, but are not limited to, re-scheduling already scheduled crews or deploying limited reserve crew members.

The contribution of this study lies in developing an optimization approach for solving the integrated pairing recovery and assignment recovery problem for pilots and copilots simultaneously. This integrated approach considers at the same time the decisions on pairing re-optimization and recovery of monthly plans for crew members. This approach implements both the regulations related to pairing and monthly schedules. This recovery problem is solved for pilots and copilots simultaneously in order to provide more robust schedules. In other words, if the pairing of pilots and copilots are the same, the propagation of disruption to other flights in the schedule is reduced. The re-scheduled flights are considered as input data for our problem. To the best of our knowledge, this study introduces the first attempt in developing a mathematical programming method for the simultaneous recovery problem for pilots and copilots. The main contribution of this paper is to show that this mathematical programming method is able to solve the personalized recovery problem for test instances with up to 610 pilots and copilots quickly enough. We solve this problem by using a column-generation solution approach.

MISTA 2015

Multi-objective energy-aware scheduling

David Van Den Dooren · Thomas Sys · Tony Wauters · Greet Vanden Berghe

1 Introduction

Scheduling determines the way in which jobs are assigned to resources. Multiple resources, e.g. machines and human operators, are available for the problem under consideration. Jobs and resources are defined by various characteristics and constraints, required to match in feasible assignments. Manufacturing companies strive for good quality schedules, in terms of operational efficiency and custom-related objectives. Makespan and tardiness are two objectives often separately considered during single objective optimisation. These objectives are denoted as "business objectives" and show a latent correlation. For example, makespan optimisation may positively influence the total tardiness of the schedule.

In the last years, energy consumption has gained considerable attention as the cost (kWh) impacts the total production cost in energy-intensive sectors. Hence, the need for minimising energy consumption and, consequently, energy cost increases. (Van Den Dooren et al., 2015) define a methodology for addressing multi-machine scheduling problems with the focus on minimising energy consumption. Experiments were conducted on the ICON challenge benchmark datasets (O'Sullivan et al., 2014), providing both real and forecasted energy cost data. The energy cost is time dependent and is enforced by assigning a corresponding energy price to every time slot. The energy consumption depends on resource requirements during execution of the jobs.

To take into account both energy and business objectives, a multi-objective optimisation approach is needed. Multi-objective approaches have been researched thoroughly (Varadharajan and Rajendran, 2005; Pasupathy et al., 2006). The present work focuses on analysing the energy objective so as to determine a detailed and specific energy modelling approach. Additionally, alternative multi-objective approaches for combining business and energy objectives are firmly researched. The influence of both objectives are analysed. Experiments are conducted using the MOLA (Multi-Objective Late Acceptance) method (Vancroonenburg and Wauters, 2013).

David Van Den Dooren, Thomas Sys, Tony Wauters · Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC
E-mail: {david.vandendooren,thomas.sys,tony.wauters,greet.vandenberghe}@cs.kuleuven.be

2 Approach

Energy cost reduction is a relatively new scheduling objective. Extensive research is needed in order to determine the objective's inherent characteristics. Subsequently, relations with business objectives can be defined, e.g. supportive or conflicting nature of the objectives. Previous research (Van Den Dooren et al., 2015) provided some insights and a methodology concerning energy consumption modelling. Multiple schedule characteristics influence energy consumption, e.g. machine states, electricity cost per time period. The introduced methodology implements a LAHC (Late Acceptance Hill Climbing, Burke and Bykov (2012)) approach with multiple neighbourhoods.

An extension to previous research is carried out by implementing the MOLA method. MOLA consists of LAHC with Pareto dominance evaluation. The method works as follows. New solutions are generated using neighbourhoods and are accepted based on the Pareto dominance relation (Drugan and Thierens, 2012). Current best, pairwise non-dominating, solutions are saved in the Pareto set. The new solution is compared, accepted and added to the Pareto set when its objective value dominates the objective value a few iterations ago. Thus, the dominating solution replaces the oldest solution in the set. When the method come to a halt after having reached its stop criteria, this method could provide the Pareto front, which defines the best solutions for specific objective settings. Figure 1 illustrates the MOLA methodology.

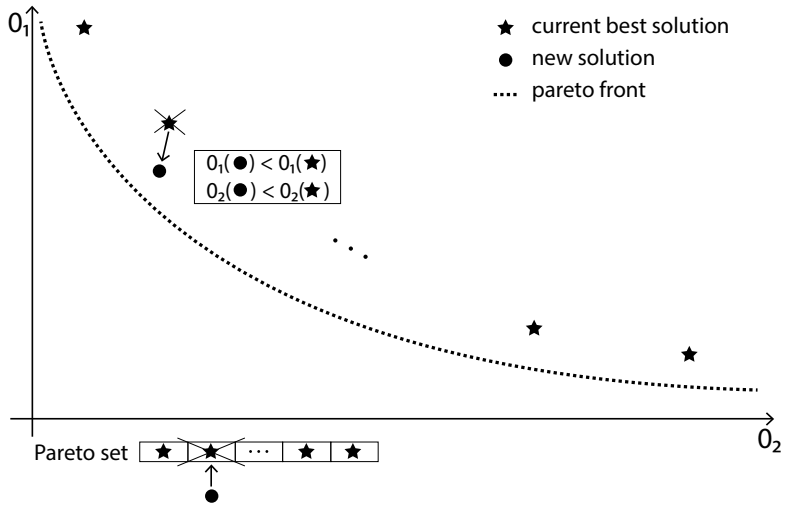


Fig. 1: MOLA methodology for a bi-objective example

3 Experimental Setup

3.1 Data

New datasets, based on the ICON benchmark sets (O'Sullivan et al., 2014), have been generated in order to investigate the effect of an energy cost objective being optimised

simultaneously with business objectives. Real energy data, energy cost per time period, is provided within the ICON benchmark sets. However, the ICON benchmark instances contain restrictions, e.g. fixed time horizons, disabling possible business objectives. Thus, modifications to the general time restrictions are necessary: the time horizon is increased, and the jobs' time characteristics are modified. These changes enable incorporating business objectives such as makespan and tardiness. In addition to the academic datasets, real datasets have been collected in industry in order to enlarge the test environment and validate the developed optimisation approach.

3.2 Experiments

The experiments can be divided into three parts: objective function analysis, multi-objective optimisation and sensitivity analysis. They are performed using the MOLA method. The objectives are examined both individually and in combination. To this end, the Pareto objective approach is examined first. Secondly, lexicographical and weighted objective function tests are executed for different objective settings. A sensitivity analysis is provided by defining mutual objective influences, examining various objective settings, and comparing the aforementioned multi-objective approaches. Finally, a suggestion on how to approach multi-objective energy-related scheduling problems is given. The end results contain both the influence of problem specific characteristics and the effect of simultaneously optimizing different objectives.

Acknowledgements SENCOM is a project co-funded by iMinds, a digital research institute founded by the Flemish Government. Project partners are Nervia Plastics, Objective, Delta Engineering and Sagility, with project support from IWT. Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX.

References

Burke, E.K., Bykov, Y., 2012. The late acceptance hill-climbing heuristic. Department of Computing Science and Mathematics University of Stirling - Technical Report CSM-192 - ISSN 1460-9673 .

Drugan, M., Thierens, D., 2012. Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. Journal of heuristics 18(5), 727–766.

O'Sullivan, B., Hurley, B., Simonis, H., Mehta, D., De Cauwer, M., 2014. ICON challenge on forecasting and scheduling. http://iconchallenge.insight-centre.org/challenge-energy. UCC - University College Cork - ICON.

Pasupathy, T., Rajendran, C., Suresh, R.K., 2006. A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. The International Journal of Advanced Manufacturing Technology 27, 804–815.

Van Den Dooren, D., Sys, T., Wauters, T., Vanden Berghe, G., 2015. Multi-machine energy-aware scheduling. Technical Report - KU Leuven .

Vancroonenburg, W., Wauters, T., 2013. Extending the late acceptance metaheuristic for multi-objective optimization. MISTA .

Varadharajan, T.K., Rajendran, C., 2005. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flow time of jobs. European Journal of Operational Research 167, 772–795.

MISTA 2015

A Greedy-based Heuristic for Shift Minimization Personnel Task Scheduling Problem

Mehran Hojati

1 Introduction

There are situations where tasks have predetermined (known) start time & duration and require a particular type of skill. One example is the maintenance check of aircraft engines at an airport between a landing and subsequent takeoff. These tasks have to be assigned to mechanics who are each certified to work on only limited types of engines. Shift Minimization Personnel Task Scheduling Problem (SMPTSP) is the assignment of predetermined tasks (with known start time and duration) to a minimum number of heterogeneous workers with predetermined shifts.

Solving SMPTSP to optimality is NP-complete. It can be formulated as an integer linear program (ILP) and solved to optimality for small to medium-sized problems. However, for large problems a heuristic must be used.

Recently, there have been three articles reporting various methods to solve large-scale SMPTSPs. Krishnamoorthy et al. [2] used lagrangian relaxation. They relaxed the task assignment constraints and used the deviations in the objective function with lagrange coefficients, then solved each worker's problem independently. The iterative process seeks to find the right lagrange coefficients such that each task is assigned to exactly one worker. Even though their method produced good solutions for most of 137 randomly-generated problems (with 0 to 22% optimality gap), it was unable to find a feasible solution for some of the largest problems.

Lin and Ying [3] proposed a 3-phase algorithm that was able to solve most of Krishnamoorthy's test problems. In phase 1, they used a constructive heuristic assigning the sorted tasks one by one, in phase 2 simulated annealing, and in phase 3 optimization on the whole problem (using a commercial ILP solver).

Smet et al. [4] used a 2-phase algorithm to solve all of Krishnamoorthy's test problems. In phase 1, they used 3 constructive heuristics, the first two assigning the sorted tasks one by one, and in the third selecting random sets of 10 or 15 workers at a time and solving each block using

Mehran Hojati
Edwards School of Business
University of Saskatchewan, Saskatoon, SK, Canada
E-mail: Hojati@Edwards.usask.ca

a commercial ILP solver. In phase 2, they used hybrid search and optimization on the whole problem (using a commercial ILP solver). They also introduced 10 more challenging new test problems.

This extended abstract of the full paper [1] also addresses large SMPTSPs, but uses a simpler method than [2].

2 ILP Formulation

Let

$J = \{j_1, j_2, \ldots, j_n\}$ = set of tasks (jobs)
$W = \{w_1, w_2, \ldots, w_m\}$ = set of workers
P_j = set of workers (personnel) that can perform task j
K_t^w = a maximal clique (set) of conflicting (overlapping) tasks for worker w at time t. A maximal clique occurs at selected task finish times, namely the first, then smallest finish time of tasks that start after the first finish time, and so on.
C^w = set of all maximal cliques of worker w
$y_w = 1$ if worker w is used, 0 otherwise
$x_{jw} = 1$ if task j is assigned to worker w, 0 otherwise

$$Min\ Z = \sum_{w \in W} y_w$$

$$s.t. \sum_{w \in P_j} x_{jw} = 1 \quad \forall j \in J,$$

$$\sum_{j \in K_t^w} x_{jw} \leq y_w \quad \forall w \in W,\ K_t^w \in C^w$$

$$y_w \in \{0,1\} \quad \forall w \in W,$$

$$x_{jw} \in \{0,1\} \quad \forall j \in J,\quad w \in W.$$

3. Greedy-based Heuristic

At each iteration, the worker with maximum objective value is chosen and all its selected tasks are assigned to him/her. Then, the process is repeated for the remaining workers and tasks. The ILP formulation of the reduced problem for worker w is as follows:

$$Max\ Z = \sum_{x_{jw} \in T_w} \left[(f_j - s_j) + \frac{h}{|P_j| - 1 + \epsilon} \right] x_{jw}$$

$$s.t. \sum_{j \in K_t^w} x_{jw} \leq 1 \quad K_t^w \in C^w$$

$$x_{jw} \in \{0,1\} \quad \forall j \in J,$$

where s_j = start time of task j, f_j = finish time of task j, T_w = set of tasks that worker w can perform, $|P_j|$ = number of remaining eligible workers for task j, h = 110, and $\epsilon = 0.1$. The bonus for each task $\frac{h}{|P_j|-1+\epsilon}$ is chosen such that as $|P_j|$ is reduced from above to small integers such as 2 or 1, the bonus becomes increasing larger, thus making the task attractive to be selected.

After the maximal cliques are determined (See Algorithm 1 of [2] for an $O(|T_w|)$ algorithm for this), the above reduced ILP problem simplifies to a longest path problem in a

directed graph with $|C^w|$+1 nodes as follows (note: this method is also mentioned on pp. 39-40 of [2]): There is a node 0; Each maximal clique, arranged chronologically, will have a node; Each task j in T_w will have an arc starting from the previous-node-to-the-first maximal clique that contains the task and ending in the last maximal clique that contains the task. Note that there are no negative cycles in this graph. To determine the longest path from node 0 to the last node, a label-correcting algorithm is used where the tasks (arcs) are first sorted by their end node (from the earliest to the latest) and the arcs are examined one at the time and the label (value) of the ending node is updated if the distance from node 0 is larger. This algorithm also runs in O($|T_w|$) .

To improve the efficiency of the computations, after any iteration a new objective value (longest path) for a worker is not determined if its solution (selected tasks) does not contain a task selected in the iteration. Even if it does, a longest path is not determined unless its current objective value is the largest among all the remaining workers.

4. Computational Results

Two available sets of test problems for SMPTSP were used: Krishnamoorthy's 137 problems (obtained from http://people.brunel.ac.uk/~mastjjb/jeb/info.html) and Smet's 10 problems (obtained from http://allserv.kahosl.be/~pieter/smptsp.html). The greedy-based heuristic was programmed in C++ and ran on a Lenovo M52 ThinkCentre desktop. The results are summarized in the following two tables.

	Greedy-based	Krishnamoorthy et al.	Lin and Ying	Smet et al.
No. of optimal solutions	107	67	72	67*
Avg. optimality gap (no. of workers)	0.6%	4.5%	1.2%	1.0%
Avg. CPU (sec)	4.1	544.3	27.6	12.3

* Avg. no. is rounded down

No. of problems Greedy used	Krishnamoorthy et al.	Lin and Ying	Smet et al.
less workers	69	61	68
equal workers	51	63	55
more workers	17	13	14

Smet's 10 test problems are more challenging. Avg. optimality gap for the greedy-based heuristic was 7.5%. However, the average CPU time was only 0.5 second.

References

1. M. Hojati. A Greedy-based Heuristic for Shift Minimization Personnel Task Scheduling Problem. European Journal of Operational Research, under review.
2. M. Krishnamoorthy, et al. Algorithms for large scale shift minimization personnel task scheduling problems. European Journal of Operational Research, 219, 34-48, 2012.
3. S-W Lin and K-C Ying. Minimizing shifts for personnel task scheduling problems: a three-phase algorithm. European Journal of Operational Research, 237, 323-334, 2014.
4. P. Smet, et al. The shift minimization personnel task scheduling problem: a new hybrid approach and computational insights. Omega, 46, 64-73, 2014.

MISTA 2015

Optimisation of a Stagger Chart for Aviation Fleet Planning

Richard Weedon • Samad Ahmadi • Mike Critchley

Abstract Within the Commercial Aviation Industry, the maintenance planning process takes into account the number of spare engines available to meet a minimum spares level (usually contractual). This is to minimize the risk of disruption to aircraft, if engines are required to be replaced due to unplanned events. To ensure the efficient use of the spare engines pool while maximizing the engine time on wing between planned removals requires a forecast that is set for the predicted life of an operator's specific aircraft type fleet. The engines are required to be refurbished at certain intervals which can be projected on to a forecast plan. The process of producing this plan by implementing the engine removals is known as Stagger. The aim of this research is to produce good quality Stagger Plans using evolutionary algorithms based upon data from an actual forecast. This paper presents our early attempts on modelling this problem and then solving it with Genetic Algorithms. Results show that the Stagger Plan produced by the GA reduced the number of weeks that the spare engines level had fallen below the minimum spare engine value when this was compared to the original forecast.

1 Introduction

Often, Operators of aircraft have various contracts with external companies such as the engine manufacturers to maintain their aircraft engine fleets. These fleets can consist of hundreds of engines. The maintenance of these engines is highly regulated to ensure maximum safety is adhered to and compliance with major air worthiness authorities. The modern commercial jet engine consists of thousands of parts which generally have specific working lives as determined by a time limits manual set by the manufacturer. These lives are also affected by how the engine is operated and in what environments. The lives are measured in hours and cycles, where a cycle in this paper is considered as a takeoff and landing. Typically these parts are affiliated with modules of the engine. The lives of certain key parts of the engine are typically tracked by using an engine health management system (EHM) and documented at shop floor visits. This information can be used in addition with the dates the engine entered into service with the operator, to predict over the life time of the fleet, a forecast of planned engine refurbishments.

To keep the aircraft operational in order to maximize revenue requires additional spare engines to account for unplanned events and also for engines that are required to come off wing due to part life time expiry or another reason as highlighted by the EHM system, for refurbishment. Typically an engine refurbishment can take between 3 – 4 months turnaround time from removal to being installed back on wing. The duration of shop visits in the Stagger Plan in this paper are between 15 and 16 weeks. If a part delivery of a fleet of aircraft to the operator occurs at a similar time then this can cause engines to require refurbishment at the same time which if not monitored can cluster and leave the spares pool empty. This could result in aircraft being grounded due to no engine availability, resulting in loss of revenue and

cancelled flights. If the engine removals are staggered, then this can reduce the risk of the number of spare engines falling below the accepted minimum value and also reduce the load on the overhaul workshops. However removing an engine off an aircraft too early than the planned removal date, results in parts being replaced which have an amount of useable life left. This is usually referenced as lost days. To monitor the operator's fleet, Forecasting and Stagger Planning tools are used but typically involve a large amount of manual processing. This paper is part of an ongoing research on this problem by authors and aims to introduce the Stagger concept and propose a basic model of the problem. A solution to the Stagger problem is developed using Evolutionary Algorithms.

2 Other studies

The maintenance of aircraft engines is a complex and expensive process that involves exhaustive fleet planning but also ensuring parts are ordered in advance (due to the complexity and associated lead times of some parts) and ready for engines that are removed. Gatland *et. al*. [1] discusses the problems that arise from the maintenance capacity planning of engines. They produce a simulation that models an engine maintenance facility that investigates effects of facility loading on turnaround time, throughput and capacity. Their paper provides further information on factors that dictate engine removals and the duration of shop visits (turnaround times). To reduce costs in the overhaul of engines accurate costing of the removal is an important part of the forecast. Engine maintenance decisions are often evaluated by using a metric commonly known as the life cycle cost (LCC).

Painter and Beachkofski [2] explain the costing implications and subsequent engine maintenance decision making by developing a simulator and data mining model to produce a more accurate LCC metric. An engine generally consists of modules. These modules are often swapped between engines to increase turnaround time at the overhaul workshop. Matching modules with similar life remaining can increase the time on wing (TOW) and reduce costs. In [3] a module swapping optimization simulator was developed for use with the air force but could equally be used in civilian engines.

A similar problem to Stagger is shown in [4] where a multi agent simulator was developed for cost reduction of engine removal scheduling. The OPS tool used an original removal plan forecast and readjusts the schedule by prioritizing engines that required overhaul due to Weibull scoring, or unforeseen circumstances. The tool does not readjust the forecast for optimization in the same way that this Stagger problem aims to provide however, it does explain the constraints such as lost days, and minimum spares. The paper also provides some mathematical representation of the turnaround times and the ratio of spare engines to fleet sizes which have not been explained in detail in this Stagger research. Additionally [4] provides information on how fleet planners/manager maintain a schedule of engine removals. Another similar problem to Stagger is also seen in [5] but for navy ship repair scheduling. Here, the paper mathematically models the constraints and fitness function to minimize the number of overlapping activities to maximize availability. The schedule is for 200 weeks with 1 maintenance cycle per ship. Instead of spare engines as a constraint, in this case 2/3 of the fleet has to be operational at any time. An evolutionary algorithm is used similarly in [5] to the Stagger solution presented in this paper.

There are software solutions available that claim to manage maintenance and cost planning such as EFPAC [6]. This software does claim to have a removal plan optimizer but does not go into detail about how this is performed. Also Clockwork Solutions [7], have a product called Insight LCM, the LSC Group [8], provide modelling and simulation solutions with optimized resource planning. SAS also provide various optimized solutions such as SAS Asset Performance Analytics [9].

Finding the optimal solution for the forecast that enables the engine to be removed with minimal lost days but maintains a minimum spare engine level can be related to timetabling problems where events can be represented as the engine removals and periods are represented

as the week numbers in the forecast subject to certain constraints which are explained later in the mathematical model shown in figure 3. There is a large amount of literature on timetabling problems and methods of solving them. Typically the approaches to solving these problems have been categorized into Sequential methods, cluster methods, constraint based methods, generalized search, hybrid evolutionary algorithms, metaheuristics, multi-criteria approaches, case based reasoning techniques, hyper-heuristics and multi-criteria approaches (for survey of approaches see [11,12]).

The study conducted in this paper, is to solve a Stagger problem that incorporates the use of an evolutionary algorithm to produce a Stagger Plan.

3 Forecasting and Stagger Planning Design

3.1 Problem Definition

The main focus of this study is to identify the best sequencing of maintenance activities based on constraints of the size of the engine spares pool and minimum contractual spares level while minimizing total lost days on wing for the fleet. A typical dataset has been produced for this problem that consisted of actual data that would normally be obtained from various information systems such as EHM data and engine shop visit forecasting data. The dataset consists of an engine number that determines the engine unique id, a start and end date of the forecast which typically shows the perceived life of the aircraft in the operator's fleet (the aircraft maybe sold to another operator at the end of the forecast or mothballed). The term mothballed is where an aircraft is kept for storage or awaiting scrap. The dataset also includes a list of all the engine removal dates that are planned for refurbishments for each engine. These refurbishments are classified as check & repair, first refurbishment, second refurbishment and mature refurbishment. There are other types of shop visits which are not relevant to this particular problem. Additionally aircraft and engine retirement dates if earlier than the end of the forecast are also included in the dataset and their induction to the Operator's fleet. The Minimum spares level for the duration of the forecast is set to 2 in this particular study.

	03/08/2015	10/08/2015	17/08/2015	24/08/2015	31/08/2015	07/09/2015	14/09/2015	21/09/2015	28/09/2015	05/10/2015	12/10/2015	19/10/2015	26/10/2015	02/11/2015	09/11/2015	16/11/2015	23/11/2015	30/11/2015
ESN	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
1046	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1070	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1071	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1072	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1073	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1074	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1110	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1111	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1113	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1114	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
AIRCRAFT_COUNT	48	48	48	48	48	50	50	50	50	50	50	50	50	50	50	50	50	50
ENGINE_DEMAND	98	98	98	98	98	102	102	102	102	102	102	102	102	102	102	102	102	102
CHECK_AND_REPAIR_COUNT	2	2	2	2	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ENGINE_COUNT	107	107	107	107	107	111	111	111	111	111	111	111	111	111	111	111	111	111
AVAILABLE_ENGINE_COUNT	104	103	103	103	106	109	109	109	109	109	108	108	108	108	108	108	108	108
SPARES_LEVEL	8	7	7	7	10	9	9	9	9	9	8	8	8	8	8	8	8	8

Fig. 1: A Non-Optimal Stagger Plan

3.2 Modelling the Stagger Problem

An example of a Stagger Plan that has not been optimized can be seen in figure 1, this shows the engines on the far left with the weekly dates for the duration of the forecast at the top. The 0 represents an engine that is currently available either as a spare or currently flying. The pink colored 1 represents engines that are currently having a shop visit. A white – 1 identifies an engine that has not been inducted into the fleet and therefore cannot be included into the calculations. Check & Repairs are not included as a main shop visit and are subsequently not recognized as a shop visit in the removal data used for the Stagger Plan of an individual engine. However, a weekly total of check & repairs has been included and is part of a calculation that affects how many engines are available as spares.

The chart in figure 1 is similar to what the final designed optimized Stagger Plan will show with an additional calculated value – lost days. The lost days and spare engines can be seen graphically for a Stagger forecast in figures 4, 5 and 6 in section 5. The calculations required to produce the minimum spares level and lost days are as follows:

Spares Available = Engine Count Available – Engine Demand. (1)

Engine Count Available = Engine Count – C&R Count - Engines in shop that week (2)

Engine Demand = Aircraft Count × no. of engines on an aircraft. (3)

Engine Count = No. of engines in shop (removal) for a particular week. (4)

Lost Days = Original engine removal week – adjusted engine removal week × 7. (5)

3.3 A Genetic algorithm for the Stagger Problem

In this paper an evolutionary algorithm is developed to produce an optimal solution for a Stagger Plan. A chromosome was represented by a 3 dimensional array. The first dimension contained all of the week numbers in the forecast, the second dimension was the engine serial numbers (ESN's) and the third dimension consisted of 9 values: ESN, start of removal week number, end of removal week number, Number of weeks removal moved forward, week ESN inducted into fleet, week ESN removed from fleet, Total Fleet spares Value, Lost Days and finally a Marker that is a Boolean data type. All of the dates have been converted to week numbers over a range from 0 to the last week of the forecast – 1.

The fitness function is required to determine the quality of each member of the population. In this study the fitness function is actually an inequality which was developed around the number of spare engines available each week and the number of lost days. The number of spare engines available should never fall below the minimum value and the function should be weighted accordingly. The lost days should be kept as minimal as possible but should not override the spare engines available. When the child is being scored the coding checks every week of the forecast and if the spare engines level drops below the minimum level then the inequality shown in equation (6) is used.

$$\frac{-Av.LostDays + 1}{50} \quad For\ Spare\ engines < Minimum\ spare\ engine\ level \tag{6}$$

Alternatively if the spare engines level is above the minimum for the whole forecast then the inequality in equation (7) is used:

$$\frac{500}{Av.LostDays + 1} \quad For\ Spare\ engines \geq Minimum\ spare\ engine\ level \tag{7}$$

These inequalities were designed so that the lost days should achieve a higher performance as the lost day's approaches 0 if the minimum spare engines level is maintained for the whole forecast. A positive fitness score would represent that the minimum spares level has been met for the whole forecast. A negative fitness function would indicate that the minimum spares figure has fallen below the threshold. How close the negative fitness score is to zero indicates that the lost days are minimal.

The crossover operation consisted of copying one part of one parents array by randomly picking an ESN and copying all of the ESN's to the left (including the randomly picked ESN) and then copying the remaining ESN's to the right from another parents array to create a child. To enable the selection of the chromosomes to represent the parents then 3 methods of selection were developed to determine if any specific method produced an improvement to the results. These selection methods are roulette wheel selection, elitism selection and a random selection. The roulette wheel selection used an array to store the proportioned fitness of all the individuals by using the formula:

$$\text{Stagger Fitness} / \text{Total Population Fitness} \times 360 \quad (8)$$

The elitism method uses an array that orders those individuals by their fitness and the highest scoring chromosomes are then selected as parents. Likewise for the random method a random number generator was used to pick parents randomly in an array. The mutation operator stage of the process was developed by calling a random number generator with the range of numbers from 1 to 2 as shown in the pseudo code in figure 2.

```
A = Engine original removal date
B = Engine Induction date
Max stagger forward (M) = 26
For each week (x) in schedule
For each original engine scheduled (O) removal in week (x)
        If A < M+1 then
          New removal(N) = Randomise(A-B)
          Elseif A >= M+1 then
                If B >= M then
                  N = A-Randomise(26)
                Else if B < M
                  N = A – Randomise(B)
Repeat for each engine
Repeat for each week
```

Fig. 2

If a 1 was generated then the mutation operation was implemented. At mutation, another random number was generated from 0 to the number of engines in the forecast – 1, to pick an engine at random. The code was developed to determine if an engine is in shop in a particular week and whether the engine was inducted into the fleet within 26 weeks of the forecast start week. Also if the removal date is less than the 26 weeks from the induction week then this needs to be accounted for. The resultant figure from the above logic is sent as a range for another random number to be generated. This random number is subtracted from the original removal week to produce a new removal week. Finally the weakest (fitness) chromosome was erased from the population after each chromosome is spawned.

The software used to develop the optimized Stagger Plan was Microsoft Excel VBA 2010. The structure of the coding involved the use of 2 and 3 dimensional arrays. As mentioned earlier, an array was used to store the data that was imported from the worksheets which is referenced in the Setup worksheet with the cell ranges that the data lies within. Another array was created that produced a generic Stagger Plan with the original engine removals and the week number of the forecast. A third dimension was used to switch between removal start and end dates, engine induction dates, adjusted removal date, engine retired from fleet date, lost days and fleet spare engines. This array was used as a template and was copied into each

chromosome at the initialize population stage of the fleet prior to mutation. At initialization of the population stage no crossover operation was used due to no parents. To create a chromosome a class module was used to create a collection called *Staggers.*

A Stagger is a member (property) of the Staggers collection which was used to store the chromosome. Each Stagger had a property associated with it. These include a variant type array which was a version of Stagger Plan, a generation property to store the generation number that it was spawned in and a name property. After each Stagger was created another array was used to work out its fitness and store the result. This array *varXScore* was used to compare all the other active chromosomes in the population to identify which one is erased. A temporary array was used to sort the highest score in descending order, from the *varXScore* array when this process was initiated.

Once the population was initialized then subsequent generations were created using a loop until the required number of generations was reached as set on the Setup worksheet. At the Mutation stage, if it was selected by the random generator the mutation was set by using the original removal date and randomly moving the removal date forward by the criteria discussed earlier.

When the last generation was created the Stagger Plan from the chromosome with the best fitness was exported into a worksheet called Final. The scores of all of the chromosomes can be seen in the Score worksheet.

4 Results

4.1 Initial Tests

The test runs were produced initially with the population set to 4 and the number of generations set at 3. The number of parents per generation for this study was always set to 2. These settings were used initially to test the duration of the run and whether any major performance issues would be encountered. The table below shows the fittest chromosome from each run for both the roulette wheel selection and elitism selection methods used for the selection of parents:

Run 1						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-3	14.28	-0.31	6	3	00:00:36	Elitism
-3	15.04	-0.32	7	4	00:00:33	Elitism
-3	15.85	-0.34	6	3	00:00:34	Elitism
-2	15.90	-0.34	7	4	00:00:37	Roulette
-3	14.30	-0.31	5	2	00:00:36	Roulette
-3	14.35	-0.31	5	2	00:00:34	Roulette

Due to the very small population and number of generations created, there was an improvement to the Stagger Plan. A positive fitness score would represent that the minimum spares level has been met for the whole forecast. A negative fitness would indicate that the minimum spares figure has fallen below the threshold. How close the negative fitness score is to zero indicates that the lost days are minimal. In fact the average lost days for all removals in the Stagger is around 14 days in the above table. All the parents that are created at the initialization stage contain an average lost day's figure > 250 which is suggesting there is a problem with the calculation of the creation of the parents as the maximum Stagger should only be 180 days.

However the figure settles with the next generations. Due to time constraints of this study this problem was not investigated further. Changing the population to 20 with 10 generations did increase the processing time from around 35 seconds to 105 seconds as shown in the table below.

Run 2						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-1	13.89	-0.30	28	9	00:01:44	Roulette
-2	13.75	-0.29	23	4	00:01:39	Elitism
-3	14.28	-0.31	24	5	00:01:28	Roulette
-2	13.75	-0.29	23	4	00:01:40	Elitism

As can be seen in the above tables increasing the population size and the number of generations has not improved the results too much at this point. Increasing the generations to 25 with a population of 20 increased the processing time to approximately 225 seconds. The best fitness achieved for these runs only improved by +0.01 and the average lost days decreased to 13.04.

Run 3						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-2	13.49	-0.29	45	26	00:03:47	Elitism
-3	13.04	-0.28	41	22	00:03:52	Roulette

It can be seen through the above tables that there is no real difference between the elitism and roulette wheel selection methods at this stage. Increasing the generations to 75 again showed an improvement with the fitness with respect to the lost days being reduced to 11.6 days for the whole Stagger Plan. The processing time was 13 minutes. However the minimum spares value is not improving for this run because the best chromosome still had – 2. The fitness function is improving the lost days consistently with the increase in the number of generations.

4.2 Improving the Fitness Function

An amendment was made to the inequality of equation (6) to give:

$$\frac{-Av.\,LostDays + 1}{50} + 2 \times (No.\,of\ Spare\ engines) \tag{9}$$

This was designed to provide more weight to the minimum spares level to ensure it would add more bias to the spare engines rather than the lost days. Using a population of 20 and 10 generations provided the following results as shown in run 4.

Run 4						
Min Spares	**Av. Lost days**	**Fitness score**	**Id**	**Gen.**	**Time**	**Crossover**
-2	13.52	-0.29	22	3	00:01:42	Roulette
-3	13.54	-0.29	22	3	00:01:44	Elitism

Run 5								
Min Spares	**Av. Lost days**	**Fitness score**	**Chromosome**	**Gen.**	**Duration (min)**	**Crossover**	**p**	**g**
-1	11.97	-2.26	50	41	00:07:12	roulette	10	50
0	12.78	-0.28	54	45	00:07:07	elitism	10	50
-1	16.56	-2.35	7	4	00:00:34	roulette	4	3
-2	15.31	-4.33	6	3	00:00:33	elitism	4	3
0	15.66	-0.33	21	2	00:01:38	roulette	20	10
-2	14.61	-4.31	29	10	00:01:40	elitism	20	10
-1	15.89	-2.34	38	19	00:03:42	roulette	20	25
0	13.09	-0.28	45	26	00:03:44	elitism	20	25
-3	14.80	-0.32	7	4	00:00:38	roulette	4	3
-1	13.46	-0.29	5	2	00:00:33	elitism	4	3
-2	14.61	-0.31	22	3	00:01:39	roulette	20	10
-2	13.59	-0.29	30	11	00:01:40	elitism	20	10
-3	14.09	-0.30	30	11	00:03:42	roulette	20	25
-2	11.17	-0.24	45	26	00:03:45	elitism	20	25
-2	14.11	-0.30	57	48	00:07:05	roulette	10	50
-1	11.25	-0.24	51	42	00:07:09	elitism	10	50

In run 5, the p represents the population size and g represents the number of generations. The colored section represents the fitness inequality in equation (8) and the uncolored section represents the fitness inequality in equation (7). Each of the results displayed in the above table are those chromosomes who have the best fitness function of that particular test. Interestingly, the elitism selection for the fitness inequality in equation (7) outperforms the roulette wheel selection for fitness and lost days and the weekly remaining spare engines. For the revised fitness function, the operators are similar. Comparing the spare engines remaining between the 2 fitness inequalities clearly show that the fitness inequality in equation (8) produces a better spare engine remaining figure and appears to consistently improve the spare engine figure as the generations are spawned.

A further elitism selection run produced a minimum spare level of 1 which was using a population size of 10 and 40 generations. The fitness was 1.64 and the average lost days for each week of the forecast was 16.85 days. The lost days can be seen in a plot as shown in figure 3 and the minimum spare engines can be seen in figure 4 below.

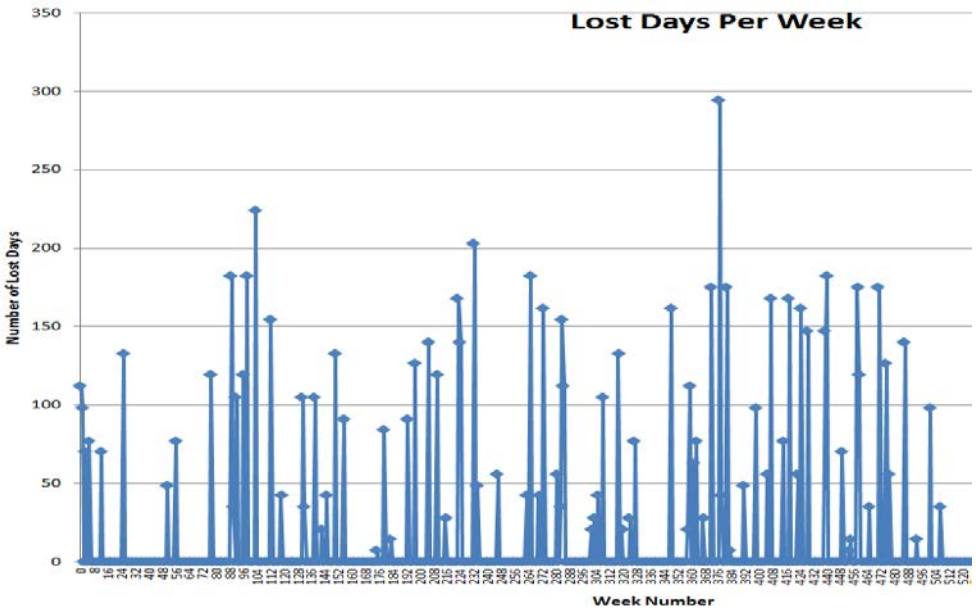


Fig.3

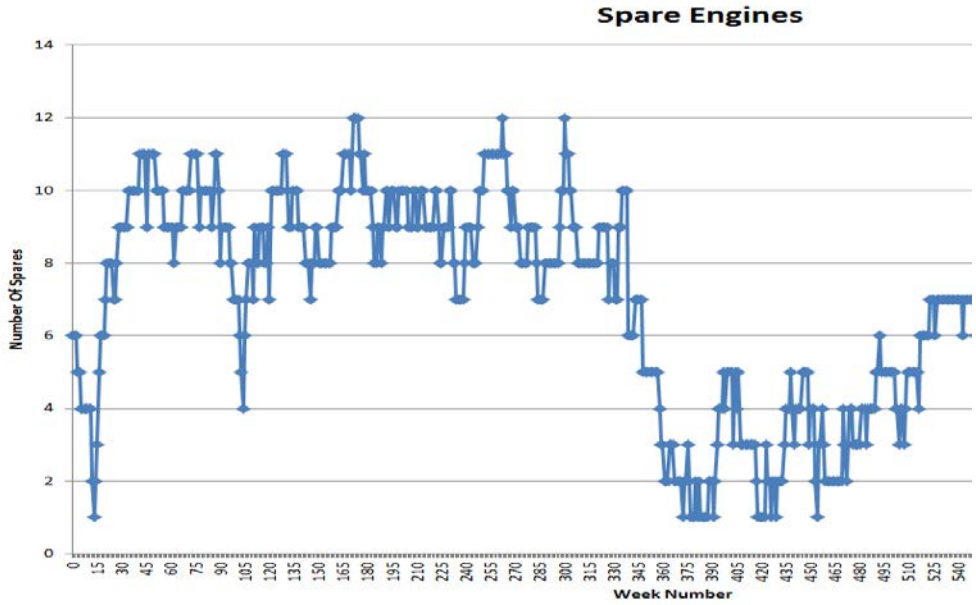


Fig.4

The Stagger Plan produced above did fall below the minimum spares level of 2 however, the total lost days across all weeks in the forecast was 8372 days. Although this figure seems high but the lost days per engine (114 in this fleet) are 73.4. Also if the average daily cycles flown per engine is 2.2 then the total number of cycles lost is 18418.4. The lost cycles per engine are then 161.6. Typically an engine is taken off wing at approximately 50 cycles prior to the latest possible removal date to provide a safety margin.

The results have shown that the evolutionary algorithm overall improved the optimization of Stagger. The results from tables 1 to 4 show an improvement in the fitness score for lost days. However, the minimum spares level was completely random from –1 to -3 for the majority of the test runs above. Improving the fitness function to that of equation 8 did not improve the fitness scores until a code change to the mutation operator was addressed. After this amendment

the fitness scores did improve. The elitism selection and the roulette wheel selection did not seem to be too different for both fitness inequalities. Run 3 shows the roulette wheel selection produced a better fitness score and lost day values where run 2 produced better elitism selection results. The final run produced better results that were an improvement from the standard Stagger Plan that used the forecasted removal dates with – 4 spare engines as the lowest value of spares as shown in figure 5 below:

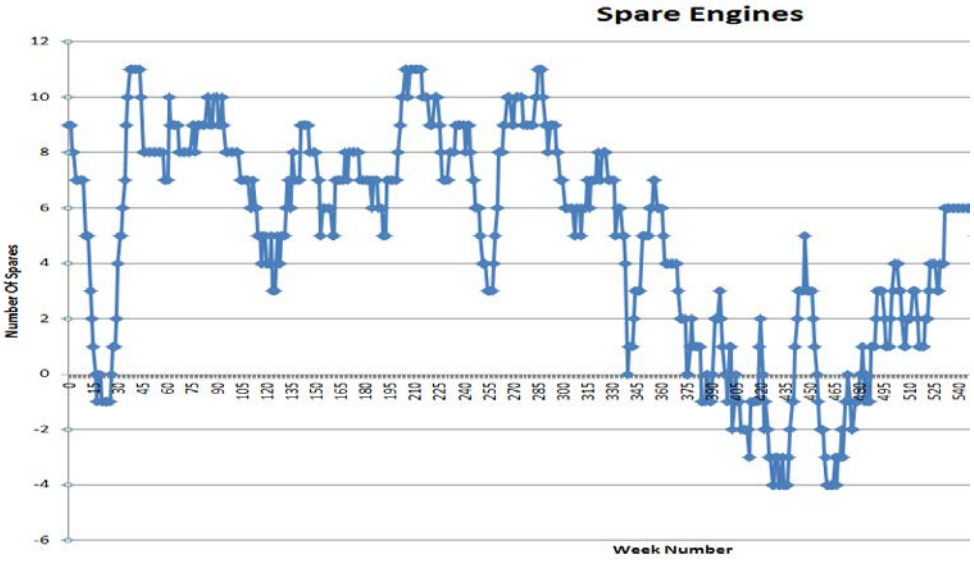


Fig.5

The majority of the runs above were timed to determine that the processing is consistent for each run of similar settings and also to evaluate how much time a large population and number of generations would take to complete, providing the system could handle the memory and processor resources required. The times were fairly consistent with the runs. The computer that performed the test runs was running windows 7, 32-bit operating system with 4GB ram and an AMD Phenom II quad core processor – 3.3GHz. Typically performing a run used approximately 25% on the CPU and increased the RAM usage by 0.06GB consistently, providing no other applications were used in addition to Microsoft Excel. These values include all the other system processes that are running in the background.

None of the above runs produced an optimal solution that satisfied the condition that no spare engines would fall below the minimum value. However, if the population was increased to 100 and left for 10000 generations, then this may produce a better solution. This figure would create a total of 10,100 members. To create a population with 10 members over 50 generations (60 in total) involved a processing time of approximately 7 minutes which if used as a guide for 10,100 members would take 19.64 hours to generate. The best result was obtained with a population of 50 to a 100 with 40,000,000 iterations (generations). Unfortunately using that many generations with the existing set up would not be feasible.

5 Conclusion

In this paper the Stagger problem has been introduced with a basic solution developed that uses an evolutionary algorithm. The Stagger problem could be expanded to cover Life Separation. This is where some engines are taken off wing as they are delivered to the fleet and replaced with a spare engine that has half of the life of the new engine. The engine that has been taken off is typically stored as new for a length of time to allow a break between future removals and subsequently reduce demand in the future as all of the engines need to be refurbished. However, there are a limited number of spare engines for a fleet to use as can be seen in figures 7 and 8 and where a high volume of shop visits occur these spare engines may be required. This extra feature of Stagger could be investigated in future work.

Future work is also planned to present a Mathematical model and also more accurate and sophisticated algorithms to find optimal solutions.

References

1. Gatland, R.; Yang, E.; Buxton, K., “Solving engine maintenance capacity problems with simulation”, Simulation Conference, 1997, IEEE, pages 892-899, 7-10 Dec. 1997
2. Painter, M.K.; Erraguntla, M.; Hogg, G.L.; Beachkofski, B., "Using Simulation, Data Mining, and Knowledge Discovery Techniques for Optimized Aircraft Engine Fleet Management," Proceedings of the Winter Simulation Conference, 2006. IEEE, pages 1253,1260, 3-6 Dec. 2006
3. Yutsung W; Jaw, L.; Rendek, P.; Moses, E.; Robinson, M.; Driver, S.; Senior, K., "Demonstration of A Reliability Centered Maintenance (RCM) Tool to Extend Engine's Time-On-Wing (TOW)," Aerospace Conference, 2007, IEEE, Pages 1,5, 3-10 March 2007
4. Stranjak, A.; Dutta, P.S.; Ebden, M.;Rogers, A.;Vytelingum, P.
5. “A multi-agent simulation system for prediction and scheduling of aero engine overhaul”, 2006, Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (Industrial Track), ACM DL, Industrial Track, 8-12 May 2006
6. Deris, S.; Omatu S.; Ohta H.; Kutar S.; Samat P. A.; “Ship maintenance scheduling by genetic algorithm and constraint-based reasoning”, European Journal of Operational Research, 1999, Volume 112, Issue 3, 1 February 1999, Pages 489-502
7. Aerdata, 2015, EFPAC Engine Maintenance Cost Planning [Online] Available from http://www.aerdata.com/efpac-engine-management-system.html, [Accessed 14/05/15]
8. LSC Group, 2015, Modelling & Simulation [online] Available from: http://www.lsc.co.uk/our_services/ikm__business_analysis/modelling__simulation/LSC Group, [Accessed 03/06/2015]
9. Clockwork Solutions, 2015, Insight LCM, [online] Available from: http://clockwork-solutions.com/products/insight-lcm/, [Accessed 03/06/2015]
10. SAS, 2015, SAS Asset Performance Analytics, [online] Available from: http://www.sas.com/en_us/software/supply-chain/asset-performance-analytics.html, [Accessed 03/06/2015]
11. Babaei, H, Karimpour, J. and Hadidi, A., A survey of approaches for university course timetabling problem, Computers & Industrial Engineering, Available online 21 November 2014, ISSN 0360-8352, http://dx.doi.org/10.1016/j.cie.2014.11.010.
12. Burke, E.K. and Petrovic, P., Recent research directions in automated timetabling, European Journal of Operational Research, Volume 140, Issue 2, 16 July 2002, Pages 266-280, ISSN 0377-2217, http://dx.doi.org/10.1016/S0377-2217(02)00069-3.

Author Index