# TORSCHE SCHEDULING TOOLBOX: LIST SCHEDULING

**STIBOR MILOSLAV, KUTIL MICHAL**

Department of Control Engineering,
Czech Technical University in Prague,
Technicka 2, 166 27 Prague 6, Czech Republic
phone: +420 224 355 711, fax: +420 224 355 703
{stibom1,kutilm}@fel.cvut.cz

**Abstract:** TORSCHE is a scheduling toolbox for MATLAB environment which has been developed at the Department of Control Engineering (Czech Technical University in Prague, Faculty of Electrical Engineering) and is distributed under the terms of the GNU General Public License. It's designed to solve various problems of scheduling and validation by proper algorithms whose number is still extending. As a basic combinatorial algorithm, List Scheduling has been well known for almost 50 years. In this algorithm, tasks are fed from a pre-specified list. The available first task on the list is scheduled and removed from the list. There are many heuristics algorithms like The Earliest Starting Time first (EST), The Earliest Completion Time first (ECT), The Longest Processing Time first (LPT) and etc., based on simple ordering of tasks in the list by various parameters. These algorithms are needed in a wide range of practical problems in industry, logistics, informatics and etc.

**Keywords:** Scheduling, Matlab, List Scheduling

## 1 INTRODUCTION

Scheduling theory has been a popular discipline for a last couple of years. However, there is no tool, which can be used for a complex scheduling algorithms design and validation. Creation of this tool is our goal. TORSCHE (Time Optimization of Resources, SCHEduling) is a scheduling toolbox for MATLAB environment intended for education and rapid prototyping. It contains complex set of algorithms and utilities for classical scheduling problems, cyclic scheduling, scheduling with positive and negative time lags, graph theory and more. TORSCHE is distributed under the terms of the GNU General Public License and number of included algorithms is still extending.

## 2 SCHEDULING TOOLBOX BASICS

Main objects of TORSCHE Scheduling Toolbox are Task, TaskSet and Problem. Object Task is a data structure containing all parameters of the task as process time, release date, deadline etc. Objects of a type Task can be grouped into a set of tasks and other related information as precedence constrains can be added. Object Problem is a small structure describing classification of deterministic scheduling problems in Graham and Baewicz notation. These objects are used as a kernel providing general functions and graphical interface, making the toolbox easily extensible by other scheduling algorithms.

### 2.1 Task

Task is a basic object in scheduling problems, which represents any unit of work or process that is scheduled by appropriate algorithm and executed by the given system. The graphic representation of task and its basic parameters are shown on the Figure 1, where

**processing Time (ProcTime, $p_j$)** is an execution time of the task on the procesor;

**release Time (ReleaseTime, $r_j$)** is a time at which a task becomes ready for execution;

**deadline (DeadLine, $\tilde{d}_j$)** is a time limit by which the task has to be necessarily completed, otherwise the schedule is unsuccessful;

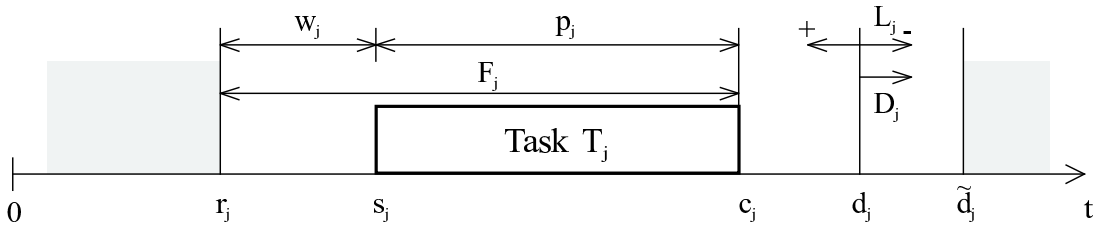


Figure 1 – Taks parameters

**due date (DueDate, $d_j$)** is a time limit by which the task should be completed, otherwise the criterion function is charged by penalty;

**completion time ($c_j$)** is a real time of completion of the task;

**starting time ($s_j$)** is a real time at which the task execution begins;

**waiting time ($w_j$)** is a delay between release time and starting time of the task;

**flow time ($F_j$)** is a time interval from releasing to completion of the task;

**lateness ($L_j$)** is a time interval between due date and completion time of the task in the positive and the negative sense;

**tardiness ($D_j$)** is a time interval between due date and completion time of the task only in the negative sense.

Task can be also described by parameters like:

**weight (Weight)** number which defines the priority of the task in relation with other tasks (it often be represented by $w_j$ symbol which is the similar symbol for waiting time);

**processor (Processor)** defines processor whereon the task have to be executed.

In the TORSCHE Scheduling Toolbox a task is represented by the object data structure with the name task. Constructor for this object is command with the following syntax:

```
t1 = task([Name,]ProcTime[,ReleaseTime[,Deadline[,DueDate[,Weight
          [,Processor]]]]])
```

Input parameters of task constructor fully corresponding with terms mentioned above. Parameter Name is variable of string data type, which represents name of the task. ProcTime parameter is only one required and the others are optional. For unrelated processors the ProcTime parameter is a vector where each number represents processing time on concrete processor. Any parameter of the task can be modified or define lately by common dot-syntax as it is used to do. Another approach to read or modify parameters is through common routines get() and set().

### 2.2 Taskset

Objects of the type task can be grouped into a set of tasks. A set of tasks is an object of the type taskset which can be created by the command taskset. Syntax for this command is as follows

```
T = taskset(tasks[,prec])
```

where variable tasks is an array of task objects or a vector of task processing times. Vector of task processing times define the taskset directly without previous task objects definition. Variable prec is a square matrix containing precedence constrains between tasks. An example of taskset definition by the task processing time vector is given by the following code, which creates the taskset variable T with five tasks.

```
>> T = taskset([5 2 6 3 8], prec);
```

### 2.3 Problem

The object problem is a small structure describing the classification of deterministic scheduling problems in the notation proposed by Graham et al. [1979] and Błażewicz et al. [1983]. An example is given by the following code.

```
p = problem('P|prec|Cmax')
```

This notation consists of three parts ($\alpha|\beta|\gamma$). The first term (alpha) describes the processor environment, the second term (beta) describes the task characteristics of the scheduling problem as the precedence constrains, or the release time. The last term (gamma) denotes an optimality criterion.

## 3 LIST SCHEDULING

List Scheduling is a basic and popular combinatorial algorithm intended for scheduling of set of tasks on a single and even parallel processors. In this algorithm tasks are fed from a pre-specified list and, whenever a processor becomes idle, the first available task on the list is scheduled and removed from the list, where the availability of a task means that the task has been released and, if there are precedence constraints, all its predecessors have already been processed Leung [2004]. The algorithm terminates when all tasks from the list are scheduled. Its time complexity is O(n). In multiprocessor case, the processor with minimal time is taken in every iteration of algorithm and the others are relaxed.

The fact, which is obvious from the principle of algorithm is that, there aren't any requirements of knowledge about past or future of content of the list. Therefore, this algorithm is capable to solve offline as well as online no-clairvoyance scheduling problems. There are many heuristics algorithms like The Earliest Starting Time first (EST), The Earliest Completion Time first (ECT), The Longest Processing Time first (LPT) and etc., based on simple ordering of tasks in the list by various parameters.

## 4 HEURISTICS BASED ON LIST SCHEDULING

Heuristic algorithms tend toward but do not guarantee to find optimal solutions for any instance of an optimization problem. On condition of appropriate choose of heuristic it often provide acceptable results with very good time and memory complexity.

**The Earliest Starting Time first rule:**
Reorder tasks in the list to no-decreasing tend of starting time $s_i$ before the application of List Scheduling algorithm. Its time complexity is $n\log n$.

**The Earliest Completion Time first rule:**
Reorder tasks in the list to no-decreasing tend of completion time $c_i$ in every iteration of List Scheduling algorithm. Where completion time is computed as

$$c_i = max\left(r_i, t_{proc}\right) + p_i, \tag{1}$$

and $r_i$ is release time, $p_i$ is processing time and $t_{proc}$ is a minimal actual time on processors.

**The Longest Processing Time first rule:**
Reorder tasks in the list to no-increasing tend of processing time $p_i$ before the application of List Scheduling algorithm. Its time complexity is $n\log n$.

**The Shortest Processing Time first rule:**
Reorder tasks in the list to no-decreasing tend of processing time $p_i$ before the application of List Scheduling algorithm. Its time complexity is $n\log n$.

## 5 IMPLEMENTATION

List Scheduling is implemented in TORSCHE Scheduling Toolbox by function `listsch` which also allows to user to use any of implemented heuristic algorithms and visualize process of scheduling step by step in text form in MABLAB workspace. Moreover, the last version is able to solve scheduling problems on unrelated parallel processors. Figure 2 shows a flowchart of `listsch` function.

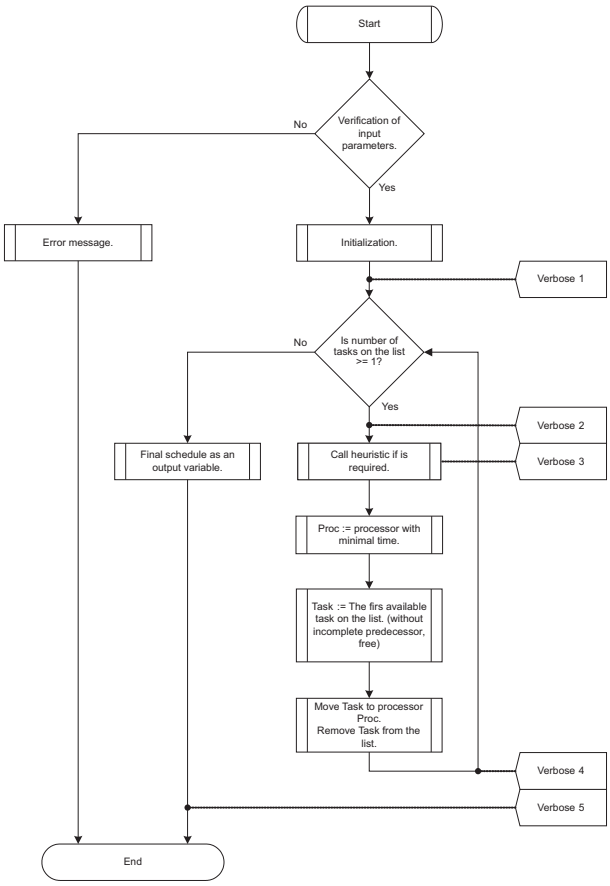


Figure 2 – Flow chart of listsch function

The syntax is given by following code

```
taskset = listsch(taskset,problem,processors [,heuristic] [,verbose])
```

or

```
taskset = listsch(taskset,problem,processors [,options]).
```

Where

**taskset (taskset)** is a set of task;

**problem (problem)** is an object problem;

**processors** is a number of processors;

**heuristic** is an algorithm like LPT, SPT, EST;

**verbose** is a level of verbosity;

**options** is a global variables of Scheduling Toolbox.

# 6 CASE STUDIES

## 6.1 Conveyor Belts

Transportation of goods by two conveyor-belts is simple example of using List Scheduling in practice. Construction material must be carried out from place to place with minimal time effort. Transported articles represent five kinds of construction material and two conveyor-belts as processors are available. Table 1 shows assignment of this problem. Solution of the case study is shown in five steps:

Table 1 – Material transport processing time.

| **Name** | sand | grit | wood | bricks | cement |
|---|---|---|---|---|---|
| **Processing Time** | 40 | 50 | 30 | 50 | 20 |

1. Create taskset directly through vector of processing time.

```
>> T = taskset([40 50 30 50 20]);
```

2. Since the taskset has been created, is possible to change parameters of all tasks in it.

```
>> T.Name = {'sand','grit','wood','bricks','cement'};
```

3. Define the problem, which will be solved.

```
>> p = problem('P|prec|Cmax');
```

4. Call List Scheduling algorithm with taskset and problem created recently and define number of processors (Conveyor-belts).

```
>> S = listsch(T,p,2)
Set of 5 tasks
 There is schedule: List Scheduling
```

5. Visualize the final schedule by standard plot function, see Figure 3.

```
>> plot(S)
```

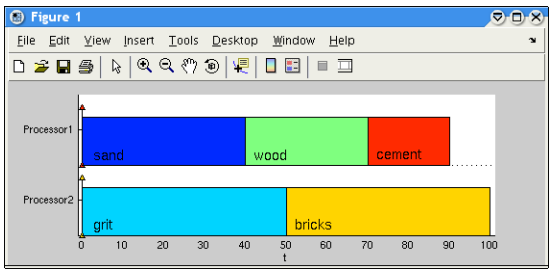


Figure 3 – Result of case study 1 as Gantt chart

### 6.2 Chair manufacturing

This example is slightly more difficult and demonstrates some of advanced possibilities of Toolbox. It solves a problem of manufacturing of a chair by two workers (cabinetmakers). Their goal is make four legs, seat and backrest of the chair and assembly all of these parts with minimal time effort. Material, which is needed to create backrest, will be available after 20 time units of start and assemblage is divided out into two stages. Figure 4 shows the mentioned problem by graph representation.

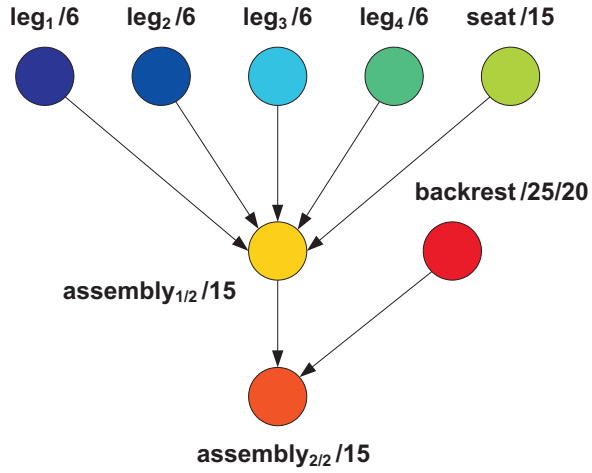


Figure 4 – Graph representation of Chair manufacturing

1. Create desired tasks.

```
>> t1 = task('leg1',6)
Task "leg1"
 Processing time: 6
 Release time:    0

>> t2 = task('leg2',6);
>> t3 = task('leg3',6);
>> t4 = task('leg4',6);
>> t5 = task('seat',6);
>> t6 = task('backrest',25,20);
>> t7 = task('assembly1/2',15);
>> t8 = task('assembly2/2',15);
```

2. Define precedence constraints by precedence matrix prec. Matrix has size $n \times n$ where $n$ is a number of tasks.

```
>> prec = [0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 0 0 1;...
           0 0 0 0 0 0 0 1;...
           0 0 0 0 0 0 0 0];
```

3. Create an object of taskset from recently defined objects.

```
>> T = taskset([t1 t2 t3 t4 t5 t6 t7 t8],prec)
Set of 8 tasks
 There are precedence constraints
```

4. Define solved problem.

```
>> p = problem('P|prec|Cmax')
P|prec|Cmax
```

5. Call List Scheduling algorithm with taskset and problem created recently and define number of processors and desired heuristic.

```
>> S = listsch(T,p,2,'SPT')
Set of 8 tasks
 There are precedence constraints
 There is schedule: List Scheduling
   Solving time: 1.1316s
```

6. Visualize the final schedule by standard plot function, see Figure 5.

```
>> plot(S)
```

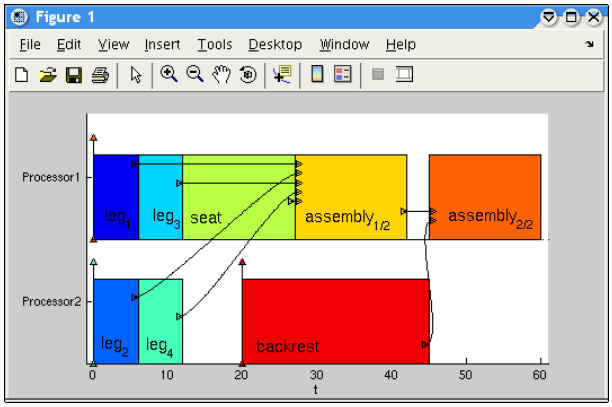


Figure 5 – Result of case study 2 as Gantt chart

## 7 SUMMARY AND FUTURE WORK

This paper presents TORSHE Scheduling toolbox for MATLAB, grounding of its usage and a couple of simple examples inspired by practice problems. This toolbox is a spreading set of algorithms, utilities, interfaces and applications intended for solving time optimization of resources by a quick and simple way. It provides a possibility to anyone to realize, try and verify solution of problem in wide range of real-world projects. In the future work we will focused on incorporating new algorithms and improving connections to another tools and projects.

This work was supported by the Ministry of Education of the Czech Republic under Project 1M0567.

**References**

BŁAŻEWICZ, J.; LENSTRA, J. K.; KAN, A. H. R. 1983. Scheduling subject to resource constrains: classification and complexity. *Ann. Discrete Math.*, 5, 11–24.

GRAHAM, R.; LAWLER, E.; LENSTRA, J.; KAN, A. R. 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Ann. Discrete Math.*, 5, 287–326.

LEUNG, J. Y.-T., editor 2004. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press.