A Low Time-Consuming Rosters Evaluation in Personnel Scheduling Problems Based on Pattern Learning

Roman Vaclavik, Premysl Sucha, Zdenek Hanzalek*

Abstract
Numerous (meta-)heuristics for solving personnel scheduling problems have been introduced in various papers over the last few years. In most cases, these methods consist of two usual steps: 1) generate new solutions and 2) determine their quality (cost) given by an objective function which is very computationally expensive. Our paper proposes a faster evaluation of the objective function based on the solution structure (pattern). The inspiration was found in creating a roster by a human, who is able to recognize an obviously bad roster using their own experience instead of complex computing. For this purpose, a neural network is used as a tool of pattern recognition to distinguish between good and bad solutions. The given approach is applied to the standard benchmark instances for the nurse rostering problem. We demonstrate that the proposed classifier can reduce the runtime of the scheduling algorithm in comparison with standard cost-oriented evaluation of the objective function with equivalent solution quality.

Keywords: tabu search, personnel scheduling, nurse rostering problem, neural network, pattern learning.

1 Introduction

Personnel scheduling, namely the Nurse Rostering Problem (NRP), is a famous combinatorial problem, which is known to be NP-hard [1, 7]. There are numerous papers dealing with algorithms, both exact or heuristic, for solving NRP [3, 4]. Good rosters can save a significant amount of company money as well as improve employee satisfaction with their workload. The focus of the article is on the use of a human-inspired solution quality determination. The basic idea is a quick recognition of an obviously bad solution structure, which is a relatively easy task for an experienced human scheduler. The main motivation is to speed up execution of the scheduling algorithm where a significant portion of the time is used for the evaluation of the potential solutions.

The first papers dealing with the applications of learning techniques to the scheduling problems appeared in the 90's. A search control policy for the resource-constrained scheduling problems was described in [12]. A neural network is used as an approximator for a future value of the resource dilation factor in the next iteration of a scheduling algorithm. For each possible neighbor (i.e. all available changes from the current solution), the result from the neural network is combined with the immediate reward value and the solution with the best evaluation is chosen. This approach, which was verified on the NASA space shuttle payload processing problem, outperformed the best known non-learning search algorithm at that time.

A Neural network and logistic regression are used in [8] to determine the right order of execution of low-level heuristics in a hyper-heuristic method. These techniques work as classifiers and decide whether the execution order of low-level heuristics is good or not. The experiments on the exam timetabling problems showed that both described methods can speed up search of the algorithm significantly.

Li et al. [9] also introduced recommendations in the area of pattern recognition usage for an evaluation of the solutions quality. The neural network serves to distinguish between good and bad solutions based on the structure of the whole solution. Unfortunately, the paper does not mention clearly how to replace a cost-oriented objective function evaluation by a neural network classifier in a scheduling algorithm. The authors use the best known objective function values from the literature to determine whether a roster is good or bad which is unrealistic since they use the historical result to find the new result. Their proposed classification is also problematic at the beginning of the rostering algorithm search because almost all inspected solutions are bad from the classifier point of view and it is hard to decide which one is better. Moreover, the authors used a rostering algorithm to obtain a set of intermediate solutions. They split this set into two subsets, while the first one was used for the classifier training and second one was used to determine the performance of their approach (see Table 2 and 3 in that paper). Furthermore, the results in the experiment part demonstrate only heuristic runtime reduction, but not the quality of solutions. Finally, taking into account that the number of employees can vary for each scheduling period, a different clas-

*The authors are with the Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic (email: {vaclarom,suchap,hanzalek}@fel.cvut.cz) sifier must be learnt for all periods and this fact degrades applicability of their approach.

In general, neural networks are proven to be a suitable tool for pattern recognition [2, 10]. The main reason for that is their ability to mimic the human mind [11]. They are able to learn from experience and can also deal with noise in the input data almost as well as human beings.

The contribution of this work is a design of classifier which significantly reduces the use of a time consuming cost-oriented objective function evaluation. The classifier estimates whether a single change in the roster of one employee (nurse) leads to the objective function improvement or not. Compare to [9], who classify the whole roster even though it is not necessary, our approach results in a faster and more accurate neural network learning. In our case, the objective function value of the optimal (or near optimal) solution is not needed. The resulting classifier can also handle with the different number of employees for each scheduling period, i.e. without additional learning to new data. Furthermore, we demonstrate a possible usage of the trained classifier on unknown data from the same or similar instances which is not dealt with in the above cited paper.

The rest of the paper is organized as follows: at first, the nurse rostering problem is described. Then, we outline usage of pattern recognition for the roster evaluation and we describe a practical realization. After that, we discuss the results of our approach on the standard benchmark instances. Finally, we conclude this work.

2 Problem statement

In the case of the Nurse Rostering Problem, which is a specific task of the staff assignment problem, activities are shifts that have defined a start time, duration and a finish time. The resources are nurses to whom no more than one shift is assigned each day only if they have the necessary qualification for the shift [3, 4].

The problem is parameterized by the number of nurses n, the number of days during the planning period d and the number of shifts s. Then the roster R is a binary matrix such that $\forall i \in \{1 \dots n\}, \forall j \in \{1 \dots d\}, \forall k \in \{1 \dots s\}$

$$R_{ijk} = \begin{cases} 1 & \text{shift } k \text{ is assigned to nurse } i \text{ on day } j, \\ 0 & \text{otherwise.} \end{cases}$$

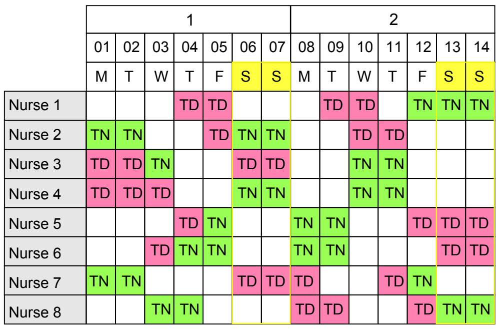
The quality of roster R is given by the objective function Z which is defined as

$$Z(R) = \sum_{i=1}^{n} Z_i(R),$$

where Z_i is the quality of the assignment related to nurse i.

For example, Figure 1 illustrates a simple roster for 14 days. It is a standard benchmark instance from http://www.cs.nott.ac.uk/~tec/NRP/. The task is to assign two shift types (i.e. TD = day shift and TN = night shift) to eight nurses based on the desired coverage of shifts. In this case, two TD shifts and two TN shifts have to be scheduled every day.

During the gradual allocation of shifts into the roster, the various constraints have to be taken into account. They may refer to any roster dimensions (days and nurses) or shifts themselves. There are two types of constraints: hard and soft. Hard constraints should never be violated and have no effect on the value of the objective function Z. If the roster satisfies all hard constraints, it is a feasible solution of the problem. Otherwise, it is an infeasible roster. In contrast to hard constraints, soft constraints may be violated. Our goal, however, is to achieve the minimum number of violations, with the need to take into account the cost (weight) of a violation which is reflected in the objective function Z. Scheduling then solves an optimization problem, where either the minimum or maximum value of the objective function Z is searched.



	1							2						
	01	02	03	04	05	06	07	08	09	10	11	12	13	14
	M	T	W	T	F	S	S	M	T	W	T	F	S	S
Nurse 1				TD	TD				TD	TD		TN	TN	TN
Nurse 2	TN	TN			TD	TN	TN			TD	TD			
Nurse 3	TD	TD	TN			TD	TD			TN	TN			
Nurse 4	TD	TD	TD			TN	TN			TN	TN			
Nurse 5				TD	TN			TN	TN			TD	TD	TD
Nurse 6			TD	TN	TN			TN	TN				TD	TD
Nurse 7	TN	TN				TD	TD	TD			TD	TN		
Nurse 8			TN	TN				TD	TD			TD	TN	TN

Figure 1: An example of the roster (Millar instance): 8 nurses, planning period of 14 days and 2 shift types (day shift (TD): 06:00-18:00, and night shift (TN): 18:00-06:00)

3 Pattern recognition for roster evaluation

One of the successful meta-heuristics for NRP is Tabu Search (TS) which belongs to the class of local search algorithms [6]. These algorithms try to choose a new solution from the neighbors of the actual one. The algorithms keep the best solution and continue until some stopping criterion is met (e.g. a time limit or no improvement after several steps). In general, the problem of local search algorithms is their tendency to get stuck in a local extreme. TS partially resolves this issue by using an adaptive memory, so called Tabu List (TL), that stores already visited solutions in the recent past. If a new potential solution is in the TL then TS skips it and tries to find another one which is not in the TL.

```
Require: initial solution $R$
1: $R_{best} = R$
2: $best\_Z = Z(R_{best})$
3: while stopping criterion is not met do
4:     select employee $emp_a$ to improvement
5:     if $emp_a = null$ then
6:         break
7:     end if
8:     for $\forall cand \in neighborhood(emp_a, R_{best})$ do
9:         if $cand \in tabu\_list$ then
10:            continue
11:        end if
12:        $R_{tmp}$ = apply changes from $cand$
13:        $actual\_Z = best\_Z + (Z_{cand.emp_a}(R_{tmp}) - Z_{cand.emp_a}(R_{best})) + (Z_{cand.emp_b}(R_{tmp}) - Z_{cand.emp_b}(R_{best}))$
14:        if $actual\_Z < best\_Z$ then
15:            $R_{best} = R_{tmp}$
16:            $best\_Z = actual\_Z$
17:        end if
18:    end for
19: end while
20: return $R_{best}$
```

Figure 2: Pseudo-code of the simple Tabu Search with the cost-oriented objective function evaluation

Figure 2 shows the pseudo-code of the simple Tabu Search. The variable *cand* represents a change in the roster between the employee $cand.emp_a$ and $cand.emp_b$ at the specific day. The neighborhood for some employee emp_a is all the possible changes in the roster where one of the candidate employees is the employee emp_a. A cost-oriented objective function evaluation (line 13) is the critical place in terms of time complexity because the algorithm typically spends 80% of its whole runtime while too many obviously bad solutions are unnecessarily evaluated. The usage of some technique, which is able to quickly decide whether the structure of the roster is bad or good, could speed up execution of TS significantly.

This problem can be solved using classifiers belonging to the group of pattern recognition techniques [5]. The classifier defines a set of classes characterizing the input data. Its aim is to assign the correct class to each input vector according to the experience gained in the learning phase on the training data. The typical classifier realization is a neural network.

3.1 Neural network background

Many types of neural networks exist, however, in this paper we use the multilayer perceptron network (MLP) because it is one of the most widely used and showed the best general results in our experiments. The basic components of a neural network are neurons. They are arranged into the layers and connected via the links. Each link (u, v) has its weight w_{uv} where u is the source neuron and v is the target neuron. The output y_v of the neuron v is defined as

$$y_v = F\left(\sum_{\forall u|(u,v)\in\delta^-(v)} w_{uv} \cdot y_u + \Theta\right) \quad \text{where}$$

$\delta^-(v)$ is a set of links entering neuron v, Θ is a threshold of the neuron v and F is an activation function (e.g. the sigmoid function). To learn a neural network, training data is needed in the format: (input pattern, corresponding class). The training (learning) process continuously updates weights of links to ensure better behavior of the neural network on the input data. In the other words, the result of the neural network should be equal to the desired output for each input pattern. It is necessary to also use testing data to be aware of overfitting. Otherwise, the trained network would not be applicable to the data other than the training one.

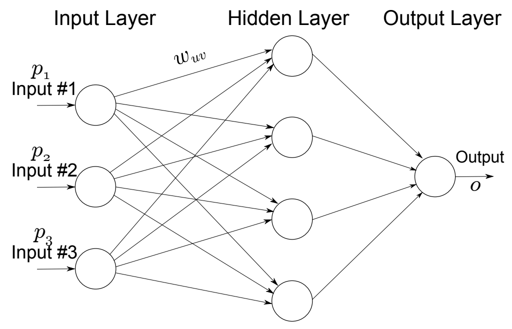


Figure 3: An example of the neural network

A 2-layer neural network, namely MLP, is shown in Figure 3.[1] There are three inputs nodes, one output neuron and four hidden neurons. The usage of one hidden layer allows one to reproduce any differentiable function. On the other hand, a network without hidden layers (i.e. 1-layer network) can work only with linearly separable problems.

3.2 Neural network usage

In our approach, the input pattern is considered as a simple change in the roster of one employee (see Figure 4). The input pattern could be regarded as a vector p with length $2 \cdot d$. Variable $p_r, \forall r \in \{1, \dots, d\}$ represents the roster of employee i before the change and $p_r, \forall r \in \{d+1, \dots, 2 \cdot d\}$ after the change. The output of the neural network o expresses whether the change leads to an objective function improvement ($o = 1$ - class 1/good) or not ($o = 0$ - class 2/bad).

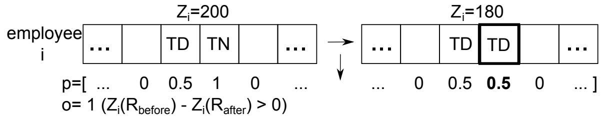

before the change
Z_i=200
after the change
Z_i=180
employee i
...
TD
TN
...
...
TD
TD
...
p=[... 0 0.5 1 0 ... ... 0 0.5 0.5 0 ...]
o= 1 ($Z_i(R_{before}) - Z_i(R_{after}) > 0$)

Figure 4: A change in the roster of employee i

Comparing results of the available possibilities, we decided not to use two classes only (i.e. good and bad) but five "pseudo-classes" – super good (desired output=1), good (=0.7), equal (=0.5), bad (=0.3)

[1] In this paper we do not count the input layer as a layer.

and super bad (=0). The reason is to derive more precise information from the neural network which can be used for a more accurate decision process. Moreover, with the use of this approach, the classification rate (i.e. the percentage of successfully classified data) was improved. For example, let the threshold for classification be 0.5, if the classifier returns an output $o > 0.5$ for the super good or good pattern then it is classified correctly.

3.3 Classifier in Tabu Search

In general, the trained classifier will never have a 100% success rate in matching the input pattern to the correct class. Therefore, we use the classifier in Tabu Search as a filter which eliminates the majority of the potentially bad solutions. The remaining solutions are then evaluated by the standard objective function Z (see Figure 5). Otherwise, if Tabu Search depended only on the classifier, it could easily get stuck in some place where the classifier fails.

```
1: $filter = 50$ {number related to the filter eliminating}
2: for $\forall cand \in neighborhood(emp_a, R_{best})$ do
3:    if $cand \in tabu\_list$ then
4:       continue
5:    end if
6:    $classification = classifier(cand.emp_a) + classifier(cand.emp_b)$
7:    if $size(list\_cand) < filter$ then
8:       $list\_cand.add(cand)$
9:    else if $worst(list\_cand) < classification$ then
10:      $list\_cand.replace\_worst(cand)$
11:   end if
12: end for
13: for $\forall cand \in list\_cand$ do
14:    $R_{tmp}$ = apply changes from $cand$
15:    $actual\_Z = Z(R_{tmp})$
16:    if $actual\_Z < best\_Z$ then
17:       $R_{best} = R_{tmp}$
18:       $best\_Z = actual\_Z$
19:    end if
20: end for
```

Figure 5: Pseudo-code of the modified Tabu Search with a classifier as a filter. The code replaces the code from Figure 2 on lines 8–18

4 Experimental results

The above proposed approach was verified on the computer with an Intel Core i7-3520M 2.90 GHz CPU and an 8.0 GB DDR3 RAM. Our TS algorithm was implemented in C# programming language. The experiments were executed on the benchmark instances "Valouxis", "Millar", "Millar-s", "Ortec" and "GPost" (http://www.cs.nott.ac.uk/~tec/NRP/). The description of instances is depicted in Table 1.

In our experiments only 2-layer neural networks were used because the neural network without a hidden layer did not reach a good rate. Each neuron used a sigmoid activation function and the hidden layer consisted of 10 neurons.

Table 1: The description of the tested instances

Instance	Number of			
	shifts	days	employees	constraints
Millar	2	14	8	14
Millar-s	2	14	8	8
Ortec	4	31	16	52
Valouxis	3	28	16	18
GPost	2	28	8	33

For each benchmark instance the classifier was trained on data obtained as follows: we collected data from 500 TS runs where each run had a different initial solution. Subsequently, 10000 different samples were selected with uniform distribution to the classes. Then 70% of them were used as the training data and 30% as the testing data. A back-propagation method was chosen for the neural network training with learning rate of 0.3 and 100 iterations (epochs) which corresponds approximately to two seconds under the circumstances. The set of testing data was used to detect overfitting of the trained classifier and to verify the classification rate.

4.1 Results

Table 2: Experiments with the trained classifiers

Instance	CPU time [s]		Deviation in solution quality
	classifier	standard	
Valouxis	0.39	3.86	-1
Millar	0.05	0.11	0
Millar-s	0.04	0.07	0
Ortec	2.69	5.23	+2
Gpost	0.53	1.07	+3
Average	0.74	2.07	+0.8

Trained classifiers were employed in TS (see Figure 5), while, in all experiments, we never used the same initial solution that was used for obtaining the training and the testing data. Table 2 shows the important values from the comparison of the cost-oriented objective function evaluation and the classifier used as a filter which eliminates 90% of potentially bad solutions in each iteration of TS. The first column identifies the benchmark instance. The following columns show the CPU time consumed by the classifier-oriented and the standard cost-oriented evaluation (only two parts of the entire roster, i.e. the roster of employee $cand.emp_a$ and $cand.emp_b$, are re-evaluated). To measure as accurately as possible, we counted only the total time spent in the given evaluation methods. Finally, the last column represents the deviation in quality of the solution (i.e. the difference in the number of soft constraint violations). For this comparison, the resulting roster from the cost-oriented evaluation was chosen as the reference solution.

Table 3 investigates the behavior of the classifier if the problem instance is slightly changed. The particular instance modification is indicated in the bracket next to its name: a) *sc+/sc-* – one soft constraint is added/removed and b) *e+/e-* – one employee is added/removed. For example, "Valouxis

Table 3: Experiments on similar instances

Instance	CPU time [s]		Deviation in solution quality
	classifier	standard	
Valouxis (sc-)	0.21	2.36	0
Valouxis (sc+)	0.50	2.83	+2
Valouxis (e+)	0.68	3.23	+3
Valouxis (e-)	0.62	3.81	+5
Average	0.50	3.06	+2.5

(sc+)" is the standard Valouxis instance with one added soft constraint. The same classifier was used for each corresponding instance, e.g. for Valouxis in Table 2 and Valouxis (sc-) in Table 3 we considered the same weights of MLP, etc.

4.2 Discussion

Figure 6 demonstrates the results from Table 2, namely for the Valouxis instance. Speedup is visible there in the execution of the scheduling algorithm with comparable quality of the final solution. The neural network ends its run in 0.4s and at the same time the cost-oriented evaluation has a 5 times bigger value of the objective function. It should be noted that the same stopping criterion was used in both cases. Furthermore, the graph shows a slow convergence for the cost-oriented evaluation compared to the neural network.

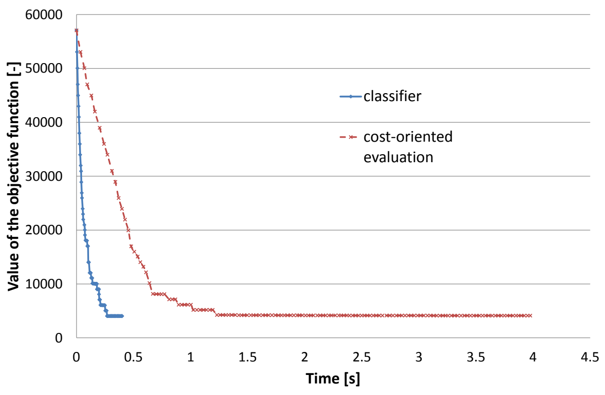


Figure 6: The run of the Tabu Search algorithm on the Valouxis – progress of the objective value over time for the cost-oriented evaluation and the evaluation using the classifier

Overall, the experiments confirm a good classification rate (86.93% in average). The average speedup for the 2-layer neural network is roughly 4.5 times while the solution quality remains similar.

The experimental results reveal two important observations. The trained classifier can be successfully used: 1) on the same instances with data different from the training data and 2) on the similar instances with added/removed soft constraint/employee. Our current approach can also handle various types of employees' contracts (e.g. part-time vs. full-time which leads to different soft constraints). This problem is solved by several classifiers for each type of employees' contracts, e.g. the Ortec benchmark instance.

5 Conclusions

The paper shows that usage of a learning technique can be a benefit for personnel scheduling problems. As a learning technique, we use the neural network. The trained neural network works as a classifier and, based on a simple change in the roster of one employee, it tries to estimate whether this step is good (i.e. improvement in the objective function evaluation) or not. This behavior is used as a filter which eliminates the major part of the potentially bad rosters while remaining solutions are evaluated by the objective function. The experiments demonstrate interesting algorithm runtime reductions on the same or similar instances with the preservation of comparable solution quality. With respect to the approach in [9], our solution can deal with a varying number of employees during different planning periods. Moreover, our obtained classification rate and speedup are slightly better mainly due to the considered input pattern (the roster change for one employee versus the whole roster in [9]).

In our future research, we want to focus on solving more complex instances and on a combination of the multiple classifiers (e.g. Adaptive Boosting algorithm) to achieve more accurate results.

Acknowledgment

The authors would like to thank Prof. Jiri Matas from the Center for Machine Perception at CTU for his valuable advice on pattern recognition techniques. This work was supported by the Ministry of Education of the Czech Republic under the Projects OK7X 295372 and by the Technology Agency of the Czech Republic under the project CAK3 TE01020197.

References

[1] U. Aickelin and K. A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *CoRR*, abs/0802.2001, 2008.

[2] C.M. Bishop and G. Hinton. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[3] 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, 2004.

[4] 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.

[5] C.H. Chen. *Handbook of Pattern Recognition and Computer Vision: 4th Edition*. Imperial College Press, 2010.

[6] F.W. Glover and M. Laguna. *Tabu Search*. Springer, 1998.

[7] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[8] J. Li, E. K. Burke, and R. Qu. Integrating neural networks and logistic regression to underpin hyper-heuristic search. *Know.-Based Syst.*, 24(2):322–330, 2011.

[9] J. Li, E. K. Burke, and R. Qu. A pattern recognition based intelligent search method and two assignment problem case studies. *Applied Intelligence*, 36(2):442–453, 2012.

[10] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 2008.

[11] B Yegnanarayana. Artificial neural networks for pattern recognition. *Sadhana*, 19:189–238, 1994.

[12] W. Zhang and T. G. Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Technical Report, Department of Computer Science, Oregon State University.*, 1997.