Rostering from Staffing Levels

a Branch-and-Price Approach

Egbert van der Veen[1], Bart Veltman[2]
[1]ORTEC, Gouda (The Netherlands), Egbert.vanderVeen@ortec.com
[2]ORTEC, Gouda (The Netherlands), Bart.Veltman@ortec.com

Abstract: Many rostering methods first create shifts from some given staffing levels, and after that create rosters from the set of created shifts. Although such a method has some nice properties, it also has some bad ones. In this paper we outline a method that creates rosters directly from staffing levels. We use a Branch-and-Price (B&P) method to solve this rostering problem and compare it to an ILP formulation of the subclass of rostering problems studied in this paper. The two methods perform almost equally well. Branch-and-Price, though, turns out to be a far more flexible approach to solve rostering problems. It is not too hard to extend the Branch-and-Price model with extra rostering constraints. However, for ILP this is much harder, if not impossible. Next to this, the Branch-and-Price method is more open to improvements and hence, combined with the larger flexibility, we consider it better suited to create rosters directly from staffing levels in practice.

Keywords: Personnel rostering, staffing levels, employee preferences, Branch-and-Price, column generation

1 Introduction

The efficient use of human resources is important in many industries. For sure this holds for health care, where the main costs are the loans of the workforce. Inefficient rostering leads to inefficient use of scarce and expensive resources, and hence to unnecessary costs.

In most literature the rostering process is decomposed into three important subphases. First, there is the *staffing* phase. In this phase 'staffing levels' are created from a predicted workload. Staffing levels represent the number and skill level of required resources within a given time slot. For example, between 7:30 AM and 9:00 AM, two nurses need two be available at the South ward, from 8:00 AM till 10:30 AM, an anesthetist needs to be available at Operating Theatre 8, etc. Next, in the *shift scheduling* phase, shifts are created from the staffing levels. These shifts are supposed to cover the staffing levels as efficient as possible. After that, in the *shift rostering* phase, workers are assigned to the created shifts, i.e. rosters are created. This rostering process is illustrated by Figure 1.

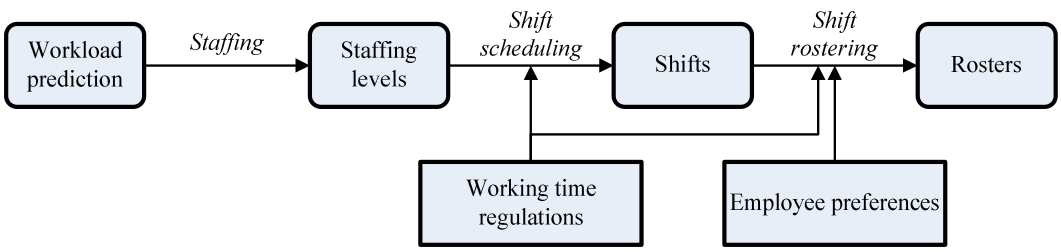

Figure 1*: Rostering from staffing levels: a two-step approach.*

As illustrated by Figure 1, working time regulations and employee preferences constrain the creation of shifts and rosters. However, where working time regulations constrain the creation of the shifts (e.g. shift length between 6 and 9 hours) as well as the creation of rosters (e.g. an employee is not allowed to work more than 6 shifts a week), employee preferences are only accounted for when creating rosters. A major downside of this way of rostering is that the shifts that result from the shift scheduling phase might not allow for the creation of good or even valid rosters, since the employee preferences further constrain the feasible shift set. To solve this, we propose a rostering method that creates rosters *directly* from staffing levels, as is graphically illustrated by Figure 2.

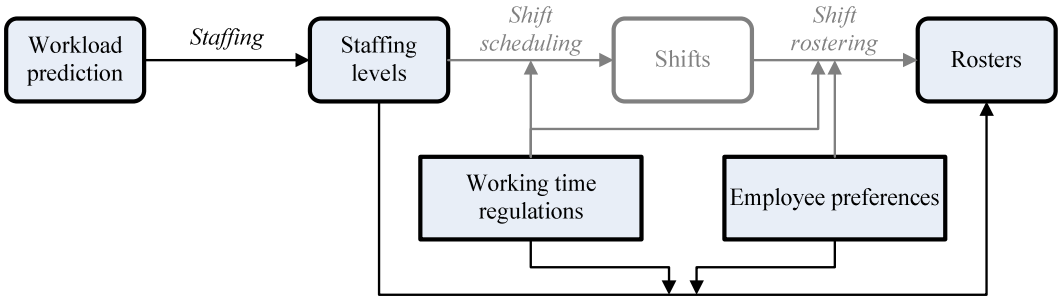

***Figure 2**: Rostering from staffing levels: a one-step approach.*

Now, as is observed from Figure 2, this 'new' scheduling method creates rosters directly from staffing levels, and, while doing this, it keeps track of working time regulations and employee preferences simultaneously. In this way, employee preferences are accounted for directly during the creation of the shifts. Taking employee preferences into account is a hot topic in health care scheduling due to a general lack of care and cure professionals like nurses and physicians. Now, more than before, there is a strong orientation on matching staffing levels and employees as good as possible. This implies, in some extent, that the two-step approach was adequate in the past, but it is expected to be too limited in the (near) future.

The main objective of this paper is to develop a method that creates rosters directly from staffing levels, and while doing this taking working time regulations and employee preferences into account.

This paper is structured as follows. The next section first discusses some literature related to this rostering problem. After that, in Section 3, the model used to solve the rostering problem is outlined and Section 4 discusses the results of our software implementations. Section 5 draws conclusions from this research.

2 Related literature

Some literature already (partly) looks at rostering directly from staffing levels. Keith (1979) outlines a method that creates rosters from 'shift templates'. With this method, only shifts that come from a predefined set of (template) shifts may be used when creating the rosters. However, this method does not keep track of employee preferences, because the cost of assigning shifts to employees is not employee specific. Dowsland and Thompson (2000) outline a straightforward extension of the model of Keith that makes the cost of shifts employee specific. By making the cost of shifts employee specific, employee preferences are taken into account (to some extend).

Although the 'classic' scheduling method has some downsides, as outlined in Section 1, it has one major positive aspect. In this scheduling method the larger problem of creating a roster is decomposed into two smaller subproblems, the shift scheduling and shift rostering subproblems. Caprara, Monaci and Toth (2001) introduce a method that solves the shift rostering and shift scheduling phase iteratively. When the shift scheduling phase comes up with a set of shifts that

does not match employee preferences very well, a recreation of shifts can take place, such that the new set of shifts (hopefully) meets the employee preferences better. However, the method of Caprara et al. cannot directly be applied to our problem, since they apply it to a train personnel rostering problem. Whereas in our problem we are rostering from staffing levels, the train personnel rostering has to roster tasks. Furthermore, the objective of the train personnel rostering problem of Caprara et al. is to minimize the amount of time needed to work a complete schedule, whereas in our problem the time horizon is fixed; all staffing levels need to be matched within the given timeframe. The major problem, however, with iteratively solving the shift scheduling and shift rostering phase is that it is unclear what information should be provided to the shift scheduling phase when shift rostering cannot find a solution. Moreover, when the shift rostering phase is unable to find a solution, it is also not clear whether this is due to the set of created shifts or whether there is no solution at all.

3 Branch-and-Price model

In this study we restrict ourselves to instances where employees are allowed to work at most one shift that consists of one (large) time slot without interruptions. Although this seems rather restrictive, these instances are already NP-complete, since we are dealing with staffing levels for multiple skills, see Van der Veen (2009). We use a Branch-and-Price (B&P) method to create the rosters. The Brand-and-Price method presented here can be easily extended to rostering problems with more general shifts.

For readers not familiar to Branch-and-Price methods we outline a short introduction to this topic in Section 3.1. A more elaborate introduction to the concept of Branch-and-Price can e.g. be found in Hans (2001). Both Hans (2001) and we assume that readers are familiar with linear programming theory, and concepts like linear programming, dual problems and reduced cost. A general introduction to linear programming theory can be found in e.g. Chvátal (1985).

3.1 Branch-and-Price: general introduction

Large linear programs are often solved by means of column generation. This method was first applied by Gilmore and Gomory (1961). With a column generation approach initially only a small set of columns is included in the linear program, which often is referred to as the *master problem*. After the master problem is solved it is checked, via the so-called *pricing problem*, whether there is a column with profitable reduced cost that is not yet included in the master problem. If it exists it is added to the master problem. After that, the procedure is repeated until the pricing problem cannot find additional columns with reduced cost. However, when this procedure is applied for integer linear programs we might not end up with an integer solution. To find a solution anyway, some kind of branching is applied. After a branching took place, often a subset of the generated columns is not valid anymore for the current node. These columns are removed from the master problem, and, after that, the pricing problem is called until no profitable column is found anymore. Two things can happen now. First, the solution can be integer. If it is a better solution than the best one so far, it is set as the current best integer solution, otherwise it is fathomed. Secondly, the solution can be fractional. If the objective value of the fractional solution is worse than the current best integer solution it is also fathomed, otherwise additional branching is needed.

In Figure 3 the Branch-and-Price method is schematically summarized.

Note that a direct ILP formulation of our rostering problem, where every possible shift is represented by a variable, contains a huge number of variables. Moreover, for reasonable problem dimensions this number of possible shifts would be too large to be modelled and solved by computers. Hence, we chose to apply a Branch-and-Price method to solve the rostering problem.

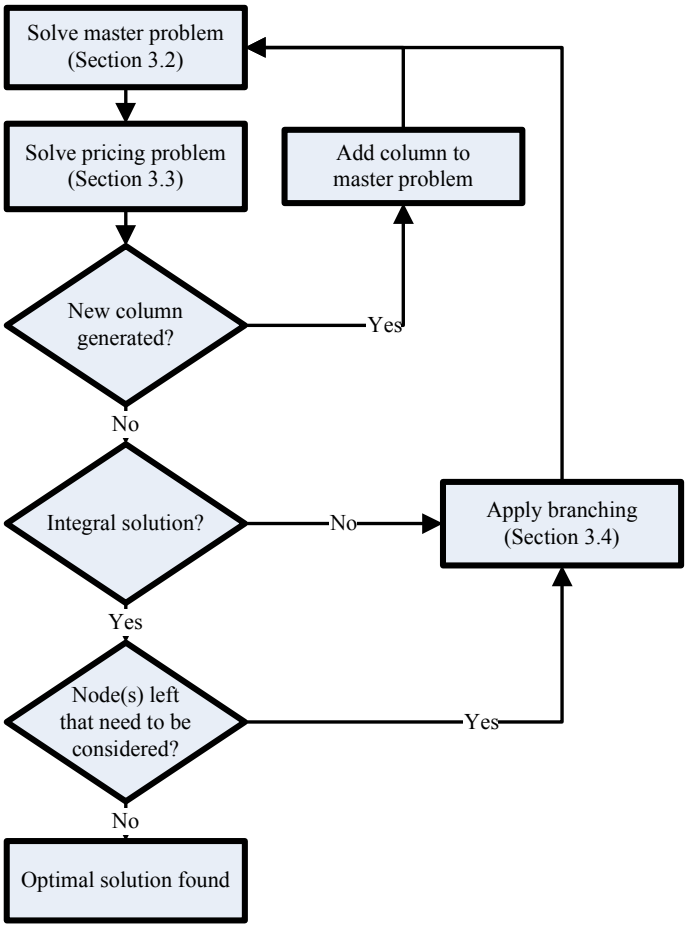

***Figure 3**: Branch-and-Price: a schematic overview.*

The most important parts of this Branch-and-Price method, the master problem, the pricing problem and the branching method are explained in more detail in respectively Section 3.2, Section 3.3 and Section 3.4.

3.2 Branch-and-Price: master problem

Given a set of employees ($i = 1,\ldots,n$), skills ($j = 1,\ldots,m$), time slots ($t = 1,\ldots,T$), one may think of an hour each, and staffing level d_{jt} per combination of skill and time slot, we come to the following master problem

$$\min \quad \sum_{i=1}^{n} \sum_{k \in K_i} c_{ik} \cdot x_{ik} \tag{1a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{k \in K_i} a_{jtk} \cdot x_{ik} \geq d_{jt} \quad \text{for } j = 1,\ldots,m; t = 1,\ldots,T \tag{1b}$$

$$\sum_{k \in K_i} x_{ik} \leq 1 \quad \text{for } i = 1,\ldots,m \tag{1c}$$

$$x_{ik} \geq 0, \quad \text{integer, for } i = 1,\ldots m; k \in K_i \tag{1d}$$

In the above model, c_{ik} denotes the cost of employee i working shift k. We define c_{ik} to be equal to the number of working periods in shift k. Next to this, x_{ik} equals 1 when employee i works shift k. The set K_i denotes the set of shifts that are created specifically for employee i. Furthermore, a_{jtk} equals 1 when according to shift k the related employee would work on skill j during period t. It equals 0 otherwise. Hence, the objective function (1a) minimizes total cost. Constraints (1b) thus imply that the staffing levels are met. Constraints (1c) ensure that an employee works at most one shift. Constraints (1d) imply that the x_{ik} are non-negative and integral. Note that in fact the x_{ik} are binary variables, but we modeled them as general non-negative integers. By doing this the dual problem of the master problem is smaller, because it does not include variables corresponding to the upper bound constraints on the variables. This makes the pricing problem that is presented in Section 3.3 easier.

To get the Branch-and-Price method started we have to create some initial variables (i.e. columns) for the master problem. Otherwise there is no dual problem and then no reduced cost can be calculated. An easy way to initialize the master problem is to set all K_i such that they contain exactly one shift k for which $a_{jtk} = d_{jt}$. When one of these shifts is worked, the total demand is covered. Note that such a shift is not valid, since we assumed an employee cannot work on two skills at the same time. However, by setting the $c_{ik} = M$, for these initial shifts, where M is a significantly large number, we make sure that after generating a number of additional (valid) columns, via the pricing problem, these initial (invalid) shifts do not appear in the solution. Hence, all information about the validity of shifts is assumed to be incorporated in the pricing problem. We have to admit that this is probably not the smartest way to initialize the master problem. However, since all information about validity of shifts is contained in the pricing problem, this way of initialization works independent of the constraints that are implied on shifts.

3.3 Branch-and-Price: pricing problem

To determine the next column to be added to the master problem, the pricing problem needs to be solved. The pricing problem is solved when the column with the least reduced cost is found. Note that negative reduced costs are profitable for our problem, since we have a minimization problem. Hence, the column with the most negative reduced cost is the one we like to add to the master problem. However, in order to calculate reduced costs the dual of the master problem needs to be solved first. The dual of the LP relaxation of the master problem is given by

$$\max \quad \sum_{j=1}^{m}\sum_{t=1}^{T} d_{jt} \cdot \pi_{jt} + \sum_{i=1}^{n} \nu_i \tag{2a}$$

$$\text{s.t.} \quad \sum_{j=1}^{m}\sum_{t=1}^{T} a_{jkt} \cdot \pi_{jt} + \nu_i \le c_{ik} \text{ for } i = 1,\ldots,n; \forall k \in K_i \tag{2b}$$

$$\pi_{jt} \ge 0 \text{ for } j = 1,\ldots,m; t = 1,\ldots,T \tag{2c}$$

$$\nu_i \le 0 \text{ for } i = 1,\ldots,n \tag{2d}$$

The reduced cost of a shift k' of employee i is now given by

$$c_{ik'} - \sum_{j=1}^{m}\sum_{t=1}^{T} a_{jk't} \cdot \pi_{jt}^{*} - \nu_i^{*} \tag{3}$$

where π_{jt}^{*} and ν_i^{*} denote the optimal values of π_{jt} and ν_i respectively. To determine the shift with the least reduced cost it is not too inefficient to enumerate all possible shifts, calculate their reduced costs, and select the one with the least reduced cost. To do this, let J_i denote the set of skills employee i has. First determine

$$\min_{j \in J_i} \{\pi_{jt}^*\} \tag{4}$$

for all t. Denote the vector containing all these π_{jt}^* by π. Now, let w represent a working pattern: a sequence of 0s and 1s indicating in which time slots the employee should (1) or should not (0) work. By enumerating all working patterns w that employee i is allowed to work, and calculating $w \cdot \pi$, we find the working pattern with the least reduced cost. Via the working pattern and π we then determine for every period on which skill the employee is working, i.e. the shift is determined. If $w \cdot \pi < v_i^*$ the reduced cost of this shift is negative, and it is added to the master problem.

Let the minimal and maximal number of consecutive time slots employee i needs to work (if called to work) be denoted by $mindu_i$ and $maxdu_i$, respectively. Then the number of working patterns is bounded by

$$\sum_{t=T-\text{maxdu}_i+1}^{t=T-\text{mindu}_i+1} t \leq \sum_{t=1}^{T} t \leq \frac{1}{2}(T+1)T \leq T^2 \tag{5}$$

Determining $\min_{j \in J_i} \{\pi_{jt}\}$ for $t = 1,\ldots,T$ has time complexity $O(mT)$, hence the time complexity of the pricing problem is $O(mT + T^2)$, which is polynomial.

Employees are simply selected by increasing index, i.e. first we solve the pricing problem for employee 1 and see whether a shift with negative reduced cost exists for employee 1. If it exists it is added to the master problem, and the master problem is resolved. Otherwise, employee 2 is selected and the procedure repeats. If there is no employee for whom a shift with negative reduced exists we need to apply branching. The branching procedure is described in the next section.

3.4 Branch-and-Price: branching

When the solution obtained to the LP relaxation after column generation is fractional, branching needs to be applied. The branching needs to result in an integer solution. It is also important that branching decisions can be incorporated in the pricing problem, i.e. it must be possible to adjust the pricing problem in such a way that the generated columns respect the branching decisions.

With the rostering problem we apply branching to the time slots in which employees start and stop working. This is easily incorporated in the pricing problem, and it leads to an integral solution. The former is trivial; simply do not include shifts that have working periods before the start period or after the stop period in the enumeration. The latter is less obvious. Moreover, when start and stop periods have been fixed for all employees there might not even be a feasible integer solution amongst the generated columns. This is illustrated by Example 1.

Example 1. Given a rostering problem with 2 employees, 2 time slots and 2 skills. Demand (d_{jt}) is given by the following matrix where $j \in \{1,2\}$ and $t \in \{1,2\}$

$$D = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

And the generated shifts are $a = (1,1)$ and $b = (2,2)$ for employee 1; and $p = (1,2)$ and $q = (2,1)$ for employee 2. The first number indicates the skill that is worked on during time slot 1, the second number indicates the skill that is worked on during time slot 2. Furthermore, the cost of the shifts equal the number of working periods in it, hence $c_{1a} = c_{1b} = c_{2p} = c_{2q} = 2$. For both employees 1 and 2 we fixed time slot 1 as start period and time slot 2 as stop period. The optimal solution clearly equals $x_{1a} = x_{1b} = x_{2p} = x_{2q} = ½$. Furthermore, this is the unique optimal solution, and there does not exist an integer solution amongst the generated shifts. Furthermore, there are no shifts

with negative reduced cost (since this solution is optimal given the demand D) and the start and stop periods are fixed for both employees. Hence, the example shows that after full branching an integer solution does not necessarily exist.

An integer solution is easily constructed when start and stop periods are fixed for all employees. We do this via the construction of a b-matching problem. A b-matching problem is similar to regular matching problems except that in b-matching for every vertex v the matching has to contain exactly b_v edges that are connected to v. A more detailed description of (b-)matching problems is found in e.g. Schrijver (2003).

Now, for every time slot t, create vertices for all employees that are allowed to work during t. Furthermore, for every time slot t, create vertices v for every skill j where $d_{jt} > 0$ with $b_v = d_{jt}$. Now, for every time slot t, connect an employee vertex to a demand vertex whenever the employee has the corresponding skill. Note that we now have t independent b-matching problems. After these b-matching problems are solved for all time slots t, shifts are created from their solutions and this offers a solution to the rostering problem for the current node of the branching tree. Note that b-matching problems are polynomially solvable, see Schrijver (2003). We know that there are feasible solutions to these b-matching problems since the fractional solutions for the rostering problem offer fractional solutions for the b-matching problems, which implies that there are integral solutions to the b-matching problems. Since b-matching has a totally unimodular technology matrix, such a fractional solution is a convex combination of integral solutions. Solving the b-matching problems directly returns integral solutions, and from these the shifts are created. Hence, the Branch-and-Price method, of which the branching scheme is an important part, leads to an integral solution.

As a final note to this chapter we want to remark that although we engineered the above Branch-and-Price approach in such a way that it solves our rostering problem, it offers a lot of flexibility to solve other rostering problems. In fact the pricing problem determines the kind of constraints that are implied on shifts, since the master problem is nothing but a constraint set covering problem, and only shifts generated by the pricing problem are used here. So even when there are lots of constraints implied on shifts, and it is not very easy to find additional shifts, this 'difficulty' is isolated in the pricing problem part of the solution approach. Furthermore, it is also straightforward to combine heuristic and optimal pricing problems to generate additional columns, such that it is well possible to prevent the pricing problem from consuming too much solving time. Of course, the branching part also needs to be adjusted when other rostering problems are modeled, but we believe that this will not be too hard. Note that for these 'larger' rostering problems columns correspond to week or month rosters, instead of single day shifts. Finally, we want to stress that there are lots of possibilities to incorporate 'practical constraints' in the Branch-and-Price formulation. For example, in practice it is often preferred that employees only work shifts from a predefined set of 'template' shifts. This is easily incorporated in the Branch-and-Price model by simply letting the pricing problem check which of the template shifts offers the best reduced cost.

4 Experimental results

To assess the performance of our Branch-and-Price (B&P) method we created an ILP formulation of the rostering problem. In the previous chapter we mentioned that an ILP formulation based on shifts has too many variables for reasonable problem dimensions. Hence, we created an ILP formulation where shifts are implicitly defined. To do this, we created variables indicating whether an employee works on a specific skill during some specific time slot or not. This implies that the number of variables is linear in the problem dimensions, and will not become too large when the problem dimensions get larger. For a complete overview of the created ILP model we refer to Van der Veen (2009). However, we need to stress here that ILP is not a very good way to solve (health care) rostering problems. ILP works here, due to our restriction that employees are allowed to

work only one shift that consists out of a single time slot without interruptions. However, for this ILP model, we need artificial binary variables for every employee and every time slot to model the rostering problem correctly. When employees are allowed to work multiple shifts, or when more elaborate constraints are implied on shifts, this number of artificial variables certainly increases further, making the rostering problem both hard to model *and* hard to solve in an ILP setting.

The ILP model is solved via the ILOG CPLEX 11.0 callable library. The B&P model is implemented in SCIP. For a general description of SCIP the reader is referred to Achterberg (2007). Within B&P, we again use CPLEX 11.0 to solve the LP relaxations.

To assess the quality of the B&P method we used randomized datasets. Lots of instances are generated and solved, in order to get a good indication of the overall quality of the algorithm. To create randomized demand the following procedure is applied for every skill category *j*.

First, create *n* integers N_i for which

$N_i \overset{\text{iid}}{\sim} U(1,T)$, such that *n* satisfies $\sum_{i=1}^{n-1} N_i < T \leq \sum_{i=1}^{n} N_i$

Next, generate *n* integers $D_i \overset{\text{iid}}{\sim} U\left(0, \lfloor \frac{n}{m} \rfloor\right)$, and create sets

$$\tau_i : \tau_1 = \{1,\ldots,N_1\}; \tau_i = \left\{\sum_{p=1}^{i-1} N_p + 1,\ldots,\sum_{p=1}^{i} N_p\right\} \text{for } 2 \leq i \leq n-1; \tau_n = \left\{\sum_{p=1}^{n-1} N_p + 1,\ldots,T\right\}$$

Now, for all time slots $t \in \tau_i$ set demand to D_i.

Note that with this demand we have 'blocks of time slots' with equal demand. We think this kind of demand better fits reality than when a uniform random number is drawn for every individual time slot. The latter implies very strong fluctuations in demand, whereas the former implies a more 'controlled' demand pattern, which, we think, better fits reality.

Next to random demand, we created 'random' skills. For every employee first a uniform random number *m'* is generated from [1,*m*]. After that, uniform random numbers are drawn from [1,*m*] until *m'* different numbers are obtained, indicating the skills the employee *has*. With these randomized demand and skills there is no assurance that there are enough available (skilled) employees to cover demand. However, in Van der Veen (2009) it is indicated that, with these randomized skill sets, it is *likely* that demand can be met. Detailed results of the experiments we performed can be found in Van der Veen (2009). Table 1 summarizes some important results. The main focus is on the comparison of solving times of our B&P approach versus the ILP model. We do not have to look at solution quality, since both implementations return optimal solutions.

Test case						*Imple-mentation*	*avg solving time (sec.)*	*#unsolved instances*
#	*n*	*m*	*T*	*mindu*	*maxdu*			
1	20	2	24	11-20	16-24	B&P	5.3	0
						ILP	3.6	2
2	20	5	24	11-20	16-24	B&P	68.6	24
						ILP	9.4	23
3	20	2	$4t : 1 \leq t \leq 12$	$\frac{1}{2}T$	$\frac{3}{4}T$	B&P	103.3	4
						ILP	7.2	0

4	20	5	$4t : 1 \leq t \leq 12$	$\frac{1}{2}T$	$\frac{3}{4}T$	B&P	430.8	30
						ILP	25.6	3
5	$5i : 1 \leq i \leq 10$	2	24	12	18	B&P	6.5	0
						ILP	13.4	0
6	$5i : 1 \leq i \leq 10$	5	24	12	18	B&P	65.3	5
						ILP	44.6	9

Table 1: *Computational experiment: results*

The five leftmost columns of Table 1 show information on the test case solved. Columns 1, 2 and 3 show respectively the number of employees, the number of skills, and the number of time slots that are part of the test case. Columns 4 and 5 indicate the minimal number of time slots an employee should work when called to work, and the maximal number of time slots an employee is allowed to work, respectively. The sixth, seventh and eighth column of Table 1 indicate the average time needed to find a solution (column 7) by the corresponding solver (column 6), and the number of unsolved instances (column 8). Note that for all parameter sets (set of fixed values of *n*, *m*, *T*, *mindu* and *maxdu*) 20 randomized test instances are created and solved. This implies that the total number of unsolved instances per test case can be significant. Note that for test cases 1 and 2 the values of *mindu* and *maxdu* are varied. For test cases 3 and 4 the value of *T* is varied, and for test cases 5 and 6 the values of *mindu* and *maxdu* are varied. Test cases for which *mindu* > *maxdu* are ignored, since they are infeasible.

Summarizing from Table 1 we observe that B&P performs relatively bad for the test cases where *T* grows large. Furthermore, we observe that B&P has a stronger dependence on *m* than ILP has. This can be observed from results 1 and 2; the average solving time of B&P increases faster than the average solving time of ILP when *m* increases. This is also observed from test cases 5 and 6.

The relative bad performance of B&P for larger *T* is caused by the fact that the time needed to solve the pricing problem depends quadratically on the value of *T*, see Section 3.3. Furthermore, for larger *T* (and *m*) the number of columns that needs to be generated is also larger, which implies that the total time consumed by the pricing problem grows larger and larger.

For most cases we see that ILP outperforms B&P. Even though, the performance of B&P is quite acceptable in most cases. However, it is not too difficult to improve the B&P method, but improving the ILP implementation is much harder. We list some good possibilities for improving the B&P implementation. As outlined above, the time needed by the pricing problem increases a lot when *T* gets larger. There also is a slight dependence on the value of *m*. Decreasing these dependencies will improve both the time needed by the pricing problem as well as the overall solving time needed. Next to the pricing problem the most time of the B&P algorithm is consumed by solving the LP relaxations of the master problem. For larger *m* and *T* the master problem gets very large during the solving process, and hence solving the LP relaxations consumes more and more time. Column management strategies, that try to keep the master problem from getting (very) large, probably further decrease the total solving time. Finally, we only generate one column at a time when the pricing problem is called. Generating multiple columns at once might also improve the solving speed.

Finally, we want to stress that in this paper we studied a relatively simple rostering problem. As already indicated in this section, it is expected that for more realistic and more elaborate models, it is (almost) impossible to model these problems with ILP, and hence for those problems B&P will be the preferred solving method.

5 Conclusions

Lots of rostering methods first create shifts based on staffing levels, and after that create rosters from this set of created shifts. In order to create rosters *directly* from staffing levels, which allows accounting for employee preferences when creating shifts, Branch-and-Price turns out to be a flexible approach to do so. On a small subclass of rostering problems Branch-and-Price and ILP perform almost equally well. Although this subclass of rostering problems is NP-complete, it contains only a small set of constraints that are implied on shifts in practice. However, due to the structure of the Branch-and-Price method it is not too hard to extend it in order to include the complex constraints from practice. Moreover, due to the structure of Branch-and-Price methods there is a lot of flexibility to deal with these 'difficult' constraints efficiently. However, extending the ILP model with more and hard constraints is not at all an easy task, if not impossible. Furthermore, extending the ILP model with such practical constraints results in a significant increment in solving time.

Considering the fact that Branch-and-Price offers far more flexibility to deal with practical constraints than the ILP model does, we expect Branch-and-Price to perform relatively better for rostering problems where more constraints are implied on the shifts, or where week or month rosters are created. Furthermore, there are some good and easy ways to incorporate practical concepts, like template shifts, into the Branch-and-Price approach, which also makes it of better practical use.

References

Achterberg, T. (2007) Constraint Integer Programming, Ph.D. Thesis Technische Universität Berlin

Caprara, A.; Monaci, M.; Toth, P. (2001) A Global Method for Crew Planning in Railway Applications, In: Voß, S.; Daduna, J.R. (eds.) Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin Heidelberg, pp. 17 – 37

Chvátal, V. (1983) Linear Programming, W.H. Freeman and Company, New York

Dowsland, K.A. (1998) Nurse Scheduling with Tabu Search and Strategic Oscillation. European Journal of Operational Research 106 (1998) 2-3, pp. 393-407

Dowsland, K.A.; Thompson, J.M. (2000) Solving a Nurse Scheduling Problem with Knapsacks, Networks and Tabu Search. Journal of the Operational Research Society 51 (2000) 7, pp. 825-833

Gilmore, P.C.; Gomory, R.E. (1961) A Linear Programming Approach to the Cutting-Stock Problem, Operations Research 9 (1961) 6, pp. 849-859

Hans, E.W. (2001) Resource Loading by Branch-and-Price Techniques, Twente University Press, Enschede

Keith, G.K. (1979) Operator Scheduling, IIE Transactions 11 (1979) 1, pp. 37-41

Schrijver, A. (2003) Combinatorial Optimization – Polyhedra and Efficiency, Springer-Verlag, Berlin

Van der Veen, E. (2009) Rostering from Staffing Levels: a Branch-and-Price Approach, M. Sc. Thesis Rijksuniversiteit Groningen