First-order Linear Programming in a Column Generation-Based Heuristic Approach to the Nurse Rostering Problem

Petter Strandmark[a], Yi Qu[b,*], Timothy Curtois[c]

[a] *Vårdinnovation, Stockholm, Sweden*
[b] *Newcastle Business School, Northumbria University, Newcastle-upon-Tyne, NE1 8ST, UK*
[c] *Staff Roster Solutions, Sir Colin Campbell Building, Nottingham, NG7 2TU, UK.*

Abstract

A heuristic method based on column generation is presented for the nurse rostering problem. The method differs significantly from an exact column generation approach or a branch and price algorithm because it performs an incomplete search which quickly produces good solutions but does not provide valid lower bounds. It is effective on large instances for which it has produced best known solutions on benchmark data instances. Several innovations were required to produce solutions for the largest instances within acceptable computation times. These include using a fast first-order linear programming solver based on the work of Chambolle and Pock to approximately solve the restricted master problem. A low-accuracy but fast, first-order linear programming method is shown to be an effective option for this master problem. The pricing problem is modelled as a resource constrained shortest path problem with a two-phase dynamic programming method. The model requires only two resources. This enables it to be solved efficiently. A commercial integer programming solver is also tested on the instances. The commercial solver was unable to produce solutions on the largest instances whereas the heuristic method was able to. It is also compared against the state-of-the-art, previously published methods on these instances.

*Corresponding author
Email addresses: petter.strandmark@gmail.com (Petter Strandmark), yi.qu@northumbria.ac.uk (Yi Qu), tim.curtois@staffrostersolutions.com (Timothy Curtois)

Analysis of the branching strategy developed is presented to provide further insights. All the source code for the algorithms presented has been made available on-line for reproducibility of results and to assist other researchers.
Keywords: First-order linear programming, Heuristic, Column Generation, Nurse Rostering.

---

1. Introduction

The nurse rostering problem has received a significant amount of research. One of the earliest papers was in 1972 when Warner and Prawda proposed a mixed integer quadratic programming method for the nurse rostering problem [1]. Even though at this time computers were still in their infancy, the benefits and demand for the automation and optimisation of nurse scheduling were already clear. Optimisation allowed more efficient use of the staff available, providing better levels of healthcare, while automation removed the chore of manually compiling rosters. Additionally, computer-generated rosters are perceived to be fairer rather than the possibly biased approach of a manager. Over the following three decades following the first publication, as computing power increased, so did the scale and richness of problems that could be solved. Methods based on integer programming, such as [2, 3, 4] and [5], and metaheuristics, such as [6, 7, 8, 9, 10, 11] and to a lesser extent constraint programming [12, 13, 14] have all appeared. More unusual approaches such as case based reasoning, for instance, [15, 16, 17] and expert systems have also been tried [18, 19, 20]. Comparing and identifying the best suited methods for the nurse rostering problem have been difficult. This is possibly due to the wide varieties of nurse rostering problems. This in turn may be due to such differences as organisational and national working laws and regulations. There was also a lack of standard benchmark data instances although efforts have been made to improve this [21, 22, 23]. Two competitions have also been used to stimulate research and have generated further insight and advances [24, 25]. For a survey of nurse rosterning papers pre-2004 we refer the readers to the thorough sur-

vey by [26]. After this survey was published, research on the nurse rostering problem has continued. In Section 2, we discuss some of the more recent publications. As will be shown, there has been very little research on solving large instances. Here we consider instances large if the planning horizon is longer than two months, the number of staff is more than one hundred and/or the instance has more than ten shift types. Large instances often occur in practice though. One of the reasons is because nurses can often be assigned to one or more potential facilities. The optimal way to solve this is to combine all the facilities into one instance. However the problem is usually decomposed into smaller, easier to solve instances, which results in sub-optimal solutions. Cyclical schedules are also still very popular despite their obvious disadvantages in dealing with seasonal fluctuations in demand. The reason that they are popular is that they allow staff to know and plan their holidays a long time in advance and they also fairly assign undesirable shifts over longer planning periods. This cannot be done with shorter planning periods such as four weeks. More employers are also offering more flexibility in terms of the number and types of shifts on offer in order to retain staff and accommodate more flexible ways of working. For example, they offer longer or shorter shifts and at different times of the day to try and provide more flexibility to the staff. More shift types, more staff and longer planning periods result in more variables and much larger instances that are much harder to solve. In this paper we present an algorithm that not only is very efficient for smaller and medium length instances but is also able to handle the largest instances. To handle these instances we have had to develop several innovations which will also have applications in other problem domains which require solutions to similarly structured but very large problems (such as airline crew rostering and retail staff scheduling). On the publicly available benchmark instances tested it has produced the best known solutions for some of the largest instances. These instances have planning horizons of 26 weeks and 52 weeks and/or a large number of nurses. This was achieved through several innovative features which are presented in Section 4. The computational results of testing the algorithm on the public benchmark data set are provided in Sec-

tion 5. Finally the paper concludes in Section 6 with some possible ideas for future research directions. Next, in Section 2 we discuss the most recent nurse rostering publications and place our research in context.

2. Literature Review

In this section we examine and discuss the nurse rostering literature. We focus on algorithms and solution approaches although there have also been several papers discussing other issues such as modelling [27, 28, 29]. To place our research in context, the emphasis in this literature review is on mathematical programming, column generation and branch and price methods. However, there has been several metaheuristic approaches recently proposed as well. These include: Variable Neighbourhood Search [30, 31], Scatter Search [32], Ejection Chains [33], Particle Swarm Optimization [34], Harmony Search [35], Simulated Annealing [36], Adaptive Neighbourhood Search [37], Hyperheuristics [38] and more. Metaheuristic methods do have several advantages. They can provide more flexibility in modelling non-linear constraints. They are arguably slightly easier to implement and maintain. They are often robust to problem changes and over varied instance sizes. Often, provable optimal solutions are not strictly required and local optima are sufficient. The phrase "good enough, fast enough" is aptly used. The disadvantages are that they can appear very inefficient when compared against an exact solver that can solve certain instances in acceptable time limits. They also provide no lower bounds and would continue looking for improvements to solutions even when they have found the optimum. To counter some of these disadvantages, hybrid exact/metaheuristic methods have also been successfully applied. For example, a two phase approach by [39] and integer variable fixing methods (also known as Relaxation Induced Neighbourhood Search) have also been proposed such as [40] and [41]. Another hybrid method was recently presented [42] and applied to the same benchmark instances as used here. The results of which are compared to the method presented here in Section 5. These methods have the advantage of utilising the very efficient branch-and-cut

based solvers such as CPLEX [43], [44] and others. An alternative approach is to hybridise constraint programming with integer programming such as [45].

The earliest examples of column generation used to solve nurse rostering are from 1998 [46, 47]. In these papers and all subsequent ones a common approach is used. It involves formulating a restricted master problem (RMP) involving the cover/demand constraints. These are the linking constraints. Negative reduced cost columns for the RMP are generated by solving the pricing problem formulated from the dual costs obtained after adding columns and re-solving the RMP and the constraints for each individual employee's schedule. The columns can be considered as a possible work pattern or schedule for an individual employee. The RMP is solved when no more negative reduced columns can be found. Branching and re-solving is then performed to find an integer feasible solution. Although this is the general framework, there is great scope for variety in the implementation of the algorithms and many heuristics and variations have been proposed to improve the performance. Often designs are included to try and find good solutions in acceptable time limits rather than performing a complete enumeration of the branch and bound tree. In [46] the pricing problem is formulated as a resource constrained shortest path problem. The master problem is solved using the simplex method. The branch and bound tree is searched using depth first and branching is performed on shift assignments (constraint branching [48]). The planning horizon is solved in two week intervals with 41 nurses and seven shift types. [47] use a similar approach and also use Ryan-Foster branching but is able to solve planning horizons of 28 days with 86 nurses and five shift types in reasonable time limits. A later example of a similar approach is given in 2004 by [3] but again the planning horizon was limited to three weeks. In 2010, [49] experimented with different branching strategies to improve the performance of their branch and price algorithm for nurse rostering. These included branching on the original variables in the master problem. They also used a two phase approach to solve the pricing problem. In the first phase fast heuristics based on neighbourhood search are used to try and find any negative reduced cost columns. If this is not possible, it switches

to a dynamic programming method in a second phase. The largest instances they solved have 28 days, 30 nurses and three shift types. Although a dynamic programming method is usually used to solve the pricing problem in branch and price methods for nurse rostering, alternative methods could also be used. For example, an integer programming solver can be used such as CPLEX or Gurobi. [50] demonstrated the use of a constraint programming solver on the pricing problem in their approach but again the maximum sized problems they solved was four weeks. Other ways of speeding up branch and price methods include stabilisation in the column generation such as demonstrated in [23], or using a regression model to try and predict upper bounds in the pricing problem [51]. Another possible improvement is to combine branch and price with a metaheuristic. For example, [52] hybridise branch and price with a metaheuristic by taking the solution generated by branch and price and trying to improve it further using a population-based evolutionary approach. For further reading on column generation in general we refer the reader to [53] and [54].

All of these previous branch and price methods were designed for smaller and medium-sized instances. Solving large instances is very challenging because long planning horizons create many more variables. If there are also large numbers of shift types then the number of constraints and variables increases even more. As we will show, the latest commercial solvers are unable to find even feasible solutions to the largest instances we have tested. In order to tackle these largest instances we have had to use several novel solutions. The main contribution of this paper is solving the RMP using a first-order linear programming solver based on the work of [55] and [56], which will be discussed in Section 4. This is an algorithm that has seen much use in computer vision: for example in sensor fusion for self-driving cars [57], image denoising [58] and motion segmentation [59]. We show in this paper that first-order linear programming is very suitable for the RMP in column generation.

In the next section, we introduce the problem and the benchmark data sets.

3. The Model and Test Data

Our algorithm is tested on a collection of nurse rostering benchmarks data sets publicly available from `http://www.schedulingbenchmarks.org`. The instances range in size from small (two weeks planning horizon, eight staff, one shift type) to very large (52 weeks planning horizon, 150 staff, 32 shift types). Table 1 lists the instances and their sizes.

An integer programming formulation of the problem is given below. The instances start on a Monday and the planning horizon h is a whole number of weeks ($h \bmod 7 = 0$).

Parameters

I	set of employees.
h	number of days in the planning horizon.
D	set of days in the planning horizon = $\{1...h\}$.
W	set of weekends in the planning horizon = $\{1...h/7\}$.
T	set of shift types.
R_t	set of shift types that cannot be assigned immediately after shift type t.
N_i	set of days that employee i cannot be assigned a shift on.
l_t	length of shift type t in minutes.
m_{it}	maximum number of shifts of type t that can be assigned to employee i.
b_i	minimum number of minutes that employee i must be assigned.
c_i	maximum number of minutes that employee i can be assigned.
f_i	minimum number of consecutive shifts that employee i must work.
g_i	maximum number of consecutive shifts that employee i can work.
o_i	minimum number of consecutive days off that employee i can be assigned.
a_i	maximum number of weekends that employee i can work.
q_{idt}	penalty if shift type t is not assigned to employee i on day d.
p_{idt}	penalty if shift type t is assigned to employee i on day d.
s_{dt}	preferred total number of employees assigned shift type t on day d.
u_{dt}	weight if below the preferred cover for shift type t on day d.

v_{dt} weight if exceeding the preferred cover for shift type t on day d.

Decision Variables

x_{idt} 1 if employee i is assigned shift type t on day d, 0 otherwise.

k_{iw} 1 if employee i works on weekend w, 0 otherwise.

y_{dt} total below the preferred cover for shift type t on day d.

z_{dt} total above the preferred cover for shift type t on day d.

Constraints

Maximum shifts per day. An employee is assigned at most one shift each day.

$$\sum_{t \in T} x_{idt} \leq 1, \qquad \forall i \in I, d \in D \tag{1}$$

Shift rotation. A minimum rest is required after a shift. This means that certain shifts cannot follow others. Such as, an early shift cannot follow a late shift.

$$x_{idt} + x_{i(d+1)u} \leq 1, \qquad \forall i \in I, d \in 1...h-1, t \in T, u \in R_t \tag{2}$$

Maximum numbers of shifts of each type that can be assigned to employees. For example, some employees may have contracts which do not allow them to work night shifts. Another employee may work a maximum number of late shifts

$$\sum_{d \in D} x_{idt} \leq m_{it}, \qquad \forall i \in I, t \in T \tag{3}$$

Minimum and maximum work time. The total time worked by each employee must be within a minimum and a maximum number of minutes. These limits can vary depending on whether the employee is full-time or part-time for example.

$$b_i \leq \sum_{d \in D} \sum_{t \in T} l_t x_{idt} \ \leq c_i, \qquad \forall i \in I \tag{4}$$

Maximum consecutive shifts. The maximum number of shifts an employee can work without a day off. For example, part-time employees sometimes do not work as many consecutive shifts as full-time staff.

$$\sum_{j=d}^{d+g_i} \sum_{t \in T} x_{ijt} \leq \ g_i, \qquad \forall i \in I, \ d \in 1...h - g_i \tag{5}$$

Minimum consecutive shifts. This is modelled by preventing every sequence of consecutive shifts below the minimum. For example, if the minimum number of consecutive shifts is four then it must not allow any of the sequences: {*off-on-off, off-on-on-off, off-on-on-on-off*} where *off* is a day without a shift and *on* is a day with a shift assigned.

$$\sum_{t \in T} x_{idt} + \left(s - \sum_{j=d+1}^{d+s} \sum_{t \in T} x_{ijt} \right) + \sum_{t \in T} x_{i(d+s+1)t} > 0,$$
$$\forall i \in I, s \in 1...f_i - 1, d \in 1...h - s + 1 \tag{6}$$

Minimum consecutive days off. This is modelled similarly to the minimum consecutive shifts constraint. For example, if the minimum number of consecutive days off is three then it must not allow any of the sequences: {*on-off-on, on-off-off-on*}.

$$\left(1 - \sum_{t \in T} x_{idt} \right) + \sum_{j=d+1}^{d+s} \sum_{t \in T} x_{ijt} + \left(1 - \sum_{t \in T} x_{i(d+s+1)t} \right) > 0,$$
$$\forall i \in I, s \in 1...o_i - 1, d \in 1...h - s + 1 \tag{7}$$

Maximum number of working weekends. If the employee has a shift on the Saturday or the Sunday it is a working weekend.

$$k_{iw} \leq \sum_{t \in T} x_{i(7w-1)t} + \sum_{t \in T} x_{i(7w)t} \leq 2k_{iw}, \qquad \forall i \in I, w \in W \tag{8}$$

$$\sum_{w \in W} k_{iw} \leq a_i, \qquad \forall i \in I \tag{9}$$

Days off. These are days that employees cannot work because, for example, they have booked the day off for holiday.

$$x_{idt} = 0, \qquad \forall i \in I, d \in N_i, t \in T \tag{10}$$

Cover requirements. These ensure that there are the required number of staff available during each shift.

$$\sum_{i \in I} x_{idt} - z_{dt} + y_{dt} = s_{dt}, \qquad \forall d \in D, t \in T \tag{11}$$

Objective Function

$$\text{minimise} \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} q_{idt} \left(1 - x_{idt}\right) + \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} p_{idt} x_{idt} + \\ + \sum_{d \in D} \sum_{t \in T} y_{dt} u_{dt} + \sum_{d \in D} \sum_{t \in T} z_{dt} v_{dt} \tag{12}$$

The objective function minimises the number of unsatisfied employee shift requests and also minimises under and over staffing. q_{idt} and p_{idt} are the respective weights for requested shifts on and shifts off. For example, a nurse may ask to be assigned a specific shift type on a day. The higher the weight is, the more important it is to satisfy the request. If there is no request, the parameter has the value zero. Variables y_{dt} and z_{dt} are the total nurses below and above the preferred cover for shift type t on day d. The parameters u_{dt} and v_{dt} are weights for the importance of minimising over and under cover.

In all the benchmark instances the weight for minimising under cover is much larger than the other weights. This reflects the real-world requirement of a minimum number of nurses in order to provide patient safety and healthcare. The weights for the requested shifts are relatively low because they represent preferences only. Ideally they will be satisfied but it is not guaranteed because the constraints and minimum level of cover are of greater priority.

4. Algorithm

Our method is inspired by Branch and Price but differs in two major ways. Firstly, we do not solve the column generation exactly. Instead we quickly pro-

duce approximate solutions. Secondly, we employ a diving heuristic instead of exploring the tree in full. Branch and Price is a branch and bound method with each node in the branch and bound tree being solved using column generation. Whenever new columns are added to the RMP, it is re-solved and the updated dual costs are then used to determine if there are any more possible columns to add. If a new column is found, it is added and the RMP is re-solved. After completely solving the RMP, an integer solution must be obtained. It is possible to branch on variables in the RMP but typically constraint branching [48] is used instead. This is also the method used here and is discussed further in Section 4.2. Instead of doing a complete enumeration of the branch and bound tree, we use a depth-first strategy to find near optimal solutions as quickly as possible. After the first feasible solution, the solver is restarted from the beginning without any branches. Because the solver now have access to a very large pool of potential columns, the subsequent solutions will be better, see Figure 7. In this figure we can see that the RMP solution has almost converged before the first branch even (black disc) and progress is very slow. When the solver is restarted after the first integer solution (red disc), it has access to columns generated both before and after branching, and is able to significantly improve the RMP solution during the second attempt. It is possible that after a restart, branching decisions could be repeated. In practice however we did not see this occur. This is because the pool of columns keeps growing which gives a better (different) fractional solution, which results in different decisions.

The overall algorithm can be viewed as an iterative process where each iteration proceeds as follows:

1. If LP objective change was less than δ, branch. (section 4.2)
2. Generate new columns for the RMP (pricing). These are put in a pool of possible columns. (section 4.3)
3. Pick columns from the pool that have negative reduced costs. Columns in the RMP that have been zero for five iterations are removed to keep the problem smaller.

4. Solve the LP. (section 4.1)
5. If the solution is integer, stop.

4.1. Solving the Master Problem

The master problem (RMP) is a linear program of the following form:

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimise}} \quad \sum_{i,j} c_{i,j} x_{i,j} + \sum_i u_i z_i \tag{13}$$

$$\text{subject to} \quad \sum_j x_{i,j} = 1, \quad \forall i \in I. \tag{14}$$

$$\mathsf{B}\boldsymbol{x} + \mathsf{C}\boldsymbol{z} = \boldsymbol{s}, \tag{15}$$

$$\boldsymbol{z} \geq 0, \quad 0 \leq \boldsymbol{x} \leq 1. \tag{16}$$

The variable $x_{i,j}$ determines whether candidate roster j should be assigned to employee i. The candidates are employee-specific since the feasibility of rosters depends on many employee-specific rules. For the same reason, most employees typically have different numbers of candidate rosters. The cost $c_{i,j}$ contains all costs that are roster-specific (sum of requested shifts). The slack variables $\boldsymbol{z}$ adds penalties for not covering all shifts completely.

Constraint (14) requires one roster to be picked per employee (in the integer case, that is; the linear program is a relaxation). The other set of constraints, (15), connects the rosters to the slack variables $\boldsymbol{z}$ encoding the cover requirements from Section 3.

The master problem is typically solved using a simplex solver. The solvers we tested however were not suitable for the largest instances (21–24). Instead a first-order method is used for these instances, which gives a low-accuracy solution very quickly. As we will see below, each iteration of a first-order solver only requires (sparse) matrix-vector multiplications and dot products, which makes their memory usage very attractive.

The RMP can be written as[1]

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & \mathsf{A}\boldsymbol{x} \odot \boldsymbol{b} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \end{aligned} \tag{17}$$

where $\odot$ means element-wise $\leq$, $\geq$, or $=$. The main loop of the algorithm then consists of the following steps [56]:

1. $\boldsymbol{x}_{\text{prev}} \leftarrow \boldsymbol{x}$
2. $\boldsymbol{x} \leftarrow \mathsf{T}(\mathsf{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{c})$
3. $\boldsymbol{x} \leftarrow \max(\boldsymbol{l}, \min(\boldsymbol{u}, \boldsymbol{x}))$ (element-wise projection onto $[\boldsymbol{l}, \boldsymbol{u}]$)
4. $\boldsymbol{y} \leftarrow \Sigma(\mathsf{A}(2\boldsymbol{x} - \boldsymbol{x}_{\text{prev}}) - \boldsymbol{b})$
5. (a) $y_i \leftarrow \max(0, y_i)$ if constraint i is $\leq$
   (b) $y_i \leftarrow \min(0, y_i)$ if constraint i is $\geq$.

The vector $\boldsymbol{y}$ represents dual variables for each constraint. The matrices Σ and T are diagonal, making their application very cheap. They are computed as preconditioners, which is described in [56] and our source code. Here it is important to observe that the only additional storage required is the additional vector $\boldsymbol{x}_{\text{prev}}$, so the memory requirements are practically the same as storing the problem. We choose, however, to also store the transposed matrix A^{T}, for additional optimisation of the inner loop.

The runtime of the algorithm is completely dominated by the two sparse matrix-vector multiplications. We run these in parallel (with Eigen [60]).

4.1.1. Rescaling of $\boldsymbol{c}$.

While [56] described how to precondition A with Σ and T, there is an additional degree of freedom that we take into account. We can observe in the algorithm that $\boldsymbol{x}$ is computed from $\boldsymbol{y}$, and vice versa. It then makes numerical sense if these two vectors were of roughly the same magnitude.

[1] We have now, for the remainder of this section, combined the roster variables $x_{i,j}$ with all slack variables into a single vector $\boldsymbol{x}$ for brevity.

Multiplying $\boldsymbol{c}$ by a (positive) constant will obviously not affect the optimal point of (17). We then pick that constant so that $||\boldsymbol{c}||_2 = ||\boldsymbol{b}||_2$, in order to make $\boldsymbol{x}$ and $\boldsymbol{y}$ roughly the same magnitude.

4.1.2. Stopping criteria

Finding good stopping criteria for first-order solvers can be difficult. But for our purposes, we have found simple criteria to be sufficient. We stop iterating when either:

- We reach 5000 iterations. (See Section 4.4 for how to choose this)
- We observe a relative change of both the primal and dual variables of less than 10^{-6}.

We observed that in practice the first criterion is usually satisfied before the second one.

4.1.3. Warm-start

This first-order solver is easy to warm-start for our purposes. Between solving the RMP, the only things changing are columns being added and removed.

When a column is added, we simply assign it an initial value of $x_{i,j} = 0$. When a column is removed (due to branching), we renormalise the remaining columns for that employee so they still sum to 1 ($\sum_j x_{i,j} = 1$). This ensures that we always start solving with a feasible solution each time we start the RMP.

4.2. Branching

We do not branch in the sense of exploring multiple paths in the decision tree. Instead, we simply search the tree depth-first [61] in order to find a solution quickly. This still requires a substantial amount of work and so much information is generated along the way (in the form of new candidate rosters) that we found it more effective to re-start from the beginning after an integer solution is found rather than exploring other paths in the same tree. This means that when an integer solution is found, we remove all branching constraints and

restore all rosters to the RMP and solve again starting from the root node but with the aim of finding a better solution. At each re-start there are more columns available in the RMP which often leads to a different solution. The reason for this design choice is that exploring other parts of the tree requires too much time for the large instances where most of the computation is spent on simply finding a feasible solution.

Branching is done by branching on shift assignments in the columns for each employee [48]. For each cover constraint k, every column with that shift is summed:

$$r_{i,k} = \sum_{x_{i,j} \text{ has shift } k} x_{i,j}. \tag{18}$$

Because of constraints (16), we have $0 \leq r_{i,j} \leq 1$ as well. These are the variables we are branching on. The full procedure is as follows:

1. Set $t = 0.75$. (The default; see Section 4.4 for how to set it.)
2. Fix all shifts with $r_{i,k} \geq t$.
3. If at least one shift was fixed, stop.
4. Otherwise, if $t > 0.51$, set $t \leftarrow 0.9t$ and try step 2 again. This lowers the threshold successively until something is fixed.
5. If $t \leq 0.51$, pick a column with $x_{i,j} \geq \frac{1}{2}$, fix it to one, and stop.
6. (If no such column was found, fix any value $x_{i,j} \notin \{0, 1\}$ to 1.)

Step 5 seldom happens and step 6, included for completeness, is rarely, if ever, observed in practice. Fixing a whole column $x_{i,j}$ to 1 for an employee effectively removes that employee from further optimisation.

Figures 1 and 2 show the fractional solution number during column generation. These visualisations can prove helpful when tuning the fixation threshold (Section 4.4).

Branching is not performed in every iteration, but only when the RMP seems to have converged (this is a key algorithm parameter in Section 4.4). Figure 5 shows several interesting things about branching, which is represented in the graph as small black discs. Immediately after branching, the objective values

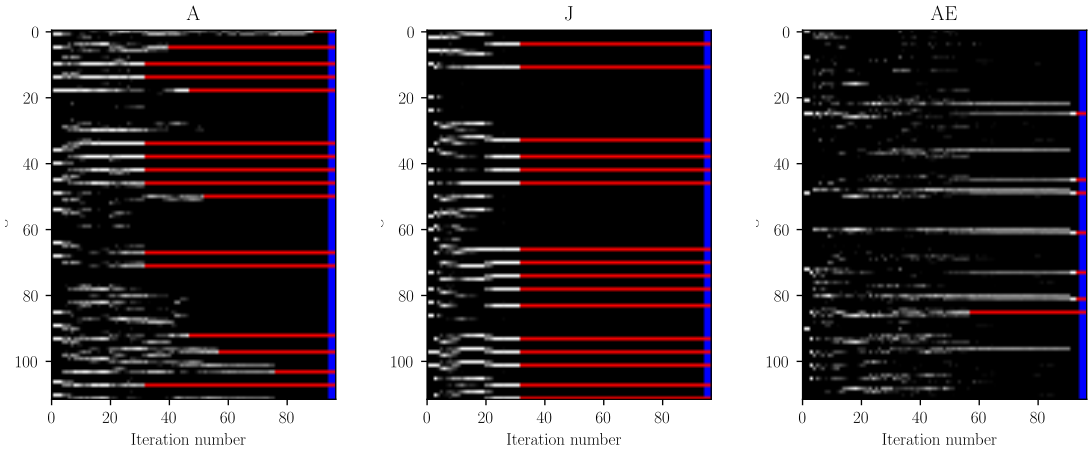

Figure 1: Shift fixing for three employees ("A," "J," and "AE") in Instance 9. Each line in the image is a possible shift. The grayscale shows the fractional assignment of the shift to the employee. Red means that the shift has been fixed to 1. The blue column is the integer solution. Employee "AE" seems to have been problematic for the solver as most shifts were assigned immediately before the integer solution. It is likely that the solver will be more successful on the next re-start when it has a larger pool of rosters to choose from.

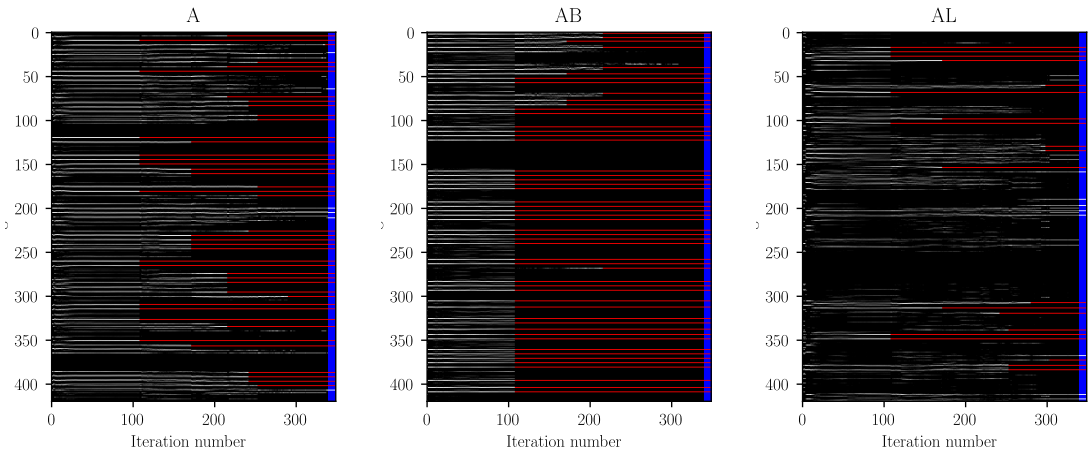

Figure 2: Shift fixing for three employees ("A," "AB," and "AL") in Instance 19. The colours have the same meaning as in Figure 1. Employee "AB" was almost completely assigned in the first round of fixing. This could mean that the solver's strategy was too aggressive.

become much worse. New rosters that fit well around the fixed shifts are however found very quickly. The objective converges much faster after fixing because the pricer finds it easier to generate good new rosters. So branching early is often, unless a unrecoverable mistake is made, beneficial for the speed of the solver.

4.3. Generating New Columns (Pricing)

The pricing problem is modelled as a resource-constrained shortest path problem in a directed acyclic graph and solved using dynamic programming. Figure 3 shows the graph for the smallest instance in the benchmark database.

The constraints from Section 3 are handled in different ways. The first category of constraints are implemented for free by the graph itself:

- Maximum shifts per day—by not having edges between shifts on the same day. See Figure 3, right.
- Shift rotation—by not including edges between certain types of shifts. See Figure 4, left.
- Minimum consecutive days off—by only having edges from shifts to a day-off node some days in the future. See Figure 3, left.
- Days off—by removing work nodes for specific days. See Figure 3, right.
- Branching descisions—we do branching by assigning a specific shift to a specific employee. This is represented in the graph by simply disconnecting all other nodes on that day (Figure 4).

These constraints are implemented in the graph preprocessing phase. The second category are constraints that are implemented as resources in the shortest path problem:

- Minimum and maximum work time.
- Minimum and maximum consecutive shifts.

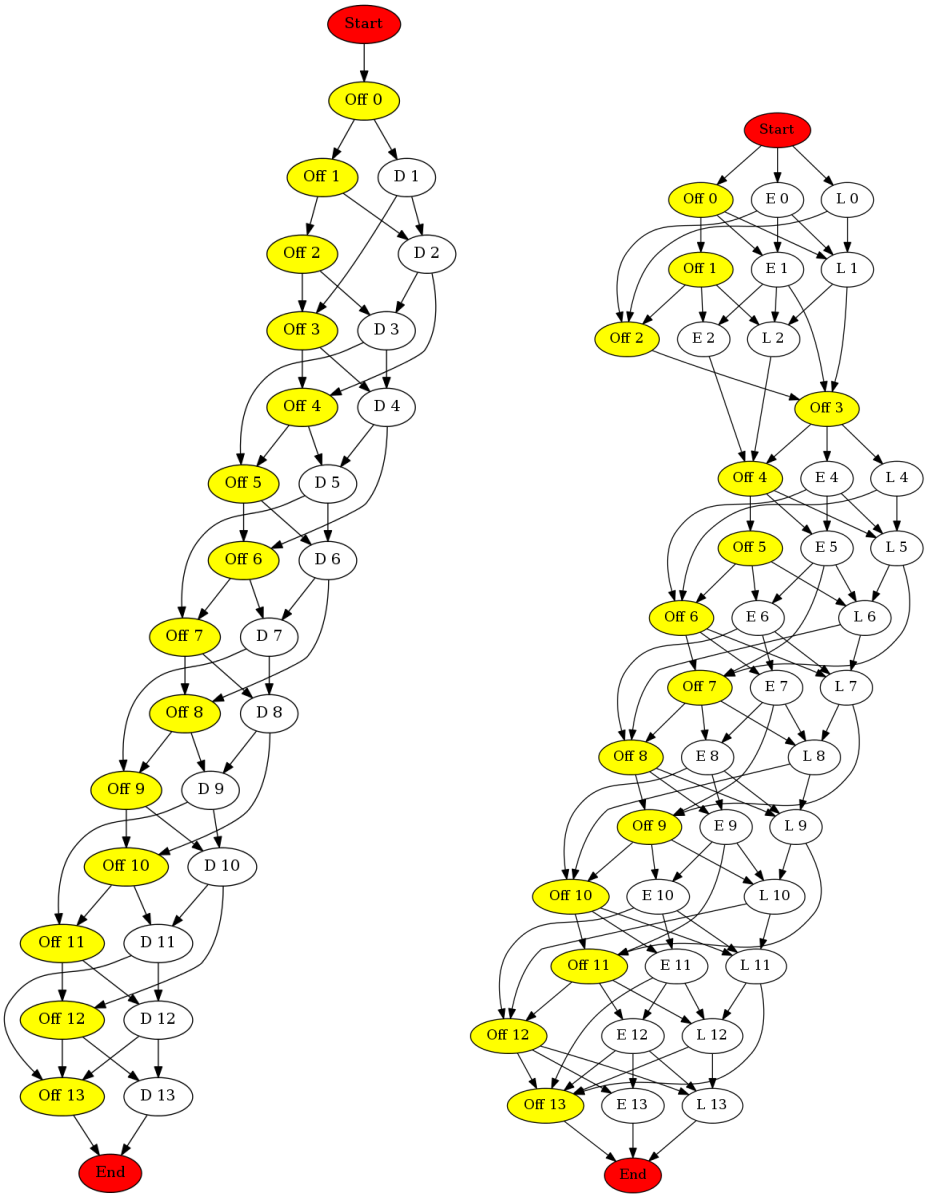

Figure 3: Pricing graph for the smallest instances among the benchmarks. **Left:** Instance 1. There are only two possibilities for each day: off and working (shift "D"). Note that there are no arrows from "D" to "Off" to the same or the next day. This encodes the constraint that for this instance, the minimum consecutive days off are 2, an example of a constraint that we can implement "for free" – by removing edges from the graph. **Right:** Instance 2. There are two possible shifts for each day: "E" and "L."

Optimisations such as dividing all working times by their greatest common divisor are important to reduce memory usage for our dynamic programming solution. The solver finds the optimal solution for the constraints mentioned above.

Finally, we have two constraints not modelled in the graph or added as resources:

- Maximum numbers of shifts of each type that can be assigned to employees.
- Maximum number of working weekends.

These two constraints are handled heuristically by assigning penalties to the solutions produced. If a shortest path is obtained with too many working weekends, we assign large penalties to the weekends exceeding this limit (sorted by their costs so we try to keep the best ones). The graph is then re-solved. The procedure does not guarantee to produce a feasible solution but in practice we observed that it nearly always does. If a feasible solution is not produced then the pricing problem for this nurse is skipped until the next iteration when the duals have been updated and the problem has changed and may be solvable. Because we do not solve to optimality, the overall method does not produce lower bounds. However, we do not solve the RMP until convergence either so we do not require the pricing problem to be solved exactly. These are both design features aimed at finding good upper bounds more quickly, particularly for the largest instances, rather than finding a lower bound.

4.3.1. Preprocessing the graph

Constraints are often implemented by removing edges from the graph. This often means that nodes or other edges become unreachable. Before solving, we preprocess the graph to only include edges and nodes that can be part of a complete path. Figure 4 shows this process, where nodes that cannot be reached from the start or cannot reach the end are removed.

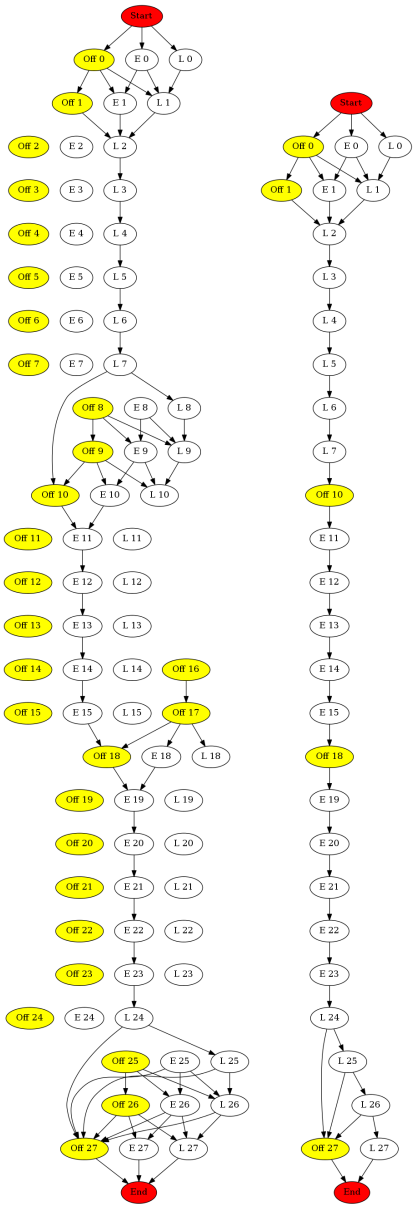

Figure 4: Pricing graph for Instance 5 from the benchmark (chosen because its single shift type reduces clutter). Many shifts have been fixed during branching. **Left:** The graph after the branching has fixed many shifts to 1 for the employee. This means removing edges from the graph that leads to shifts that are now impossible. **Right:** The graph after fixing and preprocessing. Many nodes are now redundant and have been removed. Nodes in the long straight segments can be further merged.

4.4. Algorithm parameters

The key parameters that affect the performance of the algorithm are:

- **Number of solutions**. After we have obtained a final integer solution we re-start from the beginning, only keeping the columns we have generated so far in a pool of possible columns to add to the RMP. As we show in e.g. Figure 7, doing this repeatedly usually gives better solutions. The first solution requires the longest time to compute, because all columns have to be generated by the pricing routine.

- **RMP objective change before branching** determines whether to branch as described in Section 4.2. If the objective change between a complete round of branching is less than δ, we run the branching routine. In our tests, we used a value of $\delta = 1$, except for the largest instances 23 and 24, where we used 30 and 100, respectively. Setting this parameter lower will require more time and give better solutions. On the largest two instances though it was necessary to increase it in order to find solutions in a reasonable computation time.

- **The fixation threshold** t is the criteria for determining whether to fix a shift to an employee during branching. If the sum of all columns for an employee exceeds this threshold for a specific shift, this shift is fixed for that employee. We used $t = 0.75$ for this parameter except for the largest instance, where we used 0.52. Setting this parameter higher will give fewer fixations each round and the algorithm will take longer to converge. Because fewer fixes are made each round, the solver also has more opportunity to avoid really bad fixations and the end result may become better.

  Figures 7 and 8 show the difference where all other parameter were held constant and the fixation threshold changed from 0.75 to 0.99.

- **Maximum iterations of the first-order solver**. The runtime is dominated by two factors: pricing and solving the RMP (other things that

do not require much time include branching decisions and determining columns to remove). If the time required to solve the RMP is very small compared to pricing, allowing more iterations for increased accuracy may be beneficial, and vice versa. We have used a limit of 5000 iterations of the main loop in Section 4.1.

See Table 7 for an illustration of the trade-off of these parameters.

All code is available at `https://github.com/PetterS/monolith`

5. Experimental Results

The algorithm was tested on the benchmark nurse rostering data sets available at `http://www.schedulingbenchmarks.org`.

In an initial implementation of the algorithm the open-source CLP linear programming solver [62] was used to solve the RMP. However on the largest instances it was too slow to be a practical option. This motivated the implementation of the Chambolle-Pock method.

To compare the performance of the Chambolle-Pock method with alternative linear programming solvers, Table 2 lists the time required by different methods of IBM's CPLEX Optimiser 12.9.0 [43] to solve the RMP produced after 100 iterations of column generation on Instance 24 (the largest instance). As can be seen, CPLEX's primal and dual simplex methods are also too slow on these largest instances. The barrier method is a lot faster though and could possibly be a viable alternative. However it is still slower than the Chambolle-Pock method. This test shows the importance of solving the RMP efficiently. It has to be solved after adding new columns and could be potentially called thousands of times during the overall algorithm. If it is not solved efficiently this would become a severe bottle-neck in the algorithm.

The results of applying CPLEX 12.9.0 to all the instances are given in Table 3. The formulation used is the integer programming formulation given in Section 3 known as the "time-indexed formulation". CPLEX was given a maximum computation time of one hour and the best lower bound and integer solution

found after one hour are reported. The MIPEmphasis parameter for CPLEX was set to finding feasible solutions as quickly as possible to more closely reflect the emphasis of our algorithm. Ten of the smaller and medium sized instances are solved by CPLEX to optimality. Integer feasible solutions are found for six other instances. Lower bounds are produced for all instances except the third largest. On eight of the instances (including the six largest) no solution is produced at all. The tests were then repeated on the largest instances but this time allowing much longer computation times. The results are shown in Table 4. The lower bounds were improved but still no feasible solution was found.

These preliminary experiments and the results in [63] show that the first twelve instances can all be solved within one hour using a commercial solver. The larger instances appear to be much more challenging though. Therefore our analysis and focus is on the twelve larger instances which appear to be harder to solve.

Table 5 contains the results of testing the algorithm on the twelve largest benchmark instances. The same parameter values as described in Section 4 were used on all instances apart from the two largest instances (23 and 24). For these two largest instances it was necessary to change two parameters which adjust the speed/solution quality trade-off in order to produce solutions in a reasonable computation time. These parameter values are also listed in Section 4. For comparison purposes the results of a CPLEX 12.9.0 are included as well as the best results from the literature. The best results in the literature include a network flow based integer programming formulation solved using Gurobi ([63]), a hybrid integer programming and variable neighbourhood search ([45]) and an ejection chain method ([23]).

Comparing against the time-indexed formulation solved using CPLEX, we find a better solution on nine out of twelve instances. Comparing against the network flow based formulation, we find a better solution on six out of the twelve instances. On the instances where it is out-performed it uses significantly less time. Compared to the Hybrid VNS method, we find a better solution on four of the instances. On ten of the twelve instances we use less computation time,

which makes a direct comparison difficult. Unfortunately we discovered that the solution for instance 23 (3794) reported for the Hybrid VNS method is not possible and a mistake must have been made in that paper. By removing constraints 5., 6. and 7. (max consecutive shifts, min consecutive shifts and min consecutive days off) from the IP model and solving the resulting relaxation using CPLEX, a relaxed solution and lower bound of 16887 is produced. Therefore the solution of 3794 cannot be possible. We outperform the ejection chain method on all but one instance.

On Instance 13, which has a short planning horizon but a large number of staff (120 nurses) we find the best solution compared to all methods and in under ten minutes. On Instances 21 and 24 we also find the best known but on Instance 24 we require a long computation time.

Table 6 contains further analysis of applying our column generation algorithm to the five largest instances. On Instance 20 a solution was produced within five minutes. Better solutions could also be found when given longer solving times and further improved when combined with the local search. On Instances 21 and 22, a feasible solution was found in less than thirty minutes. On Instances 23 and 24, more than one hour was required to find a feasible solution, and significantly more time to improve it.

Further discussion and insight into the progress of the algorithm is given in Figures 6 to 10. Figures 9 and 10 also provide a comparison of using a traditional simplex LP solver instead of the first-order method on the largest possible instance that could still be feasibly solved using a traditional method. The lower accuracy of the first order method is visible in comparison but it can also be seen that the integer solutions are similar in value and the time required is approximately the same.

Note we do not report any lower bounds because they are not produced. The branching starts before convergence is allowed to complete and so before an accurate lower bound can be calculated. An example of this can also be seen by comparing Figures 9 and 10. Because the algorithm is not focussed on providing a lower bound but instead in finding good upper bounds in reasonable

time limits, it means that a heuristic approach to solve the pricing problem was also suitable.

5.1. Local Search

We have also experimented with using local search in order to evaluate how close our solutions are to a "local optimum." When optimising an already feasible solution with local search, we solve subproblems of the IP described in detail in Section 3. The subproblems consist of one week at a time and a small, random, group of employees at a time. It is similar to the variable fixing approaches in [40] and [41] where all the variables outside the selected week and nurses are considered fixed to their values in the current solution and a much smaller sub-problem is formed from the remaining variables.

These small IPs are solved quickly. Their solutions then replace the respective configurations in the large solution. Because all constraints are taken into account, the new solution will have a lower or equal objective value.

We run the local search until no improvements are found. The results are shown in Table 6. For some problems there is still room for improvement. We also found that obtaining a good (or even feasible) solution with this local search on its own was very unsuccessful.

6. Conclusions

The paper presents an efficient, heuristic, column generation based method for the nurse rostering problem which is particularly effective on very large instances. It is an heuristic method which focusses on producing good solutions instead of producing valid lower bounds. Unlike traditional column generation and branch and price, it performs an incomplete search, with the aim of finding good solutions more quickly. It approximately solves the restricted master problem and uses a diving heuristic instead of a complete search. It therefore does not produce lower bounds. It has been tested on a set of standard benchmark data sets and has found best known solutions for some of the largest

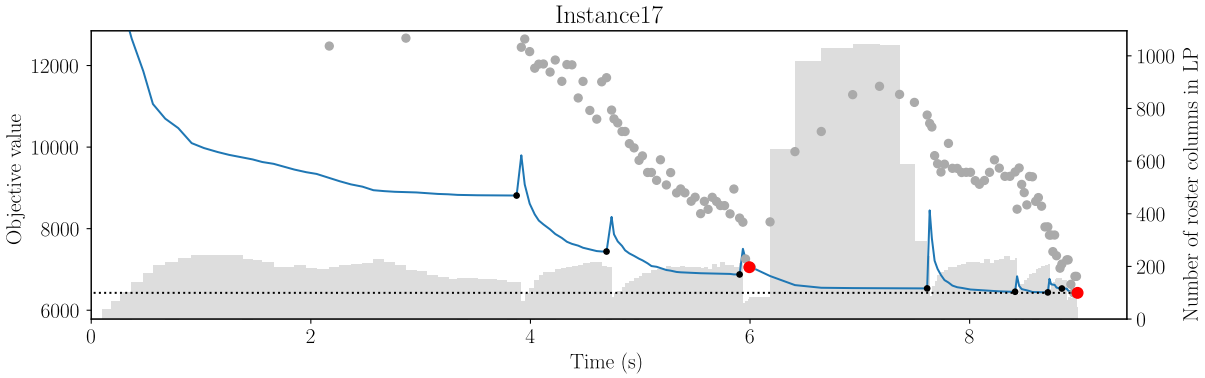

Figure 5: The convergence until the second integer solution in Instance 17. The blue line is the RMP solution. Black discs (●) show times of branching. Red (●) and gray (●) discs are feasible integer solutions, where the gray ones have been rounded from the current fractional solution. The gray bars in the background show the number of columns in the RMP. This number goes up when pricing and goes down when branching (colums that disagree with the fixation choices are removed).
In the short-term, branching results in a worse fractional solution, but it quickly recovers and converges much faster to a better objective function value.

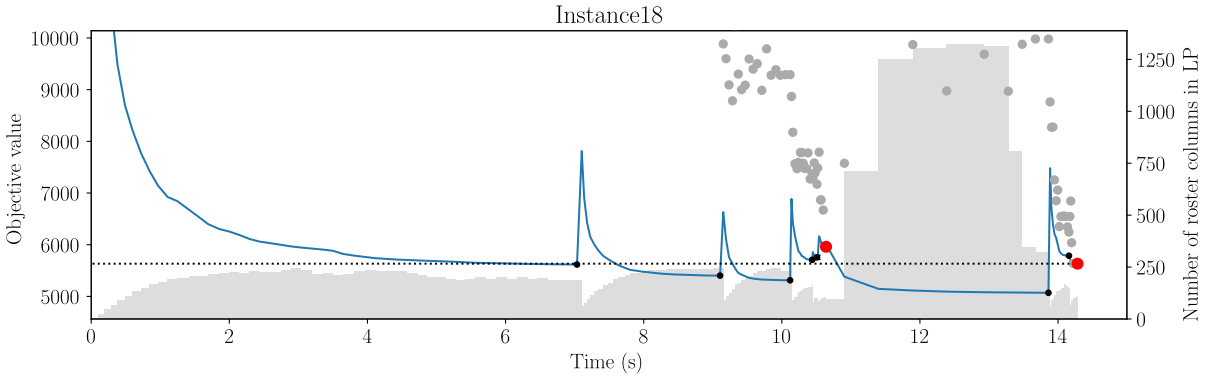

Figure 6: The convergence until the second integer solution in Instance 18. The symbols have the same meaning as in Figure 5. We can clearly see that branching (fixing shifts to employees) results in a significantly worse RMP solution, but that additional pricing rounds recover from this by generating rosters that fix nicely in the partially fixed structure.

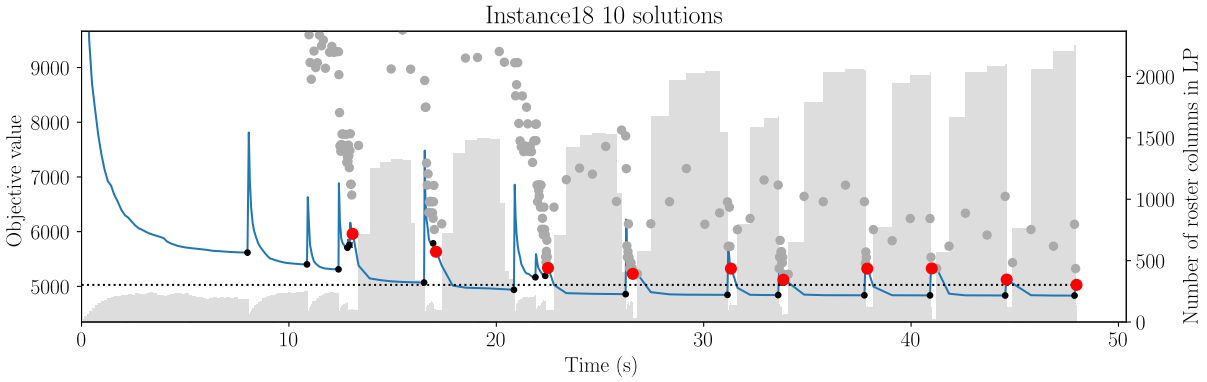

Figure 7: Same as Figure 6, but showing 10 integer solutions. There is a clear progression showing increasingly better integer solutions.

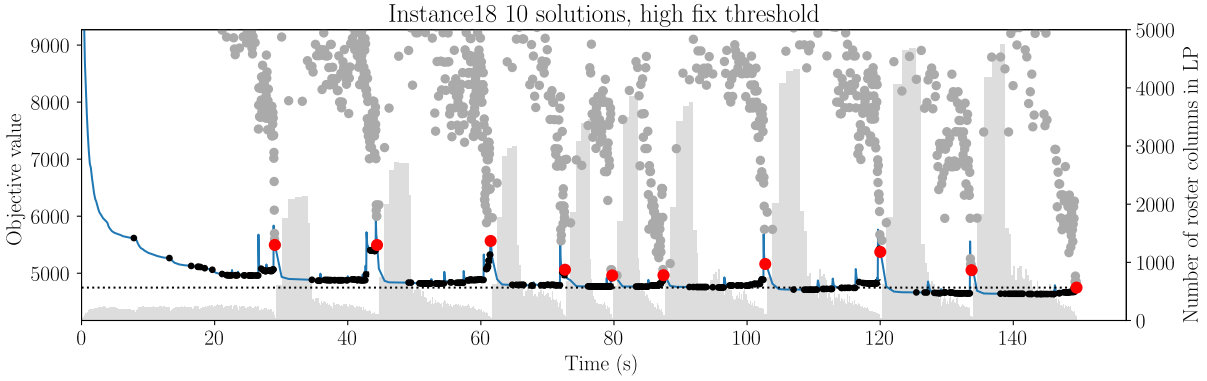

Figure 8: Same as Figures 6 and 7, but using a very high (0.99) threshold for fixing shifts (see Section 4.4). The solutions become better, but significantly more time is required (note the scale on the x-axis compared to figure 7). Many more separate fixation events are now required until an integer solution.

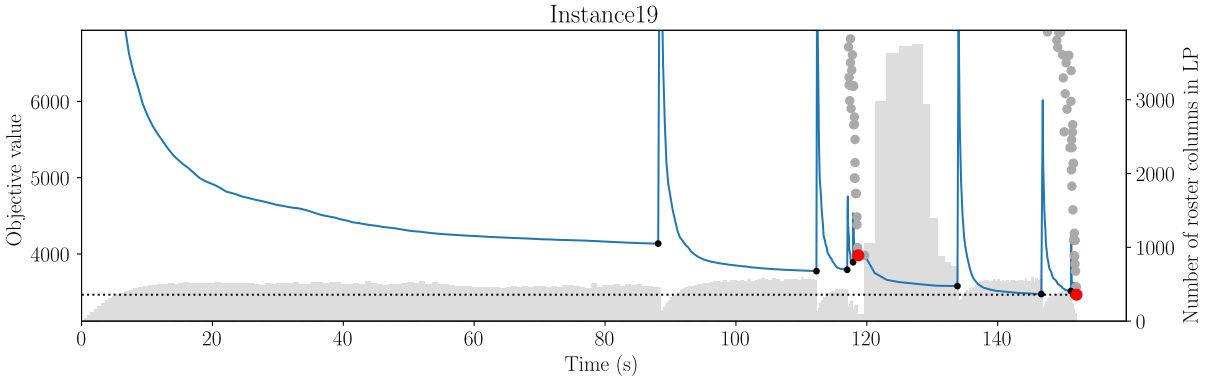

Figure 9: The convergence until the second integer solution in Instance 19. The symbols have the same meaning as in Figure 5. The objective function value takes a large hit when branching, but the pricing quickly finds new columns that fit well into the remaining ones. The integer solutions rounded from the fractional solution (•) are essentially worthless until immediately before the solver has branched all the way to an integer solution.

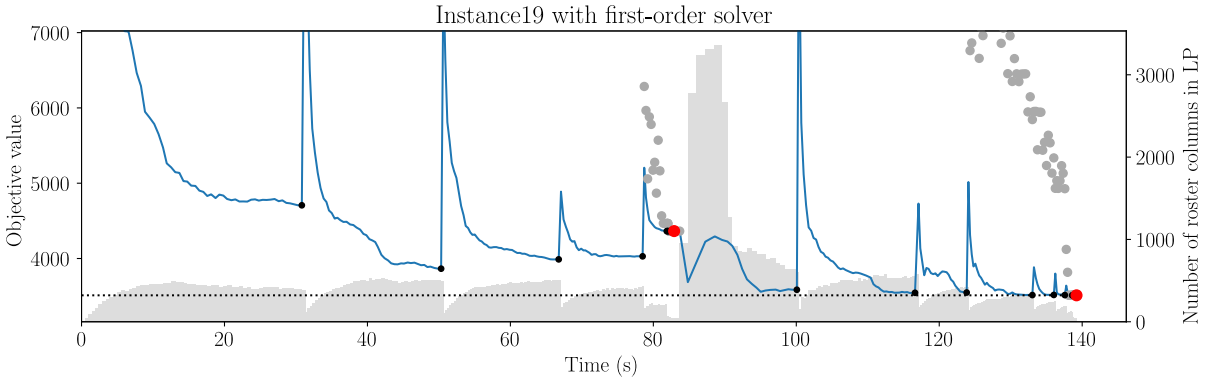

Figure 10: Convergence when Instance 19 is configured to use the first-order solver. This instance is still small enough that using a simplex solver works very well. The runtime is almost the same as in Figure 9. Some inaccuracies from the first-order solver can be seen as wiggles in the blue line. With a perfect LP solver, the RMP objective can never go up, except when fixing. After the first integer solution (•), the solver is reset with 2500 new columns. The first-order solver then has problems for a while (seen as the huge dip right after), but recovers 4 seconds later.

instances. A commercial solver was also tested on the benchmark set but it was unable to find a feasible solution on the largest instances. The algorithm was also compared against the best known solutions in the literature and was able to further improve on the best knowns for some instances. The algorithm uses several innovations but one of the most important was to use a very fast linear programming solver for the restricted master problem. The solver is based on the algorithm of [55] and [56]. Using Chambolle-Pock for column generation is novel. Approximate methods that trade accuracy for speed are very suitable for column generation because the main problem does not need high accuracy. It only needs a good enough solution in order to generate columns better than the current ones. For this problem and others which may have very large restricted master problems, if good upper bounds are required in reasonable computation times, first-order methods for solving the restricted master problem are shown to be an effective option. When compared against CPLEX's simplex optimizer it was able to solve the largest restricted master problem significantly faster. The pricing problem was solved using a dynamic programming method on a resource constrained shortest path problem. It is a two phase approach which handles some constraints in the graph structure, some as resources and some heuristically in the second phase. This innovation made the pricer very fast, which was key to being able to produce solutions for the largest instances. It is very likely that this idea could also be used in column generation approaches for other problem domains. A novel branching strategy is used to drive the search towards integer solutions quickly and effectively. Again, this branching strategy and insight is a general contribution in column generation and can be applied in other problem domains. We also provide analysis of the progress of the branching algorithm, measuring the change in objective function and integer solutions as branching decisions are made and columns are added. This assisted in the designing of the branching rules and should provide interesting insight to other researchers also. The source code for all the algorithms presented in the paper is made available for other researchers to use and recreate the results presented

here (`https://github.com/PetterS/monolith`).[2]

While we have focused on a public benchmark nurse rostering problems with long planning horizons and/or large numbers of staff, there is no shortage of challenging problems in industry with shorter (one month) horizons. For example, in the airline industry and the planning of crew rosters, it is often necessary to schedule hundreds of staff over many flights. This results in very large restricted master problems in branch and price methods, for which the proposed methods here would be very suitable. Another domain with naturally large problems is retail staff scheduling. Whereas nurse rostering requires the assignment of shifts over planning horizons split by days, retailers usually schedule staff over planning horizons split into fifteen minute time intervals. Demand for different work activities is given for every fifteen minute interval and shifts rosters must be produced that satisfy the activity demand at every fifteen minute interval, as well as satisfying other constraints such as total work hours, rest time, maximum shift lengths, shifts start times and more. If even only one week is scheduled at a time, this results in a planning horizon of seven days multiplied by 96 fifteen minute intervals per day which equals a planning horizon of length of 672 intervals. Attempting to solve this using column generation or branch and price would create very large restricted master problems and a fast first-order method would be suitable option.

There are also some other options for future research directions. It is possible the algorithm described for branching in Section 4.2 could be improved. It does not look at where in the roster the candidate shifts occur, nor to whom they belong—only at their numerical value. Taking the problem structure into account here could prove beneficial. There is also a large body of research about heuristics and local-search algorithms for scheduling problems. Given that after many iterations, we have a very large number of feasible rosters, combining and improving them with heuristics could be successful. This research has

[2] Also available for execution in the browser at `https://www.strandmark.net/wasm/shift_scheduling_colgen_page.html`.

been released as an open-source platform for experimentation to allow other researchers to pursue this in the future.

7. Acknowledgements

Thank you to the reviewers for their valuable comments and suggestions on the initial versions of this paper.

[1] D. M. Warner, J. Prawda, A mathematical programming model for scheduling nursing personnel in a hospital, Management Science 19 (4-part-1) (1972) 411–422.

[2] D. M. Warner, Scheduling nursing personnel according to nursing preference: A mathematical programming approach, Operations Research 24 (5) (1976) 842–856.

[3] P. Eveborn, M. Rönnqvist, Scheduler–a system for staff planning, Annals of Operations Research 128 (1-4) (2004) 21–45.

[4] H. H. Millar, M. Kiragu, Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming, European journal of operational research 104 (3) (1998) 582–592.

[5] H. E. Miller, W. P. Pierskalla, G. J. Rath, Nurse scheduling using mathematical programming, Operations Research 24 (5) (1976) 857–870.

[6] A. Meisels, A. Schaerf, Modelling and solving employee timetabling problems, Annals of Mathematics and Artificial Intelligence 39 (1-2) (2003) 41–59.

[7] G. M. Thompson, A simulated-annealing heuristic for shift scheduling using non-continuously available employees, Computers & Operations Research 23 (3) (1996) 275–288.

[8] A. Ikegami, A. Niwa, A subproblem-centric model and approach to the nurse scheduling problem, Mathematical programming 97 (3) (2003) 517–541.

[9] K. A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European journal of operational research 106 (2-3) (1998) 393–407.

[10] X. Cai, K. Li, A genetic algorithm for scheduling staff of mixed skills under multi-criteria, European Journal of Operational Research 125 (2) (2000) 359–369.

[11] E. K. Burke, P. Cowling, P. De Causmaecker, G. Vanden Berghe, A memetic approach to the nurse rostering problem, Applied intelligence 15 (3) (2001) 199–214.

[12] S. J. Darmoni, A. Fajner, N. Mahe, A. Leforestier, M. Vondracek, O. Stelian, M. Baldenweck, Horoplan: computer-assisted nurse scheduling using constraint-based programming., Journal of the Society for Health Systems 5 (1) (1995) 41–54.

[13] G. Weil, K. Heus, P. Francois, M. Poujade, Constraint programming for nurse scheduling, IEEE Engineering in medicine and biology magazine 14 (4) (1995) 417–422.

[14] A. Meisels, E. Gudes, G. Solotorevsky, Employee timetabling, constraint networks and knowledge-based rules: A mixed approach, in: International Conference on the Practice and Theory of Automated Timetabling, Springer, 1995, pp. 91–105.

[15] G. R. Beddoe, S. Petrovic, Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering, European Journal of Operational Research 175 (2) (2006) 649–671.

[16] G. Beddoe, S. Petrovic, J. Li, A hybrid metaheuristic case-based reasoning system for nurse rostering, Journal of Scheduling 12 (2) (2009) 99.

[17] G. Beddoe, S. Petrovic, Enhancing case-based reasoning for personnel rostering with selected tabu search concepts, Journal of the Operational Research Society 58 (12) (2007) 1586–1598.

[18] L. D. Smith, A. Wiggins, A computer-based nurse scheduling system, Computers & Operations Research 4 (3) (1977) 195–212.

[19] P. Bell, G. Hay, Y. Liang, A visual interactive decision support system for workforce (nurse) scheduling, INFOR: Information Systems and Operational Research 24 (2) (1986) 134–145.

[20] J. Chen, T. W. Yeung, Hybrid expert-system approach to nurse scheduling., Computers in nursing 11 (4) (1993) 183–190.

[21] P. Brucker, E. K. Burke, T. Curtois, R. Qu, G. Vanden Berghe, A shift sequence based approach for nurse scheduling and a new benchmark dataset, Journal of Heuristics 16 (4) (2010) 559–573.

[22] M. Vanhoucke, B. Maenhout, On the characterization and generation of nurse scheduling problem instances, European Journal of Operational Research 196 (2) (2009) 457–467.

[23] E. K. Burke, T. Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research 237 (1) (2014) 71–81.

[24] S. Ceschia, N. Dang, P. De Causmaecker, S. Haspeslagh, A. Schaerf, The second international nurse rostering competition, Annals of operations research 274 (1-2) (2019) 171–186.

[25] S. Haspeslagh, P. De Causmaecker, A. Schaerf, M. Stølevik, The first international nurse rostering competition 2010, Annals of Operations Research 218 (1) (2014) 221–236.

[26] E. K. Burke, P. De Causmaecker, G. Vanden Berghe, H. Van Landeghem, The state of the art of nurse rostering, Journal of scheduling 7 (6) (2004) 441–499.

[27] P. Smet, B. Bilgin, P. De Causmaecker, G. Vanden Berghe, Modelling and evaluation issues in nurse rostering, Annals of Operations Research 218 (1) (2014) 303–326.

[28] P. Smet, P. De Causmaecker, B. Bilgin, G. Vanden Berghe, Nurse rostering: a complex example of personnel scheduling with perspectives, in: Automated Scheduling and Planning, Springer, 2013, pp. 129–153.

[29] S. Petrovic, G. Vanden Berghe, A comparison of two approaches to nurse rostering problems, Annals of Operations Research 194 (1) (2012) 365–384.

[30] E. K. Burke, T. Curtois, G. Post, R. Qu, B. Veltman, A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem, European Journal of Operational Research 188 (2) (2008) 330 – 341. doi:10.1016/j.ejor.2007.04.030.
URL http://www.sciencedirect.com/science/article/pii/S0377221707004390

[31] B. Bilgin, P. De Causmaecker, B. Rossie, G. Vanden Berghe, Local search neighbourhoods for dealing with a novel nurse rostering model, Annals of Operations Research 194 (1) (2012) 33–57.

[32] E. K. Burke, T. Curtois, R. Qu, G. V. Berghe, A scatter search methodology for the nurse rostering problem, Journal of the Operational Research Society 61 (11) (2010) 1667–1679. arXiv:https://doi.org/10.1057/jors.2009.118, doi:10.1057/jors.2009.118.
URL https://doi.org/10.1057/jors.2009.118

[33] E. K. Burke, T. Curtois, R. Qu, G. Vanden Berghe, A time predefined variable depth search for nurse rostering, INFORMS Journal on Computing 25 (3) (2013) 411–419.

[34] T.-H. Wu, J.-Y. Yeh, Y.-M. Lee, A particle swarm optimization approach with refinement procedure for nurse rostering problem, Computers & Operations Research 54 (2015) 52 – 63. doi:10.1016/j.cor.2014.08.016.
URL http://www.sciencedirect.com/science/article/pii/S0305054814002263

[35] M. Hadwan, M. Ayob, N. R. Sabar, R. Qu, A harmony search algorithm for nurse rostering problems, Information Sciences 233 (2013) 126 – 140. doi:10.1016/j.ins.2012.12.025.
URL http://www.sciencedirect.com/science/article/pii/S0020025513000170

[36] Z. Liu, Z. Liu, Z. Zhu, Y. Shen, J. Dong, Simulated annealing for a multi-level nurse rostering problem in hemodialysis service, Applied Soft Computing 64 (2018) 148 – 160. doi:10.1016/j.asoc.2017.12.005.
URL http://www.sciencedirect.com/science/article/pii/S1568494617307196

[37] Z. Lü, J.-K. Hao, Adaptive neighborhood search for nurse rostering, European Journal of Operational Research 218 (3) (2012) 865–876.

[38] S. Asta, E. Özcan, T. Curtois, A tensor based hyper-heuristic for nurse rostering, Knowledge-Based Systems 98 (2016) 185 – 199. doi:10.1016/j.knosys.2016.01.031.
URL http://www.sciencedirect.com/science/article/pii/S0950705116000514

[39] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, E. Housos, A systematic two phase approach for the nurse rostering problem, European Journal of Operational Research 219 (2) (2012) 425 – 433. doi:10.1016/j.ejor.2011.12.042.
URL http://www.sciencedirect.com/science/article/pii/S0377221711011362

[40] F. Della Croce, F. Salassa, A variable neighborhood search based matheuristic for nurse rostering problems, Annals of Operations Research 218 (1) (2014) 185–199.

[41] H. G. Santos, T. A. Toffolo, R. A. Gomes, S. Ribas, Integer programming techniques for the nurse rostering problem, Annals of Operations Research 239 (1) (2016) 225–251.

[42] E. Rahimian, K. Akartunalı, J. Levine, A hybrid integer and constraint programming approach to solve nurse rostering problems, Computers & Operations Research 82 (2017) 83–94.

[43] IBM, Cplex optimizer reference manual (2018).
URL https://www.ibm.com/analytics/cplex-optimizer

[44] L. Gurobi Optimization, Gurobi optimizer reference manual (2018).
URL http://www.gurobi.com

[45] E. Rahimian, K. Akartunalı, J. Levine, A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems, European Journal of Operational Research 258 (2) (2017) 411–423.

[46] B. Jaumard, F. Semet, T. Vovor, A generalized linear programming model for nurse scheduling, European journal of operational research 107 (1) (1998) 1–18.

[47] A. J. Mason, M. C. Smith, A nested column generator for solving rostering problems with integer programming, in: International conference on optimisation: techniques and applications, Curtin University of Technology Perth, Australia, 1998, pp. 827–834.

[48] D. M. Ryan, B. A. Foster, An integer programming approach to scheduling, in: Computer scheduling of public transport urban passenger vehicle and crew scheduling, North-Holland, 1981, pp. 269–280.

[49] B. Maenhout, M. Vanhoucke, Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem, Journal of Scheduling 13 (1) (2010) 77–93.

[50] F. He, R. Qu, A constraint programming based column generation approach to nurse rostering problems, Computers & Operations Research 39 (12) (2012) 3331 – 3343. doi:10.1016/j.cor.2012.04.018.
URL http://www.sciencedirect.com/science/article/pii/S0305054812000986

[51] R. Václavík, A. Novák, P. Šůcha, Z. Hanzálek, Accelerating the branch-and-price algorithm using machine learning, European Journal of Operational Research 271 (3) (2018) 1055 – 1069. doi:10.1016/j.ejor.2018.05.046. URL http://www.sciencedirect.com/science/article/pii/S0377221718304570

[52] J. Inafune, S. Watanabe, M. Okudera, New approach combining branch and price with metaheuristics to solve nurse scheduling problem., JACIII 21 (7) (2017) 1251–1261.

[53] M. E. Lübbecke, J. Desrosiers, Selected topics in column generation, Operations research 53 (6) (2005) 1007–1023.

[54] G. Desaulniers, J. Desrosiers, M. M. Solomon, Column generation, Vol. 5, Springer Science & Business Media, 2006.

[55] A. Chambolle, T. Pock, A first-order primal-dual algorithm for convex problems with applications to imaging, Journal of mathematical imaging and vision 40 (1) (2011) 120–145.

[56] T. Pock, A. Chambolle, Diagonal preconditioning for first order primal-dual algorithms in convex optimization, in: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, 2011, pp. 1762–1769.

[57] C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, P. Furgale, T. Sattler, M. Pollefeys, 3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection, Image and Vision Computing 68 (2017) 14–27.

[58] L. Condat, A direct algorithm for 1-d total variation denoising, IEEE Signal Processing Letters 20 (11) (2013) 1054–1057.

[59] P. Ochs, J. Malik, T. Brox, Segmentation of moving objects by long term video analysis, IEEE transactions on pattern analysis and machine intelligence 36 (6) (2014) 1187–1200.

[60] G. Guennebaud, B. Jacob, et al., Eigen v3, http://eigen.tuxfamily.org (2010).

[61] C. Barnhart, C. A. Hane, P. H. Vance, Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems, Operations Research 48 (2) (2000) 318–326.

[62] R. Lougee-Heimer, The common optimization interface for operations research: Promoting open-source software in the operations research community, IBM Journal of Research and Development 47 (1) (2003) 57–66.

[63] P. Smet, Constraint reformulation for nurse rostering problems, in: PATAT 2018 twelfth international conference on the practice and theory of automated timetabling, Vienna, August, 2018, pp. 69–80.

Table 1: Test Instances

Instance	Planning horizon(weeks)	Staff	Shift types
1	2	8	1
2	2	14	2
3	2	20	3
4	4	10	2
5	4	16	2
6	4	18	3
7	4	20	3
8	4	30	4
9	4	36	4
10	4	40	5
11	4	50	6
12	4	60	10
13	4	120	18
14	6	32	4
15	6	45	6
16	8	20	3
17	8	32	4
18	12	22	3
19	12	40	5
20	26	50	6
21	26	100	8
22	52	50	10
23	52	100	16
24	52	150	32

Table 2: Linear programming solvers tested on the RMP for Instance 24 after 100 iterations of the column generation.

Solver	Time(s)
Chambolle-Pock	10
CPLEX Barrier Method	51
CPLEX Primal simplex Optimiser	1414
CPLEX Dual Simplex Optimiser	> 28800

Table 3: CPLEX 12.9.0 tested on the benchmark instances.

Instance	Lower Bound	Solution	Time(s)
1	607	607	0.59
2	828	828	0.97
3	1001	1001	10.08
4	1716	1716	46.98
5	1143	1143	101.91
6	1950	1950	60.71
7	1056	1056	1019.73
8	1285	1320	3600.04
9	247	440	3600.06
10	4631	4631	916.92
11	3443	3443	83.96
12	4040	4040	2831.41
13	1347	-	3600.11
14	1276	1284	3600.02
15	3810	7789	3600.06
16	3215	3230	3600.03
17	5730	-	3600.06
18	4363	4659	3600.05
19	2944	-	3600.04
20	3709	-	3600.13
21	9637	-	3600.20
22	-	-	3600.45
23	146	-	3601.30
24	1133	-	3631.30

Table 4: CPLEX 12.9.0 tested on the largest benchmark instances with longer computation times.

Instance	Lower Bound	Solution	Time(h)
20	4764	-	24
21	20943	-	24
22	23895	-	24
23	2772	-	48
24	1133	-	72

Table 5: Comparison against previously published best knowns.

	CPLEX 12.9.0		Flow Based IP [63]		Hybrid IP+VNS [45]		Ejection chain [23]		This paper	
Instance	UB	Time (s)	UB	Time (s)	UB	Time (s)	UB	Time (s)	UB	Time (s)
13	-	3600.1	1652	18000.3	1905	3600	3037	3600	1356	452
14	1284	3600.0	1278	744.7	1279	3600	1847	3600	1870	17
15	7789	3600.1	3853	18000.1	3928	3600	5935	3600	4271	61
16	3230	3600.0	3225	1689.1	3225	3600	4048	3600	3309	13
17	-	3600.1	5746	1467.1	5750	3600	7835	3600	6426	14
18	4569	3600.0	4460	18000.2	4662	3600	6404	3600	4962	82
19	-	3600.0	3204	18005.3	3224	3600	5531	3600	3293	325
20	-	3600.1	5078	18002.7	4913	3600	9750	3600	4952	1920
21	-	3600.2	-	-	23,191	3600	36,688	3600	21,402	1440
22	-	3600.4	-	-	32,126	3600	516,686	3600	50,506	1560
23	-	3601.3	-	-	3794*	3600	54,384	3600	19,704	15,840
24	-	3631.3	-	-	2,281,440	3600	156,858	3600	58,480	31,320

Table 6: Column generation results. (The parameters in the notes are command-line arguments for our solver to facilitate easy reproduction.) For Instance 20, running the simplex solver (Clp) is feasible and results in better solutions with longer run times. Local search means solving subproblems of the IP described in section 3 (one week at a time, small group of employees at a time) (see section 5.1). This can marginally improve the column generation solution, suggesting that perhaps other heuristics for pricing would be useful in addition to the shortest path computations. The parameter objective_change_before_fixing was set to different values for different instances. This parameter is a speed vs. quality tradeoff and gives us reasonable runtimes even for the very largest instance. See also table 7.

Instance	Solution	Total runtime	Notes
20	5563	5min	Solution #1. --use_first_order_solver --fix_threshold=0.85
20	5408	9min	Solution #2. ”
20	5286	17min	Solution #1. --fix_threshold=0.85
20	5083	29min	Solution #4. ”
20	4952	29+3min	With local search.
20	4918	29+15min	”
21	21402	24min	Solution #1. --use_first_order_solver
21	21274	51min	Solution #4. ”
21	21159	23h	With local search.
22	50506	26min	#1. --objective_change_before_fixing=10 --use_first_order_solver
22	41169	110min	#7. --objective_change_before_fixing=10 --use_first_order_solver
22	33155	24h	--objective_change_before_fixing=10 --use_first_order_solver
23	24151	1.6h	#1. --objective_change_before_fixing=30 --use_first_order_solver
23	19704	4.4h	#4. ”
23	17428	43h	With local search.
24	58480	8.7h	--objective_change_before_fixing=100 --use_first_order_solver
24	48777	72h	

Table 7: Parameter sentitivity analysis on instance 18. Both the relative change in objective function value to determine convergence and the threshold used to fix variables determine a speed-quality tradeoff. See section 4.4 for a description of the parameters.

RMP objective change	Fixation threshold	First four solutions	Total time (s)
100	0.75	6830, 6108, 5684, 5577	7
50	0.75	6179, 5971, 5675, 5375	9
10	0.75	5353, 5425, 5524, 5733	13
5	0.75	5260, 5464, 5366, 5655	19
1	0.75	5960, 5633, 5337, 5229	23
0.5	0.75	5426, 5045, 5131, 5020	26
1	0.6	8086, 9187, 9072, 8570	17
1	0.75	5960, 5633, 5337, 5229	23
1	0.9	4784, 5071, 4752, 4750	45