Annals of Operations Research (2021) 302:341–362
https://doi.org/10.1007/s10479-019-03487-6

S.I.: PATAT 2018

Solving the shift and break design problem using integer linear programming

Arjan Akkermans[1] · **Gerhard Post**[2,3] · **Marc Uetz**[2]

Published online: 4 December 2019

Abstract
In this paper we propose a two-phase approach to solve the shift and break design problem using integer linear programming. In the first phase we create the shifts, while heuristically taking the breaks into account. In the second phase we assign breaks to each occurrence of any shift, one by one, repeating this until no improvement is found. On a set of benchmark instances, composed by both randomly-generated and real-life ones, this approach obtains better results than the current best known method for shift and break design problem.

Keywords Shift design · Break scheduling · Timetabling

1 Introduction

Personnel scheduling was first introduced by Edie (1954) and formulated as a set covering problem by Dantzig (1954) in the 1950s. After its introduction it has received a great deal of attention in the literature and has been applied to numerous areas such as airlines, health care systems, police, call centers and retail stores (Ernst et al. 2004). The interest can be explained by labor cost being a major direct cost component for companies.

In this paper we consider the shift and break design problem which in this specific form was introduced by Di Gaspero et al. (2010). In this problem a set of *template shifts* has to be selected from a set of possible *shift types*. A template shift is characterized by a fixed starting time and a fixed duration, such as 09:00 and 8 h, while a shift type only prescribes the intervals for possible starting times and durations. A selected template shift can then be used on different days, leading to *shifts*. A shift is thus specified by a day, starting time, and a duration. A shift usually requires breaks to be admissible; when an admissible set of breaks is assigned to a shift, we will call a *duty*. The breaks in a duty are restricted by a large number of conditions which we will discuss in more detail in the next section. On a given day, one or

✉ Gerhard Post
g.f.post@utwente.nl

1 Ab Ovo, Barbizonlaan 87, 2908 ME Capelle aan den IJssel, The Netherlands

2 Department of Applied Mathematics, University of Twente, P.O. Box 217, Enschede, The Netherlands

3 PCA Mobile, Klipperweg 19, Raalte, The Netherlands

more duties, usually with different break allocations, can be created for a shift. A duty can then be assigned to a person, and the work periods in the duty contribute to fulfill the given *staffing requirements*. During a break, duties are not counted as contributing to the fulfillment of the staffing requirements, and in addition, also not during the time period immediately after a break. It is assumed that under- and overstaffing is possible, yet undesirable.

Next to avoiding under- and overstaffing, an important goal in the shift design problem is to select a small number of template shifts. This can be important, because schedules with a small number of different template shifts are easier from a managerial perspective, and better for personnel acceptance. Eventually, understaffing, overstaffing and the number of template shifts will translate into penalty costs.

The main contribution of this paper is to show that the shift and break design problem can be solved by contemporary standard ILP solvers when decomposed into two, combinatorially defined aggregation levels. The resulting algorithm seems to outperform the previously best performing method suggested by Di Gaspero et al. (2010) that combined local search with constraint programming techniques. The main idea of this paper is to try to exploit the power of contemporary ILP solvers. Our paper shows that this approach is indeed feasible, given the "right" decomposition of the problem. Having said that, it should be noted that obtaining a (proven) optimal solution for a shift and break design problem of moderate size, is still out of reach for general purpose exact solvers.

The structure of this article is as follows. Section 2 gives the problem description in more detail. Section 3 gives an overview of related literature, Sect. 4 describes our approach and Sect. 5 the computational results. Finally, Sect. 6 presents the conclusions.

2 Problem description

2.1 Shifts and breaks

We use the same problem formulation as Di Gaspero et al. (2010). The planning horizon consists of a set of consecutive days such as a week. The problem is *cyclic*, meaning that a night shift on the last day will continue on the first time periods of the planning horizon. The planning horizon is divided into time periods of a constant duration, such as 5 min. For each time period t the staffing requirement R_t is given. Per time period we have to design duties that fulfill the staffing requirement. To explain duties, we need the notions of shift type, template shift, and shift.

- **Shift type** A *shift type* specifies an earliest starting time, a latest starting time, a minimum duration and a maximum duration. Figure 1 gives an example, where the shift types are Morning, Day, Evening and Night shifts.
- **Template shift** A *template shift* specifies a starting time and a duration. Each template shift s must derive from a shift type t, meaning that the starting time of s should lie at or between the earliest and latest starting times of t, and the duration of s should be at or between the minimum and maximum durations of t.
- **Shift** A template shift combined with a day in the planning horizon constitutes a *shift*. Each template shift has as many shifts as days in the planning horizon.
- **Duty** A shift itself is not yet complete: it usually needs breaks. A shift with a valid break allocation is called a *duty*. A duty can be assigned to an employee. Note that several employees can work the same shift, but with different break allocations, i.e. they work

in different duties. The freedom in choosing the positions of the breaks can be used to follow the staffing requirements more closely.

The break allocation in a duty splits its periods into a set of alternating *breaks* and *working periods*. A duty that is on break at a certain time period, does not contribute to the staffing requirement of that period. The same holds for the period following a break: the reasoning is that the employee needs some start-up time, which makes the employee not productive in this period. The periods of the duty that contribute to the staffing requirements are called the *active periods*.

Various conditions are set on the break allocation in a shift. These conditions represent legal regulations as well as physical requirements that prevent fatigue. Let us discuss these constraints. Firstly, the total break time of a duty is given, as a function of the duration of the shift. The *number of* breaks in duty is not fixed, as the duration per break can vary: all breaks and working periods have a minimum and maximum duration. For example, if a break follows a *long* working period, which is a working period longer than a certain threshold, then the break has a longer minimum duration. Also, the start of a break cannot be too close to the beginning and end of a shift. Finally, from a certain shift duration, the duty requires a lunch break of a fixed duration which has to be scheduled around the middle of the shift. For full details we refer to "Appendix A.3".

A solution is represented by a set S of selected template shifts, the number of shifts on each day of the planning horizon for each of the selected template shifts, and a break allocation to each of the selected shifts, leading to the duties.

2.2 Example

Figure 1 and Table 1 show, respectively, an example of staffing requirements over a two-day period, and a set of shift types. Each template shift should belong to one of the shift types, which in this case means that the starting time of a template shift should lie between 05:00 and 08:00, or 09:00 and 11:00, or 13:00 and 15:00, or 21:00 and 23:00, and a template shift must have a duration between 7 and 9 h.

A solution could consist of a template shift m from 08:00 to 16:00, a template shift e from 13:00 to 20:00, and a template shift n from 21:00 to 05:00. Choosing the number

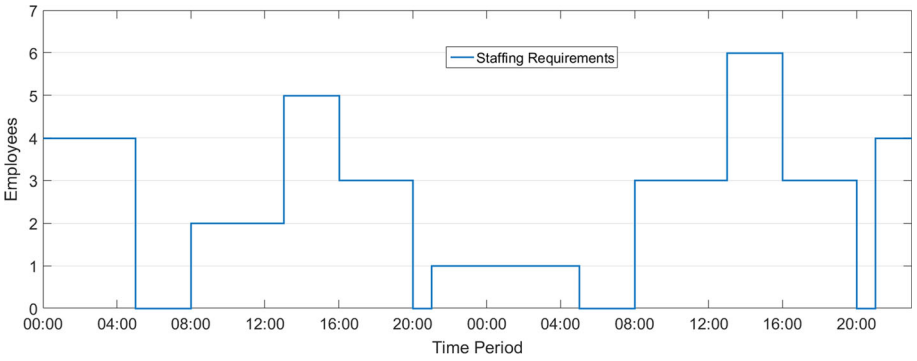

Fig. 1 Example staffing requirements

Table 1 Example shift types

Abbr.	Name	Earliest start	Latest start	Minimum duration	Maximum duration
M	Morning shift	05:00	08:00	7:00	9:00
D	Day shift	09:00	11:00	7:00	9:00
E	Evening shift	13:00	15:00	7:00	9:00
N	Night shift	21:00	23:00	7:00	9:00

Table 2 Solution to the example

Name	Type	Start	End	Day 1	Day 2
m	Morning shift	08:00	16:00	1	2
e	Evening shift	13:00	20:00	3	3
n	Night shift	21:00	05:00	1	4

of shifts starting on day 1 and day 2 as in Table 2, we fulfill exactly the staffing requirements.

If breaks have to be assigned to the shifts, there will be understaffing during all breaks, and the periods immediately after the breaks.

2.3 Objective

The objective function is the weighted sum of overstaffing, understaffing, and the number of selected template shifts. The first two factors can easily be defined by using the active time periods of a duty. Let a_t to represent number of duties that are active at time period t. The overstaffing, O, represents the total excess of active duties and the understaffing, U, the total shortage. They are defined as follows

$$O = \sum_{t \in T} \max\{a_t - R_t, 0\}, \quad U = \sum_{t \in T} \max\{R_t - a_t, 0\}$$

Using the weights W_1, W_2 and W_3 to penalize the different solution criteria, and using $|S|$ to represent the total number of selected template shifts, the objective is formulated in the following way.

$$\text{minimize } W_1 \cdot O + W_2 \cdot U + W_3 \cdot |S|. \tag{1}$$

3 Previous and related work

After the personnel scheduling problem was introduced by Dantzig (1954), many different formulations of personnel scheduling problems have been considered. A recent overview of personnel scheduling is given by Van den Bergh et al. (2013). The authors note that a wide range of solution methods is used to find solutions to personnel scheduling problems. The two main approaches used are mathematical programming methods such as linear programming, goal programming, integer programming and column generation, and heuristic methods such as simulated annealing, tabu search and genetic algorithms.

Efficient methods for solving personnel scheduling problems without breaks are available for some specific formulations. The constraint matrix in the integer linear programming formulation of Dantzig (1954), which describes a set covering problem, is a consecutive ones matrix if the problem does not contains cyclicity nor breaks, i.e. all duties are active on a consecutive number of time slots. Matrices with the consecutive ones property are totally unimodular, as first shown by Veinott and Wagner (1962), and hence integer linear programs containing such a constraint matrix can be solved efficiently by solving the linear programming relaxation. Bartholdi et al. (1980) show that personnel scheduling problems with a row circular matrix can be solved efficiently by considering at most a bounded number of flow problems. These row circular matrices follow if all duties are of the same duration and contain no breaks. Furthermore Bartholdi et al. (1980) show that the formulation of Dantzig (1954) can be slightly changed to allow for problems in which, next to the linear cost for using a duty, a linear penalty cost for overstaffing and understaffing can be considered, too.

Even when ignoring the problem of minimizing the number of template shifts, the constraint matrix for the shift design problem has neither the consecutive ones property, nor the circular row property. The constraint matrix is column circular. Cyclic scheduling problems involving duties without breaks will always have the circular ones property in the columns. Hochbaum and Levin (2006) discuss the hardness of problems with a column circular matrix, and show the problem to be equivalent to the exact matching problem which is in the complexity class NRC (Mulmuley et al. 1987). It is currently unknown whether the problem is in P.

Aykin (1996) studied a problem for a continuous 24 h workday (cyclical) and giving each duty one half hour lunch break and two 15 min breaks. For this problem an integer linear program formulation is described which differs from the original set cover formulation proposed by Dantzig (1954) by requiring three breaks to be scheduled. Rekik et al. (2010) considered an extension of the model and tested instances for which exactly three breaks were required for each duty, having a combined duration of 2 h. Furthermore, the second break was required to be the longest break and the continuous periods in-between two breaks (working period) were restricted to have a duration between 1 and 3 h. For this problem Rekik et al. (2010) propose two integer linear program formulations and compare their results with modified versions of integer linear programming formulations given by Bechtold and Jacobs (1990) and Aykin (1996). Rekik et al. (2010) show that their model is slightly slower computationally than the formulations in Bechtold and Jacobs (1990) and Aykin (1996), which is explained by the added flexibility.

The shift and break design problem of the form discussed here was introduced in Di Gaspero et al. (2010). The authors state that to the best of their knowledge the very problem as described in their paper has not been addressed before in the literature. The authors propose an approach combining local search (LS) and constraint programming (CP) (Apt 2003). Their approach starts off with finding an initial randomly generated solution for the assignment of template shifts. Random LS is employed on a part of the solution, namely a solution which specifies the template shifts, the number of shifts on each day, and the number of breaks for each shift. A CP model is then used to find a feasible break allocation. Since the authors are the first to tackle this specific shift and break design problem, there are no results to compare to. The authors use a set of randomly generated instances as well as a set of real-life instances which are publicly available (Database and Artificial Intelligence Group, Vienna University of Technology (2017)).

Obviously, the shift and break design problem is the combination of two other problems: the shift design problem (Musliu et al. 2004) and the break scheduling problem (Beer et al.

2010). Local search techniques have been applied to both of these problems, which are briefly discussed next.

Shift design problems Musliu et al. (2004) use tabu search (Glover and Laguna 1999) for the shift design problem. The authors state that they rely on local search techniques because the shift design problem is proven to be NP hard (Kortsarz and Slany 2001). Di Gaspero et al. (2007) use a hybrid heuristic, which consists of a greedy heuristic followed by a local search algorithm, which outperforms the previous results. The greedy heuristic is based on the relation of the shift design problem to a flow problem as described in Kortsarz and Slany (2001). Answer Set Programming (ASP) (Brewka et al. 2011) is an exact solution technique and has been applied to the shift design problem by Brewka et al. (2011). ASP is able to find most best known solutions within 60 min on instances of shift design where those solutions have no overstaffing and understaffing. While the execution times are often not competitive with results from Di Gaspero et al. (2007), there are instances on which ASP works very well. Solutions found for the instances in which a solution without overstaffing and understaffing might not exist are not competitive with results from the literature. A variation on the shift design problem is studied by Bonutti et al. (2016). The authors extend the shift design problem to have multiple types of skills for employees. In their formulation the staffing requirements state the required number of employees at each time period for each skill. This variation also allows for the possibility of planning one break around the middle of a shift. Simulated annealing (Kirkpatrick et al. 1983) is used to find a solution.

Break scheduling Break scheduling was first introduced by Beer et al. (2010). In their formulation different constraints are given with different weights to each type of violation and there are no hard constraints. Their formulation uses either a random assignment of breaks as an initial solution or a solution following from a simple temporal problem (Dechter et al. 1991). After the initial solution has been constructed the authors use either tabu search, simulated-annealing, or a minimum conflicts-based heuristic (Minton et al. 1992). A memetic algorithm for break scheduling was proposed by Musliu et al. (2009). Memetic algorithms combine population based methods with local search techniques and were first mentioned in Moscato (1989). The authors compare their results with the minimum conflicts-based heuristic as proposed by Beer et al. (2010). The algorithms both score best on half of the test instances and hence conclusions regarding the better algorithm are indecisive. An improved memetic algorithm is given in Widl and Musliu (2010). This algorithm obtains best results on all the instances considered by the previous authors. In this approach a part of the constraints in the break scheduling problem are considered hard and, ignoring the other constraints which regard the staffing requirements, a simple temporal problem (Dechter et al. 1991) can be formulated. Solutions from this are taken as initial solutions for the memetic algorithm. In this approach memes are defined by a set of time slots and each duty is assigned to exactly one meme in which most of the timeslots of the duty are. The reason for this approach as opposed to the approach used in Musliu et al. (2009) is that duties have a strong interference in satisfying the staffing requirements, and therefore it is difficult to find effective crossover operations when a meme represents a duty. This memetic algorithm is also used in Widl and Musliu (2014). In this paper it is also proven that break scheduling is generally NP-complete, even under the condition that all feasible break patterns of each duty are given explicitly in the input.

4 The two-phase ILP approach

Solving the shift and break design problem in one stroke is computationally hard, especially because of the importance to minimize the number of active template shifts. For this reason we propose a solution method that is composed of two phases: the first phase determines which template shifts and how many shifts will be used, while in the second phase the breaks are assigned to these shifts.

More concretely, our solution approach consists of a two-phase approach where we use integer linear programming (ILP) to find a solution in each phase. In the first phase the template shifts are designed and the number of shifts for each day are determined. The problem that we solve in the first phase is a variation of shift design in which we have to select a set of template shifts, while we heuristically account for the break time which has to be allocated in the second phase. This second phase will be treated as a series of instances of the break scheduling problem.

4.1 The first phase method

First we give an integer linear program (ILP) formulation of the shift design problem. A compact overview of this ILP is given in A.2. The first phase will use a modified version of this ILP which we will discuss after having introduced the shift design ILP.

4.1.1 The ILP for the shift design problem

The shift design problem is made up of time periods, days and shifts. We will use the following variables to refer to a single element of each set.

- t refers to a time period.
- d refers to a day.
- s refers to a template shift.

The time periods and the days are part of the input, but the template shifts are not explicitly given. However, looking at the shift types, all template shifts belonging to a certain type can be found by using pairs of possible starting times and possible durations of that shift type. For each of these template shifts we can make the decision of using it or not. In the case that a template shift is used, we call it *active* and we can assign a number of shifts to it on each day. Hereto we use the following variables.

$$a_s = \begin{cases} 1 & \text{if template shift } s \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$w_{d,s} = \text{ the number of shifts of template shift } s \text{ starting on day } d$$

If a template shift is inactive the number of shifts on each day are forced to be 0. To model this constraint we use the parameter $M = max_t\{R_t\}$. Recall that R_t denotes the staffing requirement at time period t. M is an upper bound on the number of shifts which are used in an optimal solution for the shift design problem, since using more duties will lead to unnecessary overstaffing. The following Big-M constraints force the number of shifts to be 0 in case that the corresponding template shift is not active.

$$w_{d,s} \leq M \cdot a_s \quad \forall d, s \tag{2}$$

Using the number of shifts for each template shift we can calculate the overstaffing and the understaffing. For this we need a parameter denoting on which time periods the shifts of a template shift starting on a specific day are active. We introduce the following.

$$o_t = \text{ the amount of overstaffing at time period } t, \quad o_t \geq 0$$
$$u_t = \text{ the amount of understaffing at time period } t, \quad u_t \geq 0$$
$$A_{s,d,t} = \begin{cases} 1 \text{ if the shifts of template shift } s \text{ starting on day } d \text{ are active at time period } t \\ 0 \text{ otherwise} \end{cases}$$

The following constraints specify, respectively, lower bounds for the understaffing and the overstaffing variables.

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \qquad \forall t \tag{3}$$

$$o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{4}$$

The objective function is the weighted sum of overstaffing, understaffing and the number of template shifts.

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t + W_3 \sum_s a_s \tag{5}$$

Typically, the weight W_2 for understaffing is much higher than weight W_1 for overstaffing; in practice, a quotient 10 or higher is not unusual. In the instances we study in Sect. 5, $W_2 = 10$ and $W_1 = 2$ with a rather low quotient of 5.

In Sect. 5 the weight for the number of used template shifts $W_3 = 60$. The importance of W_3 depends on the chosen time granularity, as penalties for understaffing and overstaffing are counted per time period per duty. Hence in the instances we discuss, with a time granularity of 5 min, the costs for having an additional template shift is equal to the cost of having an understaffing of 1 for 30 min.

4.1.2 Virtual shifts

In order to design template shifts that follow the requirements in the shift and break design problem more closely, we need to account for the break time which has to be allocated to each duty in the second phase. In order to do this we construct a *virtual* shift for each possible template shift. These are used to represent a shift while heuristically accounting for the break time which has to be scheduled. Instead of a binary indicator for denoting whether a duty will be active at a time period, the virtual shifts can be thought of as representing the probability of a duty being active at a time period.

In the shift design ILP we used the parameters $A_{s,d,t}$, which were binary and indicated whether shifts of template shift s starting on day d were active during time period t. For the virtual shifts we will use the values $A^*_{s,d,t}$ which can be any value from 0 to 1. A duty is not active on its breaks and on the first time period following a break. The total break time, in time periods, required for a shift belonging to template shift s is given as part of the input by $f(s)$. Before assigning the breaks we do not know the number of breaks that a duty will have and hence we cannot determine the number of inactive time periods of that duty exactly. However, we can calculate the maximum and minimum number of breaks of a duty by using the various restrictions on the break allocations. We will use $IP(s)$ to indicate the number

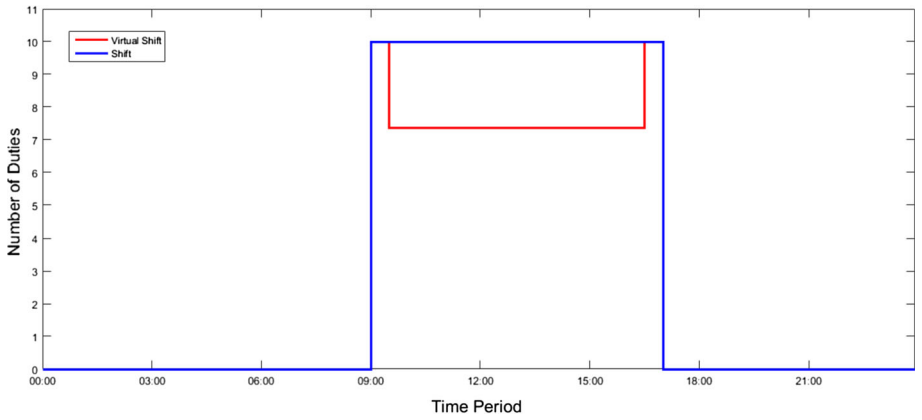

Fig. 2 Shift and virtual shift

of inactive time periods for a duty belonging to template shift s, under the assumption of having as many breaks as possible. Hereto we use $maxB(s)$ to denote the maximum number of breaks for template shift s, which leads to

$$IP(s) = f(s) + maxB(s) \tag{6}$$

We use the maximum number of breaks to account for the worst case. Therefore this approach is conservative, and may lead to scheduling too many shifts. This is justified in staff scheduling problems that consider understaffing as (far) less desirable than overstaffing.

Recall that breaks cannot be scheduled near the beginning and end of a shift. Denote by α and β the number of time periods where the shift is guaranteed to be active at the start and at the end of the shift, respectively. Outside these time intervals a duty might be inactive depending on the break schedule. We simply "divide" the rest of the breaks evenly over the remaining time periods of the duty. Using $s.length$ to denote the number of time periods in template shift s, we can express R_s^*, the ratio of inactive periods to active periods of a shift of template shift s, excluding the first α and last β time periods, by

$$R_s^* = \frac{IP(s)}{s.length - \alpha - \beta} \tag{7}$$

We define the parameters $A^*_{s,d,t}$ as follows

$$A^*_{s,d,t} = \begin{cases} 0 & \text{if } t \text{ is not part of a shift of template shift } s \text{ starting on day } d \\ 1 & \text{if } t \text{ is in the first } \alpha \text{ time periods of a shift of template shift } s \text{ starting on day } d \\ 1 & \text{if } t \text{ is in the last } \beta \text{ time periods of a shift of template shift } s \text{ starting on day } d \\ 1 - R_s^* & \text{otherwise} \end{cases} \tag{8}$$

Figure 2 shows the effect of working with a virtual shift (with 10 duties).

4.1.3 The ILP for the first phase

The shift design ILP can be adapted in various ways to incorporate the virtual shifts. It is possible to simply change the parameters $A_{s,d,t}$ to $A^*_{s,d,t}$. Preliminary tests showed that this approach gives decent results, however a drawback is that the actual flexibility of being able to choose the break allocation is not taken into account.

With an eye on the fact that understaffing is generally less desirable than overstaffing, we work with virtual shifts in a slightly "asymmetric" way, by adapting the constraints that express under- and overstaffing.

For understaffing we simply use the constraint (3):

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \quad \forall t \tag{9}$$

In other words, we penalize understaffing at time t in the case that there are not enough shifts even if all duties are active at time t. Clearly in such a case understaffing would be inevitable.

Overstaffing at a time period, however, will be penalized only if there are too many duties when using the virtual values $A^*_{s,d,t}$ as defined by the virtual shifts.

$$o_t \geq \sum_{s,d} \left(A^*_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{10}$$

This updated ILP accepts without penalty all overstaffing and understaffing, as long as the number of present workers is between the number specified by the shifts and the virtual shifts.

Using only these two constraints, it is possible that overall too few duties are scheduled since the constraints allow a duty to be always active. To tackle this issue, we enforce that over a certain number C of consecutive time periods the sum of the staffing requirements are fulfilled using the virtual shifts. Since the problem is cyclic, the index of R is $(t+i) \bmod n$.

$$\sum_{i=1}^{C} \left(\sum_{s,d} A^*_{s,d,t+i} \cdot w_{d,s} \right) \geq \sum_{i=1}^{C} R_{(t+i) \bmod n} \quad \forall t \tag{11}$$

Our experiments suggested to work with the value $C = 25$, expressing the idea that understaffing at a certain time period can be resolved by borrowing overstaffing from 12 "adjacent" time periods, so 1 h in both directions. We believe that the number C should be chosen specific to the instances: it reflects the flexibility of moving breaks around. In our instances the working periods have a duration between 30 and 100 min, making a 60 min average for avoiding understaffing plausible.

4.2 Second phase method

Here we give a description of the algorithm used to allocate the breaks. This algorithm iteratively uses a break scheduling ILP for a *single duty* at a time, and takes all other duties and their current break allocations as fixed. The ILP for the break scheduling of a single duty is given in "Appendix A.3". For a more extensive description of this ILP we refer to the MSc thesis (Akkermans 2017).

The algorithm for the second phase can be seen as an iterative greedy algorithm. The shifts on each day as determined by the first phase are used, where at the start of phase two, all duties are active from the start to the end of the duty (i.e., no breaks are scheduled yet). Then the duties are considered in a fixed but random order, and an optimal break allocation is calculated for a single duty by using the break scheduling ILP. This alters the number of active duties at some of the time periods, and hence the objective function. This iterative greedy procedure continues as long as it is possible to improve the solution by changing the break allocation of a single duty, and terminates after a full round without any improvement.

5 Computational results

The website (Database and Artificial Intelligence Group, Vienna University of Technology (2017)) contains a number of instances that we used to test our method. The instances are divided in three sets. The first two sets both consist of 30 randomly generated instances and are referred to as "First Set" and "Second Set" in Tables 3 and 4. These instances were created by "reverse engineering" as follows: The staffing requirements are generated from a solution with a randomly generated set of active template shifts and duties. Hence, there is a "best known" solution which provides an exact coverage of the requirements.

The third set consists of 5 instances and based on a real world example. For these instances a good solution in which there is no overstaffing and understaffing is very unlikely to exist. The real life instances are smaller than the randomly generated instances with respect to the total staffing requirements. For the real life case the average value for the sum of the staffing requirements over all time periods equals 10,193, for the randomly generated instances this number is 16,535. However, the given shift types for the real life instances allow more template shifts. For these instances the total number of possible template shifts is 8645, while the randomly generated instances allow for 2800 possible template shifts. Due to the higher number of possible template shifts for the real life instances our method did not obtain satisfactory results when solving an ILP containing all possible template shifts for these instances. To overcome this challenge we only considered template shifts at a time granularity of 15 min for the real life instances. This reduces the number of possible template shifts from 8645 to 1075 template shifts, but might cut out the optimal solution. In all instances, the weights W_1, W_2 and W_3, see Eq. (1) for overstaffing, understaffing, and number of used template shifts are set to

$$W_1 = 2, \qquad W_2 = 10, \qquad W_3 = 60.$$

Our algorithm for shift and break design was written in C++ using Microsoft Visual Studio 2013 (Microsoft 1980). The integer linear programs were solved using IBM ILOG CPLEX Optimization Studio 12.7.1 (IBM 2017). The algorithm was performed on a PC with an Intel i7-4702MQ quad core processor with 2.2 GHz and 8 GB RAM memory, allowing CPLEX to use all cores.

In Di Gaspero et al. (2010), the authors use a time limit of 60 min per instance to find a solution. Given the type of problem (generating a week's schedule) this seems a reasonable time, as viewed from the functional perspective. In technological sense, using 60 min now is incomparable to using 60 min 10 years ago. However, our research is not to give a technological comparison, but a functional one: what can be achieved nowadays in 1 h, using a normal laptop?

We split the 60 min calculation time in a maximum of 30 min for the first phase and the same maximum for the second phase. In the first phase these 30 min are always used completely, i.e. after 30 min CPLEX is not yet sure that the optimal solution has been obtained. The average optimality gap, however is small: in the first phase for the randomly generated instances is 0.50 (with standard deviation 0.10), for the real life instances this number is 0.86 (with standard deviation 0.02). A running time of lower than 60 min thus means that the second phase converges before the allocated 30 min are used. This is the case in 40 of the randomly generated instances and for 3 of the real life instances. Our solutions are available at the website of Database and Artificial Intelligence Group, Vienna University of Technology ((2017)).

The results for the randomly generated instances are shown in the Tables 3 and 4. The explanation of the columns is as follows.

– The colums "Inst." refers to the instance number.

Table 3 Results first set

Inst.	over			under			shifts			Objective				Time (mins)
	2P-ILP	Di Gasp.	Best	2P-ILP	Di Gasp.	Best	2P-ILP	Di Gasp.	Best	LB-I	2P-ILP.	Di Gasp.	Best	2P-ILP
1-1	1237	3355	0	3	215	0	13	28	8	3176	3284	10540	480	42
1-2	1740	4557	0	2	405	0	31	29	10	5226	5360	14904	600	40
1-3	1129	4800	0	7	369	0	13	34	10	2960	3108	15330	600	48
1-4	2215	5901	0	16	421	0	47	44	16	7066	7410	18652	960	56
1-5	1537	3713	0	10	285	0	22	23	8	4260	4494	11656	480	42
1-6	1005	2638	0	4	246	0	13	17	7	2726	2830	8756	420	37
1-7	1716	3046	0	12	227	0	22	28	9	4662	4872	10042	540	48
1-8	1715	4465	0	4	330	0	33	33	10	5276	5450	14210	600	53
1-9	1470	3930	0	6	276	0	24	25	10	4236	4440	12120	600	51
1-10	1895	4997	0	8	389	0	38	32	11	5902	6150	15804	660	**60**
1-11	948	n.a.[a]	0	0	n.a.[a]	0	3	n.a.[a]	2	2060	2076	n.a.[a]	120	39
1-12	1132	2850	0	8	164	0	10	17	6	2776	2944	8360	360	51
1-13	1494	3888	0	7	303	0	14	25	7	3738	3898	12306	420	**60**
1-14	2035	5763	0	5	380	0	44	47	13	6464	6760	18146	780	**60**
1-15	1059	1282	0	1	179	0	7	7	3	2520	2548	4774	180	34
1-16	2023	4610	0	12	426	0	39	39	15	6220	6506	15820	900	48
1-17	2079	5746	0	7	373	0	52	53	18	7006	7348	18402	1080	59
1-18	1824	4724	0	10	530	0	28	32	12	5130	5428	16668	720	58
1-19	1246	3961	0	12	368	0	36	33	12	4408	4772	13582	720	**60**
1-20	1734	5247	0	11	438	0	21	32	9	4582	4838	16794	540	**60**
1-21	1523	3164	0	3	242	0	22	24	8	4286	4396	10188	480	43
1-22	782	3288	0	2	186	0	6	23	5	1872	1944	9816	300	37
1-23	1851	4468	0	14	265	0	33	34	10	5632	5822	13626	600	45
1-24	1483	3585	0	11	330	0	23	21	8	4208	4456	11730	480	42
1-25	1757	5608	0	10	452	0	44	45	16	5896	6254	18436	960	54
1-26	1622	3723	0	8	698	0	39	31	11	5440	5644	16286	660	48
1-27	1746	4602	0	21	718	0	11	35	8	3960	4362	18484	480	**60**
1-28	1579	3086	0	4	258	0	20	20	9	4256	4398	9952	540	40
1-29	1813	3973	0	12	336	0	36	39	12	5720	5906	13646	720	51
1-30	1444	2512	0	0	250	0	8	18	5	3352	3368	8604	300	39
Average	1561	4051	0	8	347	0	25	30	10	4501	4702	13367	576	49

[a]The authors did not report a solution

Table 4 Results second set

Inst.	over			under			shifts			Objective				Time (mins)
	2P-ILP	Di Gasp.	Best	2P-ILP	Di Gasp.	Best	2P-ILP	Di Gasp.	Best	LB-I	2P-ILP.	Di Gasp.	Best	2P-ILP
2-1	1485	4376	0	13	375	0	38	25	12	5064	5380	14002	720	52
2-2	1792	3968	0	3	289	0	33	34	12	5420	5594	12866	720	**60**
2-3	1982	4319	0	45	288	0	38	39	12	5944	6694	13858	720	**60**
2-4	1501	3935	0	6	311	0	37	30	12	5088	5282	12780	720	58
2-5	1610	4161	0	8	260	0	35	34	12	5144	5400	12962	720	**60**
2-6	1678	5197	0	58	330	0	46	42	12	5822	6696	16214	720	**60**
2-7	1603	5187	0	19	427	0	57	40	12	6362	6816	17044	720	49
2-8	1848	4357	0	2	323	0	34	29	12	5606	5756	13684	720	44
2-9	1720	4461	0	15	391	0	42	35	12	5844	6110	14932	720	47
2-10	1868	5566	0	7	432	0	40	42	12	5878	6206	17972	720	58
2-11	1844	n.a.[a]	0	8	n.a.[a]	0	48	n.a.[a]	16	6312	6648	n.a.[a]	960	48
2-12	1786	4929	0	3	365	0	58	42	16	6816	7082	16028	960	52
2-13	1704	5148	0	2	421	0	58	49	16	6638	6908	17446	960	57
2-14	1976	5678	0	8	410	0	42	53	16	6268	6552	18636	960	**60**
2-15	2104	5696	0	24	518	0	47	41	16	6794	7268	19032	960	**60**
2-16	1785	6255	0	20	350	0	60	49	16	6810	7370	18950	960	**60**
2-17	1787	4537	0	7	452	0	55	36	16	6700	6944	15754	960	53
2-18	1871	5413	0	16	557	0	52	37	16	6516	7022	18616	960	**60**
2-19	1935	5418	0	5	604	0	51	43	16	6660	6980	19456	960	**60**
2-20	1988	5769	0	16	475	0	52	40	16	6782	7256	18688	960	54
2-21	2147	6150	0	7	371	0	74	48	20	8354	8804	18890	1200	**60**
2-22	1869	6312	0	5	430	0	59	48	20	6928	7328	19804	1200	58
2-23	2000	5288	0	25	402	0	69	44	20	7818	8390	17236	1200	58
2-24	1774	5634	0	8	427	0	60	44	20	6810	7228	18178	1200	59
2-25	1854	5754	0	12	487	0	60	47	20	6974	7428	19198	1200	**60**
2-26	2168	5836	0	17	517	0	59	47	20	7514	8046	19662	1200	**60**
2-27	2324	5835	0	34	529	0	66	54	20	8076	8948	20200	1200	**60**
2-28	1953	5157	0	1	340	0	65	45	20	7584	7816	16414	1200	**60**
2-29	2105	5407	0	54	506	0	63	45	20	7558	8530	18574	1200	**60**
2-30	1996	7041	0	6	678	0	66	60	20	7658	8012	24462	1200	58
Average	1869	5268	0	15	423	0	52		16	6591	7016	17294	960	57

[a]The authors did not report a solution

Table 5 Results real life instances

Inst.	Over		Under		Shifts		Objective			Time (min)
	2P-ILP	Di Gasp.	2P-ILP	Di Gasp.	2P-ILP	Di Gasp.	LB-I	2P-ILP.	Di Gasp.	2P-ILP
2fc04a	1224	2636	5	173	30	23	4050	4298	8382	56
3fc04a	1227	2732	1	130	36	21	4398	4624	8024	57
4fc04a	1081	2710	4	94	32	21	3896	4122	7620	56
50fc04	1130	2636	4	180	29	29	3744	4040	8812	**60**
51fc04	1343	2890	0	209	32	23	4360	4606	9250	**60**
Average	1201	2720	3	157	32	23	4090	4338	8418	58

- The columns below "over", "under", "shifts" and "Objective" show, respectively the overstaffing, understaffing, number of template shifts and the total objective value of the solution.
- those previous four columns are further divided according to the solution method, "2P-ILP" shows our results, "Di Gasp" shows the results of Di Gaspero et al. (2010), and "Best" show the results of the best know solution (used to generate the instances).
- The column "LB-I" under "Objective" gives a lower bound estimate for our final solution based on the solution of the first phase and is constructed as follows. After the first phase the number of used template shifts are known, as well as the number of shifts on each day. This, together with the assumption that even with the maximum number of breaks scheduled the total number of staffing requirements can still be covered (which was true for all our solutions), can be used to calculate a lower bound, since under these circumstances the best possible thing to do is to schedule as much breaks as possible without creating any additional understaffing anywhere. Thus for the lower bound estimate we only take into account *unavoidable* understaffing which is the understaffing in case that no breaks are scheduled, and the overstaffing which is *unavoidable*: the sum of the total requirements minus the sum of the total number of active duties considering the maximum number of breaks.
- The column "Time" is the computation time in minutes for our solution. This time is at most 60 min, because the first phase always uses 30 min, and the second phase gets 30 min as maximum.

The results of the real life instances are shown in Table 5, and the notations are similar. The tables show that in the randomly generated instances, the lower bound LB-I is on average 5.1% better when compared to the objective value found by our two-phase approach. For the real life instances this is 5.7%.

The low understaffing is partly a consequence of the choice made in Constraint (11) where we did not allow any understaffing over 25 time periods as measured by the virtual shifts. Although this requirement seems rather weak, it creates solutions with almost no understaffing. We experimented with soft variants of Constraint (11), which led to inconclusive results. The main reason seems to be that obtaining good lower bounds for the first phase problem is more difficult, leading to increased computation times, and worse solutions on several instances, as well as large improvements on other ones.

Solving the break scheduling problem for one shift (see "Appendix A.3") takes usually 5–10 s. On average all (more than 100) shifts are optimized 3.7 times; after a full round without improvement the optimization is stopped. The order of treatment is top-down, but the results are very stable, in the sense that the gap to the lowerbound of the first phase is stable around 5–6%. We believe that another (random) sorting of the shifts would lead to very similar results.

In Akkermans (2017) some experiments with applying the break scheduling problem for two shifts at the same time are performed. However, the computation time increases to several minutes, so that after 30 min the result is inferior to the single shift approach. In this respect it is interesting to know that a shift of 8 h has 1,878,678 possible duties (given the parameter settings in our instances). As an instance has usually well over 100 shifts, solving the break scheduling problem directly in 30 min is challenging. Even if one could, it would improve the solution by at most a few percent, given the lowerbound LB-I.

6 Conclusion

We proposed an integer linear programming approach for the shift and break design problem, which obtains better results on all instances when compared to the best results available in the literature. Note that the integer linear program of the first phase can also be used to solve the instances of shift design problem, which are all solved to optimality within the time limit of 30 min. These result are shown in "Appendix A.1".

Since we were not able to solve the ILP in the first phase to optimality we are not sure of the performance of our algorithm in case the optimal solution for this ILP would have been be found. For the randomly generated instances, for which a solution that exactly matches the staff requirements is known, the average objective value of our solutions are more than 5 times higher than the cost of the solution that exactly match the requirements. Based on the solution of the first phase, we can estimate a lower bound for the final solution, as documented in the "LB-I" columns in the Tables 3, 4 and 5. Note that the realized objective is just a few percent higher than this lower bound "LB-I". In other words, the overall results can still be improved significantly, and to do this, the first phase should generate "better" solutions.

Above we observed that the second phase works very well in the two-phase approach that we propose, with an optimality gap of 5–6%. However, applying phase two solely on instances of break scheduling problem turned out to be less effective than the memetic algorithms of Widl and Musliu (2010), when starting with the shifts that were used to construct the instances. The reason might be that there is exactly one solution without overstaffing and understaffing, which is difficult to find by optimizing shift by shift, as it needs extreme coordination to reach it.

We decomposed the shift and break design problem in two phases, a first phase where we decide on the template shifts and the number of shifts, and a second phase in which we decide where to place the breaks. The first phase is in a sense the hardest to handle, as the information is incomplete. We believe that the virtual shifts work well, because of the high flexibility of the breaks; in that case the assumption that in the middle of a shift there is a uniform distribution on the probability of a time being a break time is reasonable. A more detailed analysis could make this more precise, and might or might not improve the quality of the solutions. This is a point for further investigations.

Another approach, used often in personnel scheduling problems, is a column generation approach. Usually a column is a schedule for an employee, but in this case an approach where each separate duty, with breaks assigned, is a column might be more appropriate. Note, however, that due to minimizing the number of template shifts in use, the columns are dependent on each other. How to overcome this issue is another topic to investigate.

Appendix

A.1 Results shift design

Comparison of ILP formulation of shift design problems solved with the commercial solver Cplex (IBM 2017) and the greedy min-cost max-flow + Local Search heuristic (Gr) proposed by Di Gaspero et al. (2007). Table 6 shows the objective of the (previous) best known solution and running time of 'Gr' to reach it. For our ILP approach we show the optimal objective

Table 6 Shift design comparison set 1

Inst.	Objective		Time (s)	
	Gr	ILP	Gr	ILP
1-1	480	320	1	2
1-2	300	212	40	21
1-3	600	374	2	2
1-4	450	340	109	165
1-5	480	319	2	2
1-6	420	213	1	1
1-7	270	237	7	1
1-8	150	147	11	149
1-9	150	149	9	29
1-10	330	289	84	141
1-11	30	30	1	1
1-12	90	81	4	3
1-13	105	105	4	9
1-14	195	187	61	425
1-15	180	170	0	1
1-16	225	209	152	1552
1-17	540	394	288	283
1-18	720	447	7	6
1-19	180	177	31	54
1-20	540	353	2	2
1-21	120	119	2	7
1-22	75	75	4	4
1-23	150	150	22	100
1-24	480	343	1	6
1-25	480	352	n.a.[a]	168
1-26	600	347	9	5
1-27	480	393	2	3
1-28	270	222	4	32
1-29	360	289	10	58
1-30	75	75	2	2
Average	318	237	30	108

[a]Authors were not able to find the best known solution using the Gr algorithm

Table 7 Shift design comparison set 3

Inst.	Objective	
	Gr	ILP
3-1	2386	318
3-2	7691	845
3-3	9597	924
3-4	6681	1427
3-5	9996	551
3-6	2077	1892
3-7	6087	642
3-8	8861	725
3-9	6036	2527
3-10	3002	462
3-11	5491	1024
3-12	4171	3514
3-13	4662	3131
3-14	9661	701
3-15	11, 445	1112
3-16	10, 734	638
3-17	4729	3011
3-18	6692	893
3-19	5157	2677
3-20	9175	1845
3-21	6054	4674
3-22	12, 870	2063
3-23	8390	699
3-24	10, 418	741
3-25	13, 252	847
3-26	13, 118	1042
3-27	10, 081	1034
3-28	10, 604	887
3-29	6690	1045
3-30	13, 724	1011
Average	7984	1430

value and the running time to find and prove it. Table 7 shows the results of both approaches using a running time of 1 second.

A.2 ILP shift design

Sets

Time Periods	t
days	d
template shifts	s

Variables

a_s	Binary variable indicating whether template shift s is active
$w_{d,s}$	Integer variable indicating how many workers will be working on template shift s on day d
u_t	Will denote the understaffing at time period t
o_t	Will denote the overstaffing at time period t

Parameters

R_t	The requirement at time period t
$A_{s,d,t}$	Binary, indicating if the shift starting on day d of template shift s is active on time period t
W_1	Penalty cost for overstaffing
W_2	Penalty cost for understaffing
W_3	Penalty cost for the number of template shifts
M	Is used as a large constant, here it is the maximum demand at any time period

Constraints

$$w_{d,s} \leq M \cdot a_s \qquad \forall d, s$$

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \qquad \forall t$$

$$o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t$$

$$a_s \in \{0, 1\} \qquad \forall s$$

$$w_{d,s} \in \mathbb{N}_0 \qquad \forall d, s$$

$$o_t, u_t \geq 0 \qquad \forall t$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t + W_3 \sum_s a_s \tag{12}$$

A.3 ILP break scheduling single shift

Sets

Time Periods	$t = \{1, \ldots, Length\}, t^* = \{1, \ldots, Length + 1\}$
Breaks	$b = \{1, \ldots, M_{breaks}\}$

Parameters

D_t	The demand for staffing at time period t
M_{breaks}	The maximum number of different breaks+1
Tbt	The total amount of break time required
$Length$	Length of the duty s, in number of time periods
$Bminl$ ($Bmaxl$)	Minimum (maximum) length of a break
$Mlwp$	Minimum long working period
$Lbml$	Long break minimum length

$Wpminl$ ($Wpmaxl$)	Working period minimum (maximum) length
Ebs (Lbs)	Earliest (latest)break start; breaks can start this many time periods after (from) shift start (end)
L	Binary, indicating whether a lunch break is required
$Elbs$ ($Llbs$)	Earliest (Latest) lunch break start
Lbl	Lunch break length
W_1 (W_2)	Penalty cost for overstaffing (understaffing)
M	Used as a sufficiently large constant. Equal to the number of time periods in the duty

Variables

a_b	Binary variable indicating whether the bth break is active
a_b^l	Binary variable indicating whether break b needs to be long
l_b	Binary variable indicating whether break b is the lunch break
b_b^l	The length (in time slots) that the bth break takes
b_b^s	First time slot on which break b is active
wp_b	Indicates the length of the bth working period
$z^s_{t^*,b}$	Binary indicating if the first time period of the bth break is the t^*th time period
$z^e_{t^*,b}$	Binary indicating if the last time period of the bth break is the t^*th time period
$z_{t,b}$	Binary[1] indicating if the duty is on its bth break during time period t
$z^b_{t,b}$	Binary[1] indicating if time period t is before the start of the bth break
$z^a_{t,b}$	Binary[1] indicating if time period t is after the end of the bth break
x_t	Binary[1] indicating if the duty is working during a time period t
u_t, o_t	Understaffing, Overstaffing in time period t

Constraints

$$a_b \le a_{b-1} \qquad \forall b \in B \setminus \{1\}$$

$$a_b^l + l_b \le a_b \qquad \forall b$$

$$\sum_b b_b^l = Tbt$$

$$M \cdot (1 - a_b) + b_b^l \ge Bminl \qquad \forall b$$

$$b_b^l \le Bmaxl \cdot (a_b) + M \cdot (l_b) \qquad \forall b$$

$$a_b^l \cdot Lbml \le b_b^l \qquad \forall b$$

$$b_b^s \ge Ebs \qquad \forall b$$

$$b_b^s \le Length - Lbs + (1 - a_b) \cdot (1 + Lbs) \qquad \forall b$$

$$b_b^s \ge (Length + 1) \cdot (1 - a_b) \qquad \forall b$$

$$wp_1 = b_1^s - 1$$

$$wp_b = b_b^s - (b_{b-1}^s + b_{b-1}^l) \qquad \forall b \in B \setminus \{1\}$$

$$wp_b \le Wpmaxl \qquad \forall b$$

$$wp_1 \ge Wpminl$$

[1] These variables can be relaxed to continuous variables on [0, 1]. By the binary restriction on $z^s_{t,b}$ and $z^e_{t,b}$ the variables are forced to be either 0 or 1.

$$M \cdot (1 - a_{b-1}) + wp_b \geq Wpminl \qquad \forall b \in B \setminus \{1\}$$

$$wp_b - Mlwp + 1 \leq a_b^l \cdot M + l_b \cdot M + (1 - a_b) \cdot M \qquad \forall b$$

$$Mlwp - wp_b \leq (1 - a_b^l) \cdot M \qquad \forall b$$

$$\sum_b l_b = L$$

$$(1 - l_b) \cdot M \geq Lbl - b_b^l \qquad \forall b$$

$$(1 - l_b) \cdot M \geq b_b^l - Lbl \qquad \forall b$$

$$(1 - l_b) \cdot M \geq Elbs - b_b^s \qquad \forall b$$

$$(1 - l_b) \cdot M \geq b_b^s - Llbs \qquad \forall b$$

$$\sum_{t^*} z_{t^*,b}^s = 1 \qquad \forall b$$

$$\sum_{t^*} t^* \cdot z_{t,b}^s = b_b^s \qquad \forall b$$

$$\sum_{i=1>Length+1} z_{i,b}^s = z_{t,b}^b \qquad \forall t, b$$

$$\sum_{t^*} z_{t^*,b}^e = 1 \qquad \forall b$$

$$\sum_{t^*} t \cdot z_{t^*,b}^e = b_b^s + b_b^l - 1 + (1 - a_b) \qquad \forall b$$

$$\sum_{i=1<t-1} z_{i,b}^e = z_{t,b}^a \qquad \forall t, b$$

$$z_{t,b} + z_{t,b}^a + z_{t,b}^b = 1 \qquad \forall t, b$$

$$1 - x_t \geq \sum_b z_{t,b} \qquad \forall t \in T$$

$$1 - x_t \geq \sum_b z_{t-1,b}^e \qquad \forall t \in T \setminus \{1\}$$

$$1 - x_1 \leq \sum_b z_{1,b}$$

$$1 - x_t \leq \sum_b z_{t,b} + \sum_b z_{t-1,b}^e \qquad \forall t \in T$$

$$x_t - 1 + u_t \geq D_t \qquad \forall t$$

$$o_t - u_t - x_t + 1 = -D_t \qquad \forall t$$

$$a_b, a_b^l, l_b \in \{0, 1\} \qquad \forall b$$

$$z_{t,b}^s, z_{t,b}^e \in \{0, 1\} \qquad \forall t, b$$

$$o_t, u_t \geq \forall \qquad t$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t. \tag{13}$$

References

Akkermans, A. (2017). *A two-phase approach to the shifts and breaks design problem using integer linear programming*. Master's thesis, University of Twente. Retrieved November 30, 2019, from http://essay.utwente.nl/74147/.

Apt, K. (2003). *Principles of constraint programming*. New York, NY: Cambridge University Press.

Aykin, T. (1996). Optimal shift scheduling with multiple break windows. *Management Science*, *42*(4), 591–602.

Bartholdi, J. J., Orlin, J. B., & Ratliff, H. D. (1980). Cyclic scheduling via integer programs with circular ones. *Operations Research*, *28*(5), 1074–1085.

Bechtold, S. E., & Jacobs, L. W. (1990). Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, *36*(11), 1339–1351.

Beer, A., Gartner, J., Musliu, N., Schafhauser, W., & Slany, W. (2010). An AI-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, *25*(2), 60–73.

Bonutti, A., Ceschia, S., De Cesco, F., Musliu, N., & Schaerf, A. (2016). Modeling and solving a real-life multi-skill shift design problem. *Annals of Operations Research*, *252*, 365–382.

Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92–103.

Dantzig, G. B. (1954). A comment on Edie's "Traffic Delays at Toll Booths". *Journal of the Operations Research Society of America*, *2*(3), 339–341.

Database and Artificial Intelligence Group, Vienna University of Technology. (2017). *Shift design and break scheduling benchmarks*. Retrieved October 30, 2019, from http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*(1), 61–95.

Di Gaspero, L., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A., & Slany, W. (2007). The minimum shift design problem. *Annals of Operations Research*, *155*(1), 79–105.

Di Gaspero, L., Gärtner, J., Musliu, N., Schaerf, A., Schafhauser, W., & Slany, W. (2010). A hybrid LS-CP solver for the shifts and breaks design problem. In *7th International workshop on hybrid metaheuristics*, lecture notes in computer science (Vol. 6373, pp. 46–61). Heidelberg: Springer.

Edie, L. C. (1954). Traffic delays at toll booths. *Journal of the Operations Research Society of America*, *2*(2), 107–138.

Ernst, A., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, *153*(1), 3–27.

Glover, F., & Laguna, M. (1999). Tabu search. In D. Z. Du & P. M. Pardalos (Eds.), *Handbook of combinatorial optimization* (Vol. 1-3, pp. 2093–2229). Boston, MA: Springer.

Hochbaum, D. S., & Levin, A. (2006). Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. *Discrete Optimization*, *3*(4), 327–340.

IBM. (2017). *IBM ILOG CPLEX Optimization Studio 12.7.1*.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kortsarz, G., & Slany, W. (2001). *The minimum shift design problem and its relation to the minimum edge-cost flow problem*. Tech. Rep. DBAI-TR-2001-46, Technische Universität Wien.

Microsoft. (1980). *Microsoft Visual Studio Community 2013*, Version 12.031101.00 Update 4.

Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, *58*(1), 161–205.

Moscato, P. (1989). *On evolution, search, optimization, genetic algorithms and martial arts—towards memetic algorithms*. Technical Report, California Institute of Technology.

Mulmuley, K., Vazirani, U., & Vazirani, V. (1987). Matching is as easy as matrix inversion. *Combinatorica*, *7*(1), 105–113.

Musliu, N., Schaerf, A., & Slany, W. (2004). Local search for shift design. *European Journal of Operational Research*, *153*(1), 51–64.

Musliu, N., Schafhauser, W., & Widl, M. (2009). A memetic algorithm for a break scheduling problem. In *Proceedings of the 8th metaheuristic international conference (MIC 2009)*, Hamburg, Germany.

Rekik, M., Cordeau, J. F., & Soumis, F. (2010). Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, *13*, 49–75.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*(3), 367–385.

Veinott, A. F., & Wagner, H. M. (1962). Optimal capacity scheduling—I. *Operations Research*, *10*(4), 518–532.

Widl, M., & Musliu, N. (2010). An improved memetic algorithm for break scheduling. In M. J. Blesa, C. Blum, G. Raidl, A. Roli, & M. Sampels (Eds.), *Hybrid metaheuristics: HM 2010*, lecture notes in computer science (Vol. 6373, pp. 133–147). Berlin: Springer.
Widl, M., & Musliu, N. (2014). The break scheduling problem: Complexity results and practical algorithms. *Memetic Computing*, *6*(2), 97–112.