Math. Program., Ser. B 98: 445–476 (2003)

Digital Object Identifier (DOI) 10.1007/s10107-003-0413-7

Alberto Caprara · Michele Monaci · Paolo Toth

Models and algorithms for a staff scheduling problem

Received: May 31, 2002 / Accepted: January 14, 2003
Published online: April 10, 2003 – © Springer-Verlag 2003

Abstract. We present mathematical models and solution algorithms for a family of staff scheduling problems arising in real life applications. In these problems, the daily assignments to be performed are given and the durations (in days) of the working and rest periods for each employee in the planning horizon are specified in advance, whereas the sequence in which these working and rest periods occur, as well as the daily assignment for each working period, have to be determined. The main objective is the minimization of the number of employees needed to perform all daily assignments in the horizon.

We decompose the problem into two steps: the definition of the sequence of working and rest periods (called *pattern*) for each employee, and the definition of the daily assignment to be performed in each working period by each employee. The first step is formulated as a covering problem for which we present alternative ILP models and exact enumerative algorithms based on these models. Practical experience shows that the best approach is based on the model in which variables are associated with feasible patterns and generated either by dynamic programming or by solving another ILP. The second step is stated as a feasibility problem solved heuristically through a sequence of transportation problems. Although in general this procedure may not find a solution (even if one exists), we present sufficient conditions under which our approach is guaranteed to succeed. We also propose an iterative heuristic algorithm to handle the case in which no feasible solution is found in the second step.

We present computational results on real life instances associated with an emergency call center. The proposed approach is able to determine the optimal solution of instances involving up to several hundred employees and a working period of up to 6 months.

Key words. staff scheduling – integer linear programming – column generation – branch-and-bound – maximum flow – heuristic algorithms – computational results

1. Introduction

Staff scheduling problems are frequently encountered in the Operations Research literature. These problems require the definition of the work to be performed in a given planning horizon by employees working in companies (e.g., hospitals, fire departments, production plants, call centers, transportation companies, etc.) which are typically active from 16 to 24 hours every day. The problem is often approached by first defining the *short-term assignments* for the employees (Phase 1), corresponding to the work to be performed without interruption within a short period (typically one day), and then composing these short-term assignments into *long-term assignments* (Phase 2), so as to take care of the rests for the employees, corresponding to the overall work in the planning horizon.

A. Caprara, M. Monaci, P. Toth: Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento, 2 - 40136 - Bologna (Italy), e-mail: {acaprara,mmonaci, ptoth}@deis.unibo.it.

Mathematics Subject Classification (2000): 90B70, 90C10, 90C27, 90C39, 90C57, 90C59

For the relevant case of traveling personnel for transportation companies, there are many possible ways of defining the short-term assignments (called *pairings* in that context) and the rules that make such assignments feasible are quite complicated. This makes Phase 1 often the most important within these applications, also considering that, once the short-term assignments have been determined, Phase 2 is less critical, although still NP-hard and challenging from a research viewpoint (see e.g., [13, 4, 17]).

In many other cases, there is a small list of possible short-term assignments, e.g., either from 6 AM to 2 PM, from 2 PM to 10 PM or from 10 PM to 6 AM. In these cases, Phase 1 is essentially already solved and the relevant part is Phase 2. In our case, all the short-term assignments we consider have a duration of less than one day (typically 8 hours) and are called *duties*. We call *assignment* the work to be performed by some employee in the planning horizon and *workload* the overall work to be performed in some day of the planning horizon by the employees, i.e., the set of duties required on that day.

Surveys of the vast literature on Staff Scheduling can be found in [27, 3, 16]. In general, the solution to the problem is subdivided into five stages: (i) determination of the duties (corresponding to Phase 1 above), (ii) determination of the (minimum) number of employees required to "cover" all the duties, (iii) definition of the rest periods, (iv) definition of the sequence of working days and rest periods for each employee, and (v) definition of the duties for each employee (in the days in which he/she works).

In many cases, once Stages (i) and (ii) have been solved, the number of employees being m, the long-term assignment spans m weeks. The assignment of each employee repeats cyclically every m weeks and, in each week, employee $i + 1$ has the same assignment as employee i in the previous week. In other cases, employees have different characteristics and therefore must be assigned different assignments. In this paper, we consider a real life problem in which all employees are considered equal but their assignments may be different during the planning horizon, although the number of working days and the durations of the rest periods are the same for each employee.

More specifically, in our case, Stage (i) has already been solved, and we must cope with Stages (ii)–(v). For each employee, there is a predetermined number of *working periods* and *rest periods*, each with an associated duration (in days), and we have to decide the sequence of working and rest periods along with the duty performed in each working period (the duty must be the same for all days of the working period). A formal definition of the problem along with a concrete real life example are given in Section 2.

Our approach finds the solution of the problem in two steps. In Step 1, corresponding to Stages (ii), (iii) and (iv) and addressed in Section 3, we determine the minimum number of employees and the associated *pattern* for each employee, i.e., the days in which the employee works (*without* specifying the corresponding duty) and the days in which he/she has a rest, so as to guarantee that at least the required number of employees are working on each day. This step is approached by formulating the problem as an *Integer Linear Program* (ILP) and finding an optimal solution by branch-and-bound, possibly combined with column generation. In Step 2, corresponding to Stage (v) and discussed in Section 4, we associate a duty with each working period so as to ensure feasibility while balancing the total amount of work among the employees. We solve this step as a sequence of *Transportation Problems*, one for each day of the planning horizon. If no feasible solution is found we iteratively apply Steps 1 and 2 (suitably changing the workload) until an overall feasible solution is found.

Decomposition, that does not necessarily lead to an optimal solution, is motivated by the fact that the difficult part is the first one, whereas assigning duties to the employees once the corresponding patterns have been fixed is easy in our real life application. On the other hand, an ILP formulation for the whole problem would be considerably larger (and harder to handle) than the one we use in Step 1.

In Section 5 we present extensive computational experiments on a real life application of our problem arising in an emergency call center. At present, the number and size of call centers in Europe is rapidly increasing, and call centers have become an important source of staff scheduling problems for Operations Research practitioners. The call center we consider belongs to an Italian company providing SOS electronic devices, and receives emergency calls from a pushbutton SOS telephone system, containing an automatic dialer. The problem is to schedule the employees that answer emergency calls 24 hours a day.

To the best of our knowledge, our problem was not addressed in the literature before. This is not surprising, since different contexts typically give rise to (sometimes substantially) different staff scheduling problems. However, ILP approaches possibly combined with column generation were used in many other staff scheduling applications [21, 12, 25, 1, 8, 20, 7, 26, 10, 22, 9, 2, 11, 24], to cite the most recent ones. Our main contribution is the design of effective column generation procedures for our specific problem, that lead to a fast solution of Step 1. Moreover, the determination of the duties corresponding to the working periods, solved in Step 2, is a combinatorial problem with a simple structure, but we could not find it mentioned in the literature. Our main contribution is the derivation of sufficient conditions under which the problem has a solution, along with a heuristic for the general case that is effective for our real life instances.

2. The staff scheduling problem

We consider a planning horizon of n days, that are denoted by $N := \{1, \ldots, n\}$. The duties to be performed each day, i.e., the output of Phase 1 (also called Stage (i)) mentioned in the introduction, are actually already fixed by the company and are an input to our problem. In particular, there are n_d different duties $\{1, \ldots, n_d\}$ (generally having the same duration). For $q = 1, \ldots, n_d$ and $j \in N$, duty q requires e_{qj} employees for day j. We let f_j denote the overall number of employees that must work on day j, i.e., $f_j := \sum_{q=1}^{n_d} e_{qj}$. As already mentioned, all employees are considered equal, even if initial conditions may be imposed for some of them (see below).

Our aim is to solve Phase 2 mentioned in the introduction, i.e., to arrange the duties into assignments for the employees, with the following constraints.

- Each employee performs working periods of consecutive days, called *blocks* in the sequel, and after each block has a rest of consecutive days, called simply *rest* in the sequel. Within each block, the employee performs the *same* duty. If the duty of each day of the block is q, we say that the block is of *color* q.
- There are n_t block *types*, the tth having a duration (in days) equal to k_t. Each employee must perform exactly b_t blocks of type t for $t = 1, \ldots, n_t$.

– There are n_r rest *types*, the rth having a duration (in days) equal to d_r. Each employee must have exactly a_r rests of type r for $r = 1, \ldots, n_r$.
– There is a list of *infeasible sequences* of the form $\{(t_1, q_1), r, (t_2, q_2)\}$, with $t_1, t_2 \in \{1, \ldots, n_t\}, q_1, q_2 \in \{1, \ldots, n_d\}$ and $r \in \{1, \ldots, n_r\}$, meaning that it is not possible to have a block of type t_1 and color q_1 followed by a block of type t_2 and color q_2, with a rest of type r in between.
– Each employee works for at most s days among those in a given set S of *special days*, generally the Sundays and the other holidays within the planning horizon. (The week day corresponding to the first day of the planning horizon is specified on input.)

Sometimes we will consider the situation in which the constraint on the number of working special days is not active. We will simply say that $S = \emptyset$ in this case.

Note that block and rest types are uniquely defined by their duration, and that the above constraints are consistent only if the total number of blocks is equal to the total number of rests and the total number of working and rest days is equal to the number of days in the planning horizon, i.e., $\sum_{t=1}^{n_t} b_t = \sum_{r=1}^{n_r} a_r$ and $\sum_{t=1}^{n_t} k_t b_t + \sum_{r=1}^{n_r} d_r a_r = n$. In addition, for each employee, a block or a rest can be split between the first and the last days of the planning horizon (e.g., a block of duration 3 can span days $1, n-1, n$). In this case, different colors may be assigned to the two parts of the block. Conversely, if the first and last day of the planning horizon are both working days or both rest days, they must belong to the same block or rest, respectively. As some of the employees can be already working before the beginning of the planning horizon, for these employees *initial conditions* specify a block of type t to be performed by the employee for up to the first $k_t - 1$ days, along with the associated color, or the presence of a rest of type r that terminates within the first $d_r - 1$ days – in this case, the initial conditions specify also the type and color of the last block performed by the employee.

The natural objective for the problem is the minimization of the global number of employees (Stage (ii)).

We now show that our problem is strongly NP-hard, even in a very special case.

Proposition 1. *The problem of determining whether* one *employee can cover all the duties is strongly NP-complete, even if $n_d = 1$, $n_t = 1$, there are no infeasible sequences, and $S = \emptyset$.*

Proof. We illustrate an easy polynomial reduction from the well-known strongly NP-complete *Three Partitioning Problem* (3PP) [18], where one wants to check whether $3m$ items, the jth of integer size $s_j \in (c/4, c/2)$ $(j = 1, \ldots, 3m)$, with $\sum_{j=1}^{3m} s_j = m \cdot c$, can be packed (in triples) into m identical bins of positive integer capacity c.

We let the number of days be $n := (c + 3) \cdot m$ and $f_j := 1$ for $j = c + 3, 2(c + 3), \ldots, m(c + 3)$; $f_j := 0$ for the other days, i.e., only every $c + 3$ days one employee is requested. Here, each group of $c + 3$ consecutive days (starting from day 1) represents a bin. All blocks have the same duration, equal to 1 day, i.e., $n_t := 1$ and $k_1 := 1$. Moreover, there are $3m$ rest types, the jth of duration $d_j := s_j$ days. This means that each employee works for $3m$ days in the planning horizon. Since the duration of each rest is in $(c/4, c/2)$, it is easy to check that the only way for an employee to cover days $i(c + 3)$ and $(i + 1)(c + 3)$ for some $i = 1, \ldots, m$ is to have three rests whose overall duration is c (and two blocks without work) between two blocks on the two given days.

This shows that all the workload can be covered by one employee if and only if there exists a feasible solution to the original 3PP instance, yielding the proof.

In our application, employees have to be present 24 hours a day; every day 14 employees have to work from 6 AM to 2 PM, 14 from 2 PM to 10 PM and 14 from 10 PM to 6 AM. Moreover, from Monday to Saturday, 7 employees have to work from 9 AM to 5 PM, 7 from 10:30 AM to 6:30 PM, and 7 from 12:30 PM to 8:30 PM. The set S of special days includes all Sundays. According to our notation, we have $n_d := 6$, $e_{qj} := (14, 14, 14, 0, 0, 0)$ for $j \in S$ and $e_{qj} := (14, 14, 14, 7, 7, 7)$ for $j \in N \setminus S$. This means that $f_j = 63$ if j is not a Sunday and $f_j = 42$ otherwise. Table 1 gives the resulting number of employees required at each time window.

For each pair of consecutive blocks performed by an employee with a 1-day rest in between, the duty of at least one of the blocks must be different from 10 PM–6 AM. This is the only infeasible sequence of our case study.

The length of the planning horizon is $n := 112$ days (i.e., almost 4 months). All blocks have the same duration $k := 3$, i.e., $n_t := 1$ and $k_1 := 3$. The number of blocks $b_1 = b$ in each pattern is determined by the constraint that each employee must work for up to 38 hours a week (on average). Considering that the duration of each duty is 8 hours, this means that the number of working days must not exceed $\frac{38 \cdot n}{7 \cdot 8}$, i.e., $b := \lfloor \frac{38 \cdot n}{7 \cdot 8 \cdot 3} \rfloor$. (On average, each employee works 6–7 days out of 10 within the planning horizon.) Hence, for our application, $b := 25$. As for the rests, each pattern must have one rest of 5 days, while the duration of the remaining $b - 1$ rests is either 1 or 2. In particular, we have 16 rests of 1 day, and 8 rests of 2 days, i.e., according to our notation, $n_r := 3, d := (1, 2, 5)$ and $a := (16, 8, 1)$. The maximum number s of working special days (Sundays) is 2/3 of the number of weeks (rounded down), i.e., $s := 10$.

3. Step 1: finding the patterns

For notational convenience, unless otherwise specified, in this section all indices for the days are understood to be modulo n (letting $h \cdot n \bmod n := n$ for any integer h, so as to have the days numbered from 1 to n). Moreover, we let $M := \{1, \dots, m\}$ denote the set of employees available (supposing m employees are sufficient to cover all the workload), $T := \{1, \dots, n_t\}$ the set of block types, and $R := \{1, \dots, n_r\}$ the set of rest types.

We present two mathematical formulations for this step. The first one, illustrated in Section 3.1, is descriptive, having decision variables for each employee specifying when he/she starts a block or a rest of a certain type, whereas the second one, illustrated in Section 3.2, is of the covering type (frequently found in the literature, see, e.g., [6, 5]) and has one variable for each feasible pattern for an employee. In Section 3.3 we discuss

Table 1. Number of employees required in our application.

	12AM–9	9–10:30	10:30–12:30PM	12:30–5	5–6:30	6:30–8:30	8:30–12AM
day $\in S$	14	14	14	14	14	14	14
day $\notin S$	14	21	28	35	28	21	14

how to handle the possibly huge number of variables of this latter formulation, and in Section 3.4 we compare the lower bounds provided by the *Linear Programming* (LP) relaxations of the different formulations.

Note that $\lceil L_0 \rceil$ is an obvious lower bound on the minimum number of employees, where

$$L_0 := \max \left\{ \max_{j \in N} f_j, \frac{\sum_{j \in N} f_j}{\sum_{t \in T} k_t b_t}, \frac{\sum_{j \in S} f_j}{s} \right\}. \tag{1}$$

This means that the number of employees must be at least equal to the maximum daily request, to the ratio between the overall number of duties in the planning horizon and the overall number of working days for each employee, and to the ratio between the overall number of duties in the special days and the maximum number of working special days for each employee.

We will show that the lower bounds associated with the LP relaxation of our formulations dominate this bound, but also that, in case of constant workload, the optimal LP values coincide with (1).

3.1. A descriptive formulation

In our first ILP formulation, we use the following natural binary variables:

$$z_i = \begin{cases} 1 \text{ if employee } i \text{ is active} \\ 0 \text{ otherwise} \end{cases} \qquad (i \in M),$$

$$x_{ij}^t = \begin{cases} 1 \text{ if employee } i \text{ starts a block of type } t \text{ on day } j \\ 0 \text{ otherwise} \end{cases} \qquad (i \in M;\ j \in N;\ t \in T),$$

and

$$w_{ij}^r = \begin{cases} 1 \text{ if employee } i \text{ starts a rest of type } r \text{ on day } j \\ 0 \text{ otherwise} \end{cases} \qquad (i \in M;\ j \in N;\ r \in R).$$

Note that an employee works on day j if and only if he/she starts a working block of type t (for some $t \in T$) in one of the days $j - k_t + 1, \ldots, j$. Analogously, an employee is on rest on day j if and only if he/she starts a rest of type r (for some $r \in R$) in one of the days $j - d_r + 1, \ldots, j$.

Note also that, if we used binary variables $\tilde{x}_{ij}^t$ taking value 1 if employee i performs a block of type t on day j, it would be complicated to express the fact that the employee actually works for k_t consecutive days. Moreover, it is easy to express $\tilde{x}_{ij}^t$ as a linear function of the x_{ij}^t, namely $\tilde{x}_{ij}^t = \sum_{h=j-k_t+1}^{j} x_{ih}^t$, whereas the converse is false. Clearly, the same holds for binary variables $\tilde{w}_{ij}^r$, indicating whether employee i is on a rest of type r on day j.

The problem can be formulated in the following way:

$$\min \sum_{i \in M} z_i \tag{2}$$

$$\sum_{i\in M}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t \geq f_j \quad (j \in N) \tag{3}$$

$$\sum_{j\in N} x_{ij}^t = b_t z_i \quad (i \in M; t \in T) \tag{4}$$

$$\sum_{j\in N} w_{ij}^r = a_r z_i \quad (i \in M; r \in R) \tag{5}$$

$$\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t + \sum_{r\in R}\sum_{h=j-d_r+1}^{j} w_{ih}^r = z_i \quad (i \in M; j \in N) \tag{6}$$

$$\sum_{t\in T} x_{ij}^t - \sum_{r\in R} w_{i(j-d_r)}^r = 0 \quad (i \in M; j \in N) \tag{7}$$

$$\sum_{t\in T} x_{i(j-k_t)}^t - \sum_{r\in R} w_{ij}^r = 0 \quad (i \in M; j \in N) \tag{8}$$

$$\sum_{j\in S}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t \leq s z_i \quad (i \in M) \tag{9}$$

$$z_i \in \{0,1\} \quad (i \in M) \tag{10}$$

$$x_{ij}^t \in \{0,1\} \quad (i \in M;\ j \in N; t \in T) \tag{11}$$

$$w_{ij}^r \in \{0,1\} \quad (i \in M;\ j \in N; r \in R). \tag{12}$$

Inequalities (3) guarantee that at least f_j employees are working on day j. Equations (4) and (5) guarantee that each active employee performs b_t blocks of type t and has a_r rests of type r during the planning horizon, respectively. Equations (6) guarantee that, on each day j of the planning horizon, each active employee is either performing a block or a rest. Equations (7) guarantee that an employee starts a block of some type on day j if and only if he/she starts a rest of type r on day $j - d_r$ for some $r \in R$ (and therefore ends the rest on day $j - 1$). Similarly, equations (8) impose that an employee starts a rest of some type on day j if and only if he/she starts a block of type t, for some $t \in T$, on day $j - k_t$ (and therefore ends the block on day $j - 1$). Finally, inequalities (9) ensure that each employee works for at most s special days within the planning horizon.

Possible initial conditions can be incorporated in the above model as follows. For all $t \in T$ and $j \in \{0, \ldots, k_t - 1\}$, let A_{jt} denote the set of employees that end a block of type t on day j. Analogously, for all $r \in R$ and $j \in \{0, \ldots, d_r - 1\}$, let B_{jr} denote the set of employees that end a rest of type r on day j. Note that all the A_{jt} and B_{jr} sets form a partition of the set of employees for which an initial condition is specified (in particular these sets are pairwise disjoint). Initial conditions are imposed by forcing $x_{i(j-k_t+1)}^t = 1$ for $i \in A_{jt}$ and $w_{i(j-d_r+1)}^r = 1$ for $i \in B_{jr}$.

According to the above discussion, the solution of ILP (2)–(12) (with the possible addition of the initial conditions) by a general-purpose ILP solver yields an optimal set of patterns.

In Appendix A, we give an illustration of additional valid inequalities that can be used to strengthen the LP relaxation of (2)–(12). In particular, we show that there are polynomially many *clique* inequalities that can be added to this LP, namely

$$\sum_{t\in T}\sum_{h=j-k_t+k_{\min}}^{j+\ell} x_{ih}^t + \sum_{r\in R}\sum_{h=j-d_r+\ell+1}^{j+k_{\min}-1} w_{ih}^r \le z_i$$
$$(i \in M;\ j \in N;\ \ell = -1, \ldots, k_{\min} + d_{\min} - 1), \tag{13}$$

where $k_{\min} := \min_{t\in T} k_t$ and $d_{\min} := \min_{r\in R} d_r$ and with the convention that, if for some $t \in T$ or $r \in R$, the up index of the inner summation is smaller than the down one, no term has to be considered in the summation.

Generally, infeasible sequences involve only blocks of specified colors, and therefore they cannot be imposed in Step 1. However, if the sequence $\{(t_1, q_1), r, (t_2, q_2)\}$ is infeasible for all color pairs (q_1, q_2), the following simple constraints can be added to formulation (2)–(12):

$$x_{ij}^{t_1} + w_{i,j+k_{t_1}}^r + x_{i,j+k_{t_1}+d_r}^{t_2} \le 2 \quad (i \in M;\ j \in N).$$

These constraints can be strengthened through a standard *lifting* procedure.

3.2. A covering formulation

In order to define the second model, we let $\mathcal{P}$ be the set of all feasible patterns for an employee. In particular, each pattern can be represented by a 0–1 vector with n entries, the jth equal to 1 if the employee is working on day j and 0 otherwise. By defining

$$y_P = \text{number of employees who are performing pattern } P \quad (P \in \mathcal{P}),$$

we obtain the following ILP formulation:

$$\min \sum_{P\in\mathcal{P}} y_P \tag{14}$$

$$\sum_{P\in\mathcal{P}_j} y_P \ge f_j \quad (j \in N) \tag{15}$$

$$y_P \ge 0, \text{integer} \quad (P \in \mathcal{P}), \tag{16}$$

where $\mathcal{P}_j \subset \mathcal{P}$ denotes the set of feasible patterns that require working on day j. Note that, when $S = \emptyset$, given the 0–1 vector $v = (v_1, \ldots, v_n)$ representing a feasible pattern $P \in \mathcal{P}$, for every $j \in N$ the vector of the form $(v_{j+1}, v_{j+2}, \ldots, v_n, v_1, \ldots, v_j)$ (obtained by shifting v by j days) represents a feasible pattern as well.

For small instances, it is possible to explicitly generate all patterns in $\mathcal{P}$ and to include all the corresponding variables into the model. In other cases, one must resort to column generation techniques, which are discussed in Section 3.3. In the following, when we refer to model (14)–(16), we will use the terms "pattern", "variable" and "column" as synonyms.

The additional constraints to impose possible initial conditions are the following:

$$\sum_{P \in \mathcal{C}_{jt}} y_P \geq |A_{jt}| \quad (t \in T;\, j \in \{0, \dots, k_t - 1\}) \tag{17}$$

$$\sum_{P \in \mathcal{D}_{jr}} y_P \geq |B_{jr}| \quad (r \in R;\, j \in \{0, \dots, d_r - 1\}). \tag{18}$$

where $\mathcal{C}_{jt} \subset \mathcal{P}$ and $\mathcal{D}_{jr} \subset \mathcal{P}$ denote the set of feasible patterns for which a block of type t and a rest of type r ends on day j, respectively. Constraints (17) and (18) require the selection of at least $|A_{jt}|$ and $|B_{jr}|$ patterns ending a block of type t and a rest of type r on day j, respectively.

3.3. Column generation

The straightforward way to solve ILP (14)–(16) is by a general-purpose ILP solver. In this section, we discuss three different methods to handle the LP relaxation of this model when the number of variables is too large to have all of them explicitly in the model. In Section 5 we briefly illustrate how we incorporated these methods into a branch-and-bound algorithm.

The dual of the LP relaxation of (14)–(16) reads

$$\max \sum_{j \in N} f_j\, \pi_j \tag{19}$$

$$\sum_{j \in J_P} \pi_j \leq 1 \quad (P \in \mathcal{P}) \tag{20}$$

$$\pi_j \geq 0, \quad (j \in N). \tag{21}$$

where J_P denotes the set of working days for each pattern $P \in \mathcal{P}$. Column generation amounts to checking if a given dual solution π^* violates some of the constraints (20), i.e., if the corresponding primal variable has negative reduced cost. If each day j is assigned a *profit* π_j^*, the problem corresponds to finding (if any) a pattern whose working days have an overall profit greater than 1.

The first method, referred to as *Pricing -CG* in the sequel, is the explicit generation of all variables and the use of a *pricing* technique that computes the reduced costs for all variables not in the current model and adds (a subset of) those with negative reduced cost. Since there may be many such variables, it is often not necessary to compute all reduced costs in each pricing iteration. More details about this technique will be given in Section 5. In any case, this method can be applied in practice as long as the overall number of feasible patterns does not exceed, say, a few millions.

The second method, referred to as ILP-CG in the sequel, generates columns with negative reduced costs (if any) by using a simple variant of model (2)–(12). In particular, we are looking for a single pattern and therefore $M := \{1\}$, the objective function reads

$$\max \sum_{j \in N} \sum_{t \in T} \left(\sum_{h=j}^{j+k_t-1} \pi_h^* \right) x_{1j}^t, \tag{22}$$

variable z_1 is fixed to 1 and constraints (3) are not present. We also consider the variant of this approach, denoted by *ILPC-CG*, with the addition of the clique constraints (13).

The third method, referred to as *DP-CG* in the sequel, is based on dynamic programming. To this aim, we define a table giving the maximum profit that can be achieved in days from 1 to j by a pattern that, within these days, contains a certain number of blocks and rests of each type and works for a certain number of special days. Formally, the entries of the table are of the form

$$\rho(j, \overline{s}, \overline{b}_1, \ldots, \overline{b}_{n_t}, \overline{a}_1, \ldots, \overline{a}_{n_r}),$$

defined for $j = 1, \ldots, n+\theta$; $\overline{s} = 0, \ldots, s$; $\overline{b}_t = 0, \ldots, b_t$ $(t \in T)$; $\overline{a}_r = 0, \ldots, a_r$ $(r \in R)$, where $\theta := \max_{t \in T} k_t + \max_{r \in R} d_r$. Here, the day indices are *not* modulo n. The extended range for the values of j with respect to N is due to the fact that blocks and rests can be split between the first and the last days of the planning horizon. The computation of the entries in the table is carried out by initializing $\rho(j, 0, \ldots, 0) := 0$, for $j = 1, \ldots, \theta$. The remaining entries are initially set to $-\infty$ and computed from the initial ones by considering the insertion of a block starting on day j followed by a rest in the pattern associated with an entry $\rho(j, \ldots)$ already computed. More precisely, for each entry $\rho(j, \overline{s}, \overline{b}_1, \ldots, \overline{b}_{n_t}, \overline{a}_1, \ldots, \overline{a}_{n_r})$ already computed and such that $j \leq n$, we consider all pairs $t \in T$ and $r \in R$ such that $\overline{b}_t < b_t$, $\overline{a}_r < a_r$, and the number s' of special days contained in $\{j, \ldots, j + k_t - 1\}$ satisfies $\overline{s} + s' \leq s$. We then set the value of

$$\rho(j + k_t + d_r, \overline{s} + s', \overline{b}_1, \ldots, \overline{b}_t + 1, \ldots, \overline{b}_{n_t}, \overline{a}_1, \ldots, \overline{a}_r + 1, \ldots, \overline{a}_{n_r})$$

to the maximum between the current value and

$$\rho(j, \overline{s}, \overline{b}_1, \ldots, \overline{b}_t, \ldots, \overline{b}_{n_t}, \overline{a}_1, \ldots, \overline{a}_r, \ldots, \overline{a}_{n_r}) + \sum_{h=j}^{j+k_t-1} \pi_h^*.$$

All the entries of the form $\rho(j, \overline{s}, b_1, \ldots, b_{n_t}, a_1, \ldots, a_{n_r})$ with $j \in \{n + 1, \ldots, n+\theta\}$ and $\overline{s} \in \{0, \ldots, s\}$ correspond to profits of feasible patterns. Hence the maximum profit of a pattern is given by the maximum of these entries, and all entries with profit greater than 1 correspond to columns with negative reduced cost. The associated patterns can be reconstructed by storing the predecessor of each entry of the table in a standard way.

The overall space complexity of the dynamic programming procedure is

$$O(n \cdot s \cdot \prod_{t \in T} b_t \cdot \prod_{r \in R} a_r),$$

whereas the time complexity is

$$O(n \cdot s \cdot n_t \cdot n_r \cdot \prod_{t \in T} b_t \cdot \prod_{r \in R} a_r),$$

since each entry of the table is used to update up to $O(n_t \cdot n_r)$ other entries.

It is simple to adapt the column generation methods above in case infeasible sequences and/or initial conditions are imposed.

3.4. Comparison of the lower bounds

In this section, we show the dominance relations between the lower bounds on the minimum number of employees given by L_0, defined by (1), the value L_1 of the LP relaxation of (2)–(12), the value L_2 of the LP relaxation of (2)–(12) with the addition of the clique constraints (13), and the value L_3 of the LP relaxation of (14)–(16). We also show that, in the (relevant) case in which the workload is constant and the set of special days is empty, all these bounds coincide.

Proposition 2. *$L_3 \geq L_2 \geq L_1 \geq L_0$. Moreover, if $f_j = f$ for $j \in N$ and $S = \emptyset$, all the inequalities hold as equalities.*

Proof. The relation $L_3 \geq L_2$ follows from a general result of [19], which states that L_3 is the optimal value of the LP obtained from (2)-(12) by adding all inequalities of the form

$$\sum_{t\in T}\sum_{j\in N}\gamma_j^t x_{ij}^t + \sum_{r\in R}\sum_{j\in N}\delta_j^r w_{ij}^r \leq \varepsilon z_i \quad (i \in M), \tag{23}$$

where $\sum_{t\in T}\sum_{j\in N}\gamma_j^t x_{ij}^t + \sum_{r\in R}\sum_{j\in N}\delta_j^r w_{ij}^r \leq \varepsilon$ is a valid inequality for the convex hull of the 0-1 vectors (x_{ij}^t), (w_{ij}^r) associated with a feasible pattern for a generic employee i (the convex hull is clearly independent of i). In particular, inequalities (23) include all clique inequalities (13).

Relation $L_2 \geq L_1$ is obvious. We next show that $L_1 \geq L_0$. Note that equations (6) imply that, for each $i \in M$, $j \in N$, $\sum_{t\in T} \sum_{h=j-k_t+1}^{j} x_{ih}^t \leq z_i$. Hence, for each day $j \in N$

$$\sum_{i\in M} z_i \geq \sum_{i\in M}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t \geq f_j$$

where the second inequality follows from (3). This shows $L_1 \geq \max_{j\in N} f_j$. Moreover

$$\begin{aligned}\left(\sum_{t\in T} k_t b_t\right)\left(\sum_{i\in M} z_i\right) &= \sum_{i\in M}\sum_{t\in T} k_t b_t z_i = \sum_{i\in M}\sum_{t\in T} k_t \sum_{j\in N} x_{ij}^t = \sum_{i\in M}\sum_{t\in T}\sum_{j\in N} k_t x_{ij}^t \\ &= \sum_{i\in M}\sum_{t\in T}\sum_{j\in N}\sum_{h=j-k_t+1}^{j} x_{ih}^t = \sum_{j\in N}\sum_{i\in M}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t \geq \sum_{j\in N} f_j,\end{aligned}$$

where the second equality follows from (4), the fourth equality from the easily verified relation $\sum_{j\in N} k_t x_{ij}^t = \sum_{j\in N}\sum_{h=j-k_t+1}^{j} x_{ih}^t$, and the last inequality from (3). This shows $L_1 \geq \frac{\sum_{j\in N} f_j}{\sum_{t\in T} k_t b_t}$. Finally,

$$s\sum_{i\in M} z_i = \sum_{i\in M} s z_i \geq \sum_{i\in M}\sum_{j\in S}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t = \sum_{j\in S}\sum_{i\in M}\sum_{t\in T}\sum_{h=j-k_t+1}^{j} x_{ih}^t \geq \sum_{j\in S} f_j,$$

where the first inequality follows from (9) and the second from (3). This shows $L_1 \geq \frac{\sum_{j\in S} f_j}{s}$.

We conclude the proof by showing that $L_3 = L_0$ if f_j is equal to a constant f for each $j \in N$ and $S = \emptyset$. Note that in this case the third term in (1) is undefined (and should be considered equal to 0) and

$$\max_{j \in N} f_j = f \leq \frac{n \cdot f}{\sum_{t \in T} k_t b_t} = \frac{\sum_{j \in N} f_j}{\sum_{t \in T} k_t b_t},$$

i.e., the maximum in (1) is given by the second term. For each 0-1 vector v representing a feasible pattern $P \in \mathcal{P}$, the number of 1s is $\sum_{t \in T} k_t b_t$. Consider an arbitrarily chosen such vector and all the vectors that can be obtained by shifting it by j days, $j \in N$, letting $P_1, \ldots, P_n$ be the corresponding patterns (that are all feasible since $S = \emptyset$). It is simple to verify that the solution of the LP relaxation of (14)–(16) in which the only nonzero variables are y_{P_j} ($j \in N$), each equal to $\frac{f}{\sum_{t \in T} k_t b_t}$, is feasible and has value equal to $\frac{n \cdot f}{\sum_{t \in T} k_t b_t}$, i.e., all inequalities in the statement of the proposition are tight in this case.

For many problems, the LP relaxation of a covering formulation like (14)–(16) yields on average significantly better bounds than the one of a descriptive ILP model like (2)–(12).

However, this is not the case here since, as illustrated in Section 5, L_3 and L_1 (and hence L_2) coincide for almost all the instances in our test bed. The discussion in Appendix B is aimed at showing why this is not surprising. In any case, Section 5 also shows that model (14)–(16) (possibly handled by column generation) turns out to be much more effective than (2)–(12) in practice, even when the LP lower bounds coincide. This situation is analogous to that of multicommodity flow problems, in which a descriptive formulation with one variable for each arc-commodity pair, and a packing formulation with one variable for each path, have the same LP relaxation value, but the latter is much better in practice even if it has an exponential number of variables (see, e.g., [14]).

4. Step 2: assigning block colors

Given the patterns found in the previous step, each associated with an employee, in the second step we define the color of the blocks in each pattern. Here, the main objective is to find a feasible solution, called *feasible color assignment*. Note that there may be no feasible color assignment associated with a given feasible set of patterns, even if a feasible solution of the overall problem with the same number of employees exists (see the examples in Tables 2 and 3). Moreover, our approach to this second step is heuristic in nature, in that it does not guarantee finding a feasible color assignment even if such an assignment exists. Nevertheless, in some relevant special cases our method surely finds a feasible assignment. We first discuss conditions for the existence of a feasible color assignment in Section 4.1, and then present our approach in Section 4.2, illustrating in Section 4.3 how we handle the case in which no feasible solution is found, so as to obtain a heuristic solution for the overall problem.

As already mentioned in Section 2, for a block spanning days 1 and n, the two parts in which it is split need not have the same color. Correspondingly, such a block is replaced by *two separate parts* (each called *block* in the sequel) that may be assigned different colors. This yields sufficient conditions for the existence of a feasible color assignment, discussed in Section 4.1. Note finally that there may be a feasible color assignment in which some blocks need not have a color, since the presence of the corresponding employee is not required for any day of the block in order to reach the requested number of employees on that day. In this case, the employee may be given k_t days off instead of performing a block of type t. We call these blocks *dummy blocks*.

Table 2 shows a feasible set of patterns found in Step 1 (with the parameters of our specific application, see Section 2) for three employees for the first 15 days of the planning horizon. Assume that no infeasible sequences exist and that there are two duties: duty 1 to be performed every day of the planning horizon by one employee and duty 2 to be performed all days except Sundays (i.e., days 1, 8, 15) by one employee. Clearly, the color of the block performed by employee 1 on day 1 must be 1. This forces the blocks starting on days 2, 3, 4, 6, 7, 9, 10, 12, 13, to be, respectively, of color 2, 1, 2, 1, 2, 1, 2, 1, 2, and in particular imposes that the block performed by employee 1 on day 15 is of color 2, which is infeasible since no employee then performs duty 1 on day 15.

A set of patterns leading to a feasible color assignment, at least for the first 15 days, is given in Table 3, where only the pattern for employee 3 is changed with respect to Table 2 (see line 3′). Indeed, there exists a feasible color assignment in which employees 1 and 2 perform the same duties as before, while employee 3′ performs blocks of color 2, 1, 2 and 1 starting on days 1, 6, 10 and 15, respectively.

We observe that the problem of checking whether there exists a feasible color assignment is strongly NP-complete.

Proposition 3. *The problem of determining whether there exists a feasible color assignment is strongly NP-complete, even if $n_t = 1$ and the workload is constant.*

Proof. We illustrate an easy polynomial reduction from the well-known strongly NP-complete *Edge Coloring Problem for 3-regular graphs* (ECP) [18], where one wants to check whether the edges of an undirected graph $G = (V, E)$, in which each vertex has degree three, can be colored with three colors so that no two edges incident with a same vertex receive the same color.

Table 2. A feasible set of patterns which has no feasible color assignment.

		Day														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	x	x	–	x	x	x	–	–	x	x	x	–	x	x	x
Empl.	2	–	–	x	x	x	–	x	x	x	–	–	x	x	x	–
	3	–	x	x	x	–	x	x	x	–	x	x	x	–	–	–

Table 3. A feasible set of patterns which is a simple variation of that of Table 2 and has a feasible color assignment.

		Day														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	x	x	–	x	x	x	–	–	x	x	x	–	x	x	x
Empl.	2	–	–	x	x	x	–	x	x	x	–	–	x	x	x	–
	3’	x	x	x	–	–	x	x	x	–	x	x	x	–	–	x

We let the number of days be $n := |V|$ and the number of duties be $n_d := 3$. Moreover, each duty requires one employee every day, i.e., $e_{qj} := 1$, for $q = 1, 2, 3$ and $j = 1, \ldots, n$. We let the number of patterns (and employees) be $|E|$, namely for each edge $(i, j) \in E$, there is a pattern composed of two blocks whose duration is one day, one block in day i and the other in day j, and we define infeasible sequences forcing these two blocks to receive the same color. Note that the number of working blocks in each day is equal to three.

Observing the obvious correspondence between days and vertices, duties and edge colors, and patterns and edges, it is easy to realize that there exists a feasible color assignment if and only if one can assign the same color to the two blocks in each pattern (i.e., each edge in G receives only one color) and, for each day, a different duty is assigned to each block (i.e., the edges incident to the same vertex receive three different colors). In other words, there exists a feasible color assignment if and only if the original ECP instance has a solution.

The complexity of the problem if no infeasible sequences are imposed is open.

4.1. The case of constant workload and no infeasible sequences

There is a relevant case in which, for every feasible set of patterns, a feasible color assignment exists (and can be found efficiently). This happens when there are no infeasible sequences and the workload is *constant* during the planning horizon, i.e., according to the notation in Section 2, $e_{qj} = e_q$ for $j \in N$ and $q \in A$, where $A := \{1, \ldots, n_d\}$ denotes the set of duties. For $j \in N$, let $m(j)$ be the number of employees that are working on day j according to the given set of patterns. For convenience, we re-define the duties so that, for each duty $q \in A$, exactly *one* employee must perform q every day. This amounts to replacing each duty q that originally required e_q employees by e_q different duties with requirement one. It is easy to see that this does not change the problem since e_q does not depend on the specific day. After this re-definition we have $e_q = 1$ for $q \in A$, and n_d is the number of employees required each day.

Let n_b be the overall number of blocks. The problem of finding a feasible color assignment can be formulated as a max-flow problem, as follows. Given the set of patterns and the associated blocks, let $G = (V, E)$ be the directed graph with one vertex for each block plus a dummy source vertex s and a dummy sink vertex t. For each pair of vertices v_1 and v_2, there is an arc $(v_1, v_2) \in E$ if and only if the associated blocks, say b_1, b_2, spanning days $h_1, \ldots, j_1$ and $h_2, \ldots, j_2$, respectively, satisfy $h_1 < h_2 \leq j_1 + 1$ and $j_2 \geq j_1 + 1$, i.e., b_2 starts and ends later than b_1 (but does not start later than the day after the end of b_1). Such an arc (v_1, v_2) represents the possibility of covering some duty q with the two blocks for days $h_1, \ldots, j_2$. Moreover, E contains one arc (s, v) and one arc (v, t) for each vertex v associated with a block spanning days 1 and n, respectively. Figure 1 shows the graph G associated with the patterns in Table 3, assuming the length of the planning horizon is 15 days and also duty 2 has to be performed on Sunday: vertices $(1, \ldots, 4)$, $(5, \ldots, 7)$ and $(8, \ldots, 11)$ are associated with the blocks performed by employee 1, 2, and 3, respectively.

Accordingly, for each duty $q \in A$, in order to ensure that the assignment is performed every day by an employee, the blocks that are assigned color q in a feasible color assign-

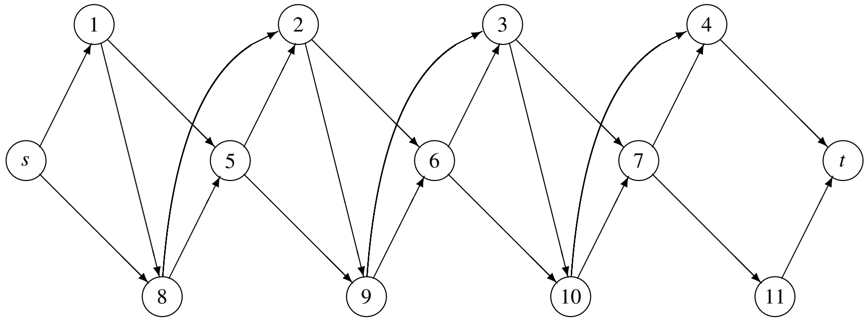

Fig. 1. Graph $G = (V, E)$ associated with the patterns in Table 3.

ment must be the vertices in a *directed path* from s to t in G. This implies that there is a feasible color assignment if and only if G contains at least n_d vertex-disjoint paths from s to t. Hence, by assigning capacity $+\infty$ to all arcs in E and capacity 1 to all vertices in $V \setminus \{s, t\}$, we have

Proposition 4. *A feasible color assignment exists if and only if the maximum flow from s to t in G has value at least n_d.*

We next show that the value of the maximum flow is at least n_d if and only if $\min_{j \in N} m(j) \geq n_d$, i.e., the minimum cut of G has value $\min_{j \in N} m(j)$, presenting a simple greedy procedure to find a feasible color assignment.

The procedure, called GREEDY_COLOR, is presented in Figure 2.

Noting $n_d \leq n_b$, procedure GREEDY_COLOR can be implemented to run in $O(n + n_b \log n_d)$ time. This time is achieved by determining, in $O(n_b)$ time, for each day $j \in N$, the blocks starting on day j (then considering the blocks according to increasing days), and using a heap for the duties, in order to determine, at each iteration, the duty which is uncovered starting from the smallest day. We next show that the algorithm is correct.

```
procedure GREEDY_COLOR
begin
    let $B$ be the set of available blocks, initially containing all blocks in the given patterns;
    repeat
        find the block $b \in B$ starting on the smallest day $j$ (if $B = \emptyset$, let $j := n + 1$);
        find the duty $q \in A$ which is uncovered starting from the smallest day $h$;
        if $h < j$ then
            failure (no feasible solution exists)
        else
            assign color $q$ to block $b$, remove $b$ from $B$, and cover duty $q$ until
                the last day of $b$
        end if
    until all duties are covered
end.
```

Fig. 2. A pseudo-code description of procedure GREEDY_COLOR.

Proposition 5. *If $m(j) \geq n_d$ for $j \in N$, then* GREEDY_COLOR *finds a feasible color assignment.*

Proof. Suppose $m(j) \geq n_d$ for $j \in N$ but GREEDY_COLOR reaches the failure state, i.e., $h < j$ within the repeat-until loop. Let q be the duty that cannot be covered. Since $m(h) \geq n_d$, there is some other duty r covered by two (or more) blocks b_1, b_2 on day h. Assume without loss of generality that b_1 was assigned color r before b_2. Then, after assigning color r to block b_1, the first day in which r is uncovered is greater than h. Hence, the color assigned to b_2 cannot be r since the first uncovered day of duty q is smaller than that of r.

Corollary 1. *The value of the minimum cut in G is* $\min_{j \in N} m(j)$.

Proposition 5 proves that every feasible set of patterns has a feasible color assignment.

If the condition of Proposition 5 is satisfied, another simple greedy algorithm, called GREEDY_COLOR_*k*, can be used to find a feasible color assignment when all blocks have the same duration k (possibly excluding those starting on the first day or ending on the last day). We present also this procedure as it is closer to the procedure presented in the next section for the real life case. GREEDY_COLOR_*k* considers one duty at a time and assigns it to the available blocks so as to minimize the number of days in which the duty is covered more than once. A pseudo-code implementation of the method is given in Figure 3.

Procedure GREEDY_COLOR_*k* can be implemented to run in $O(n_b + n \cdot k)$ time. Indeed, we first determine, in $O(n_b)$ time, the number of available blocks ψ_j starting on day j, for each day $j \in N$. Then, we define, for each day $j \in N$, the index π_j corresponding to the last day $\leq j$ in which a block spanning day j starts (possibly $\pi_j = j$). At each iteration of the repeat-until loop, the block b is found in constant time, using π_j, and ψ_{π_j} is decreased by one unit. The corresponding overall computing time is $O(n_b)$. Whenever, for some day j, the corresponding ψ_j becomes 0, π_j is set to π_{j-1} and π_h, for $h = j+1, \ldots, j+k-1$, is possibly updated. Since, for each day $j \in N$, π_j becomes 0 at most once, the overall updating of the πs requires $O(n \cdot k)$ time. A more careful implementation with *union-find* data structures as those used in Kruskal's algorithm for the minimum spanning tree (see, e.g., [15], p. 504) requires $O(n_b + n \log k)$ time, using

```
procedure GREEDY_COLOR_k
begin
    let $B$ be the set of available blocks, initially containing all blocks in the given patterns;
    for each $q \in A$ do
        $j := 1$;
        repeat
            find an available block $b \in B$ that spans day $j$ and starts as late as possible,
                letting $h$ be the last day of $b$;
            assign color $q$ to block $b$, remove $b$ from $B$;
            $j := h + 1$
        until $j > n$
    end for each
end
```

Fig. 3. A pseudo-code description of procedure GREEDY_COLOR_*k*.

separate sets for days j with the same π_j and noting that the size of each of these sets is at most k if a feasible color assignment exists. Full details about this latter approach are omitted.

Proposition 6. *If all blocks have the same duration k and $m(j) \geq n_d$ for $j \in N$,* GREEDY_COLOR_k *finds a feasible color assignment.*

Proof. The proof is by induction on n_d. If $n_d = 1$ the claim is clearly true. Supposing it is true for $n_d = g-1$, we show that it holds also for $n_d = g$. This amounts to showing the following: after having assigned blocks to duty 1, the number of employees available on each day is at least $g-1$. This is clearly true for all days in which exactly one employee performs duty 1. It is easy to check that *at most two* employees are performing duty 1 on each day. (For simplicity, we use day indices without checking if they are < 1 or $> n$ - each index j should be replaced by 1 if < 1 and by n if $> n$.) Consider a sequence of consecutive days $j, \ldots, j+l$, with $l < k-1$, in which two employees perform duty 1. This means that one block spanning days $j+l-k+1, \ldots, j+l$ and one block spanning days $j, \ldots, j+k-1$ are assigned color 1 by GREEDY_COLOR_k, and therefore no block starts on days $j+1, \ldots, j+l+1$. Hence, the, at least, g blocks spanning day $j+l+1$ are all starting on a day in $j+l-k+2, \ldots, j$, and only one of these blocks is assigned color 1, i.e., in days $j, \ldots, j+l$ there are still $g-1$ available blocks after having assigned blocks to duty 1.

4.2. A heuristic for the general case

We now present a heuristic algorithm for the case in which the workload may differ from day to day and there may be infeasible sequences. In our heuristic, we solve a sequence of *Transportation Problems* (TPs), one for each day of the given planning horizon. Note that the initial conditions may specify the duty of some employees for the first days of the planning horizon.

We consider days $1, \ldots, n$ in this order. For each day j, the duty for some employees working on day j (i.e., starting a block of type t on days $j, j-1, \ldots, j-k_t+1$) is specified either by the initial conditions or by the choices made for days $1, \ldots, j-1$. Let $e''_q(j)$ be the number of employees already performing duty q on day j, $e'_q(j) := \max\{e_{qj} - e''_q(j), 0\}$ be the number of additional employees that must perform this duty on day j, and $M'(j)$ be the set of available employees on day j, i.e., those working on day j for which the duty is not fixed yet. A necessary condition to satisfy the request for day j is that

$$m'(j) \geq \sum_{q \in A} e'_q(j), \tag{24}$$

where $m'(j) := |M'(j)|$. Note that this condition is not always guaranteed even if constraints (3) (or (15)) are imposed in Step 1, since we may have $e''_q(j) > e_{qj}$ for some q. Moreover, condition (24) may not be sufficient due to infeasible sequences.

We also introduce an objective function aimed at balancing, for each employee, the number of blocks of each color performed in the corresponding assignment as well as the number of days between each block color.

For day j, we solve the following TP, with one *sink* for each employee in $M'(j)$ and one *source* for each duty to be performed. We define the following binary variables

$$z_{qi} = \begin{cases} 1 \text{ if employee } i \text{ performs duty } q \\ 0 \text{ otherwise} \end{cases} \qquad (q \in A;\ i \in M'(j)),$$

and solve:

$$\min \sum_{q \in A} \sum_{i \in M'(j)} c_{qi} z_{qi} \tag{25}$$

$$\sum_{q \in A} z_{qi} \leq 1 \quad (i \in M'(j)) \tag{26}$$

$$\sum_{i \in M'(j)} z_{qi} = e'_q(j) \quad (q \in A) \tag{27}$$

$$z_{qi} \in \{0, 1\} \quad (q \in A;\ i \in M'(j)). \tag{28}$$

The cost matrix c is defined as follows. For each $q \in A$ and $i \in M'(j)$ we let $c_{qi} := +\infty$ if employee i cannot perform duty q (because of infeasible sequences), otherwise we suitably define c_{qi} with the following objectives: (a) penalizing the assignment of a color to a block if the color is not required for some days spanned by the block, because in these days either it is not required at all or it has already been assigned to another block, (b) favoring colors that were not assigned to the employee for a long time, (c) balancing the number of days of each color assigned to the employees, and (d) penalizing "undesired" sequences of consecutive colors. In particular, for objective (a), letting h be the number of days spanned by the current block of employee i in which color q is not required, we add to c_{qi} the penalty $\beta \cdot h$, where β is a suitable parameter.

If the cost of the optimal solution to (25)–(28) is $+\infty$, our algorithm stops with a failure state. Otherwise, for each $z_{qi} = 1$ in the optimal solution, we let the color of the block performed by employee i on day j be equal to q. Note that this block does not necessarily start on day j, and that for each employee $i \in M'(j)$ such that $\sum_{q \in A} z_{qi} = 0$, the color of the associated block is not defined after the solution of this problem: it may be defined in the following days, or the block may end up being dummy.

Note that, if all blocks have the same duration and objective (a) is *dominant*, our method is a generalization of procedure GREEDY_COLOR_k presented above. This shows that, with constant workload, our approach is guaranteed to find a feasible solution.

Problem (25)–(28) is easily transformed into a classical *Assignment Problem* by introducing a *dummy duty* $n_d + 1$ with request $e'_{n_d+1} := m'(j) - \sum_{q \in A} e'_q(j)$ and replacing each duty with request $e'_q(j)$ by $e'_q(j)$ separate duties of request 1.

4.3. An iterative coloring procedure

The definition of the cost matrix c for (25)–(28) depends on the parameters used to weigh objectives (a) to (d). We use three different sets of parameters to obtain color assignments with different characteristics. If none of these is feasible, we use the following iterative method that changes the daily workload and applies Steps 1 and 2, until a feasible solution for the overall problem is found.

During the procedure, we maintain the request e_{qj} ($q = 1, \ldots, n_d$; $j \in N$) unchanged, whereas we possibly increase the value f_j ($j \in N$) considered in Step 1. Initially, we set $f_j := \sum_{q=1}^{n_d} e_{qj}$. Iteratively, we solve Step 1 and apply the heuristic of Step 2 for all sets of parameters. If a feasible coloring is found we terminate. Otherwise, for each set of parameters, we consider the set $U \subseteq N$ of days in which not all duties were covered, and find the optimal solution value of the LP relaxation of (14)–(16) with the right hand side of (15) set to f_j for $j \in N \setminus U$, and to $f_j + 1$ for $j \in U$. This LP is not solved from scratch but starting from the set of variables at the end of the last execution of Step 1. Among all sets of parameters, we consider the one for which the LP value is minimum. Letting U^* be the corresponding set of uncovered days, we set $f_j := f_j + 1$ for $j \in U^*$ and iterate.

Note that the value z_1 of the ILP solution at the end of the first execution of Step 1 gives a lower bound for the overall problem. When a feasible color assignment is found, if the number of employees, say z, is larger than z_1, we try to improve the solution by reducing the workload as follows. For each day j of the planning horizon for which $f_j > \sum_{q=1}^{n_d} e_{qj}$ and there are, say γ_j, dummy blocks spanning day j, we set $f_j := \max\{f_j - \gamma_j, \sum_{q=1}^{n_d} e_{qj}\}$. The overall iterative procedure is then applied starting from these f_js, until either a better feasible solution is found or the ILP solution value at the end of Step 1 is not smaller than z. In the former case, we try again to improve the solution as described above.

5. Computational experiments

In this section, we present the computational results obtained by the algorithms presented in Sections 3 and 4 for instances obtained from the real life case study described in Section 2. For all the instances addressed in this section, we consider a unique block duration of k days (i.e., $n_t := 1$, $k_1 := k$) and set the number of blocks $b := \lfloor \frac{38 \cdot n}{7 \cdot 8 \cdot k} \rfloor$ and the maximum number of working special days $s := \lfloor \frac{2 \cdot n}{3 \cdot 7} \rfloor$, where n is the length of the planning horizon, and S is the set of Sundays.

We first examined the original case study and additional *real life instances* obtained by considering different horizon lengths, ranging from six weeks to half a year (and all multiples of two weeks), in order to test the effect of n on the optimal solution value, as well as the capability of the methods to handle longer planning horizons. For these instances we considered rest durations of 1, 2 and 5 days, and imposed exactly 1 rest spanning 5 days (the number of 1 day and 2 days rests being determined by the consistency constraints described in Section 2). Table 4 gives the exact characteristics of the considered instances. Since the performance of the methods for the instances in our case study was the same with and without initial conditions, we report results without these conditions.

Our algorithms were implemented in C, whereas the LP and ILP solver used was CPLEX 7.0. All times given in the tables are in CPU seconds on a Digital Alpha 533MHz, whose speed is 16.1 SpecInt95. A time limit of 3600 CPU seconds was given for each instance.

We first present results for Step 1, which is the bottleneck of our solution method, even when we have to resort to the iterative coloring procedure described in Section 4.3,

Table 4. Characteristics of the instances in the case study with different values of n.

Name	n	b	n_r	d	a	s
b42	42	9	3	1, 2, 5	6, 2, 1	4
b56	56	12	3	1, 2, 5	7, 4, 1	5
b70	70	15	3	1, 2, 5	8, 6, 1	6
b84	84	19	3	1, 2, 5	14, 4, 1	8
b98	98	22	3	1, 2, 5	15, 6, 1	9
b112	112	25	3	1, 2, 5	16, 8, 1	10
b126	126	28	3	1, 2, 5	17, 10, 1	12
b140	140	31	3	1, 2, 5	18, 12, 1	13
b154	154	34	3	1, 2, 5	19, 14, 1	14
b168	168	38	3	1, 2, 5	25, 12, 1	16
b182	182	41	3	1, 2, 5	26, 14, 1	17

and then give some results on this iterative procedure. For model (14)–(16) we explicitly generated all the columns for the instances with up to 112 days. For the LP relaxation of these instances, we tried both the CPLEX LP solver and the methods mentioned in Section 3.3. The former approach was not able to solve the LP relaxation of instance b98 because of memory requirements. For the CPLEX LP solver, the *dual* algorithm obtained the best results on these instances. For method *Pricing-CG*, we proceed as follows. We start computing the reduced costs from a uniformly random column, and then proceed by considering increasing column indices (in a circular way). As soon as a negative reduced cost is found, we store the corresponding column and start from another random column (avoiding to consider the same column twice). The procedure ends when either (i) the number of columns with negative reduced cost found equals n (the number of days), or (ii) this number is positive and the number of reduced costs computed exceeds $|\mathcal{P}|/100$, or (iii) all reduced costs were computed and no column turned out to have negative reduced cost.

The dynamic column generation methods can be preceded by the application of the following heuristic (referred to as *Heur-CG* in the sequel). Fix a starting day $j \in N$ and define a block starting on day j and ending on day $j + k - 1$. Then, the choice to be performed is the type of the rest that starts on day $j + k$. To this end, for each rest type $r \in R$, the following score is defined

$$\frac{\sum_{h=j}^{j+k-1} \pi_h^* + \sum_{h=j+k+d_r}^{j+2k+d_h-1} \pi_h^*}{2k + d_r},$$

which represents the average profit of the days from the beginning of the first block (already assigned) to the end of the second block if a rest of type r is assigned (recall that π^* denotes the current dual solution). This score is halved if the second block contains a day in S. Then, choose the rest associated with the maximum score, set $j := j + k + d_r$ and iterate (of course, a rest can be assigned only if there are still rests of that type available). The procedure is applied trying all possible starting days, and storing all feasible columns with negative reduced cost, avoiding to store the same column twice. If this heuristic finds no negative reduced cost column, we apply either *DP-CG* or *ILP-CG*.

Tables 5 and 6 give the results for the solution of the LP relaxation of model (14)–(16).

Table 5 gives, for each instance, $|\mathcal{P}|$, i.e., the number of feasible patterns (if $|\mathcal{P}|$ exceeds 150 millions, "-" is reported), and T_g, i.e., the time required to generate all patterns in $\mathcal{P}$. For each method, T denotes the time required to solve the LP relaxation. In addition, for all the column generation methods described in Section 3.3, we report nit, i.e., the number of iterations, and nc, i.e., the number of columns generated. Table 6 gives analogous information for the dynamic column generation methods preceded by $Heur\text{-}CG$.

The tables show that column generation methods outperform the solution of the entire model, as may be expected. Note that both $ILP\text{-}CG$ and $ILPC\text{-}CG$, which generate one column at a time, perform a large number of iterations, with respect to $DP\text{-}CG$, producing a relatively small number of columns. On the other hand, when the dynamic column generation methods are preceded by $Heur\text{-}CG$, they tend to have the same number of iterations and columns. Computational experience showed that limiting the number of columns generated at each iteration in $DP\text{-}CG$ and $Heur\text{-}CG$ changes the value of nit and nc, but does not affect significantly the running time T. Moreover, $DP\text{-}CG$ is better than $Pricing\text{-}CG$ (even for small instances) and is speeded-up if preceded by $Heur\text{-}CG$ (as shown in Table 6), while $ILPC\text{-}CG$ is worse than $ILP\text{-}CG$. Among all methods, the best one appears to be $Heur\text{-}CG + DP\text{-}CG$. Hence, this method is used in the branch-and-bound algorithm described in the following.

Table 5. LP relaxation of model (14)–(16).

			Cplex	*Pricing-CG*			*DP-CG*			*ILP-CG*			*ILPC-CG*		
Name	$\|\mathcal{P}\|$	T_g	T	T	nit	nc	T	nit	nc	T	nit	nc	T	nit	nc
b42	978	0.20	0.35	0.11	7	206	0.11	12	136	1.63	76	78	3.80	69	71
b56	12616	0.45	13.32	0.43	8	395	0.40	16	216	3.63	88	90	8.08	86	88
b70	116990	3.91	392.82	3.20	10	634	1.00	20	283	5.23	105	108	13.00	113	116
b84	176784	6.01	1313.32	5.00	8	591	1.75	21	294	7.51	122	124	15.38	113	115
b98	2972410	143.23	–	74.14	7	592	4.28	25	361	10.05	136	139	22.13	133	136
b112	37051792	2436.85	–	–	–	–	8.65	31	453	15.31	156	158	38.33	160	162
b126	–	–	–	–	–	–	15.23	33	735	24.03	185	187	52.28	201	203
b140	–	–	–	–	–	–	28.29	41	921	33.88	217	220	72.59	225	228
b154	–	–	–	–	–	–	44.41	46	1035	46.13	265	267	95.88	260	262
b168	–	–	–	–	–	–	61.73	47	1056	48.15	251	254	107.43	263	266
b182	–	–	–	–	–	–	91.66	51	1153	60.84	274	276	124.46	288	290

Table 6. LP relaxation of model (14)–(16) when the dynamic column generation methods are preceded by $Heur\text{-}CG$.

	$Heur\text{-}CG + DP\text{-}CG$			$Heur\text{-}CG + ILP\text{-}CG$			$Heur\text{-}CG + ILPC\text{-}CG$		
Name	T	nit	nc	T	nit	nc	T	nit	nc
b42	0.10	17	156	0.40	24	141	0.53	24	141
b56	0.26	16	242	0.73	31	209	1.53	31	209
b70	0.90	25	442	1.93	41	397	3.56	42	393
b84	1.25	20	476	2.00	31	445	3.20	31	445
b98	3.13	28	710	4.51	49	670	5.10	44	634
b112	5.63	36	817	7.83	56	799	10.70	50	807
b126	10.93	42	1277	13.45	61	1189	16.43	60	1188
b140	24.41	56	1861	32.31	84	1656	34.78	89	1662
b154	29.65	50	1995	48.41	90	2024	48.08	82	1966
b168	41.00	54	2305	56.11	87	2167	63.00	87	2167
b182	61.91	52	2924	95.61	93	3131	108.55	96	3134

Table 7. Cplex vs. our Branch-and-bound for model (14)–(16).

			Cplex			Branch-and-bound		
Name	L_0	L_3	z_1	nn	T	z_1	nn	T
b42	93.33	94.50	95	0	1.18	95	9	0.17
b56	93.33	94.50	95	20	193.60	95	42	2.02
b70	93.33	94.50	95	10	2887.02	95	54	8.33
b84	88.42	90.72	92	7	3666.98	91	59	13.05
b98	89.09	91.24	–	–	–	92	66	34.33
b112	89.60	91.63	–	–	–	92	62	43.02
b126	90.00	91.94	–	–	–	92	92	299.20
b140	90.32	92.19	–	–	–	93	66	131.95
b154	90.58	92.40	–	–	–	93	74	553.12
b168	88.42	90.72	–	–	–	91	134	1098.87
b182	88.78	91.00	–	–	–	91	73	3813.65

In order to solve the instances by using model (14)–(16) with the column generation methods, we implemented our own branch-and-bound method. For the branching, we consider the fractional variable y_P in the LP solution whose value is closest to an integer $a \geq 1$. We generate three subproblems, one by setting $y_P = a$, one by setting $y_P \geq a + 1$, and one by setting $y_P \leq a - 1$, and consider the subproblems in this order in a depth-first fashion. This branching scheme tends to generate provably optimal solutions quickly, as the LP bound at the root node is typically tight.

As is often the case, branching changes the structure of the column generation problem. More specifically, all columns with negative reduced cost found either by *Heur-CG* or *DP-CG* may correspond to variables already fixed by branching. If this is the case, we resort to *ILP-CG*, adding suitable cardinality constraints to prevent the selection of these columns.

In Table 7, we provide the results of our branch-and-bound method, compared to the CPLEX ILP solver applied to the whole model. The table gives, for each instance, L_0, i.e., the trivial lower bound (1), L_3, i.e., the value of the LP relaxation at the root node, and, for the two exact methods, z_1, i.e., the value of the best integer solution found, nn, i.e., the number of nodes of the branch-decision tree, and T, i.e., the global time. The table shows that only the three smallest instances were solved to optimality within the given time limit by directly applying the CPLEX ILP solver. The branch-and-bound method was able to solve to proven optimality 10 instances out of 11, whereas for the remaining one (i.e., instance b182) it found an optimal solution within 3813.65 seconds. Note that for all instances lower bound $\lceil L_3 \rceil$ is equal to the optimal solution value z_1.

For model (2)–(12), we considered its *Original* version, the *Fixed* version obtained by fixing $z_i := 1$ for $i = 1, \ldots, \lceil L_0 \rceil$, and imposing the *precedence constraint* $z_{i+1} \leq z_i$ for all $i > L_0$, and the *Cliques* version obtained by adding the clique inequalities (13) to the Fixed version. We set the number of potential employees (i.e., $m = |M|$) to $\lceil 1.1 \cdot \lceil L_0 \rceil \rceil$. The three models were tackled by using the CPLEX ILP solver. The LP relaxation at the root node was solved by the *barrier* algorithm, that performs much better than the simplex algorithms on these LPs.

Table 8 presents the corresponding results for instances with n up to 112 (larger instances involved too large computing times). For each instance, all the versions provided the same value of the LP relaxation at the root node (denoted with L_1 in the table). For each version of the model, we also give: $T(L_1)$, i.e., the time for solving the LP relaxation at the root node, and z_1, nn and T as in Table 7. If no feasible solution was found within the time limit, we report "*" in column z_1. The table shows that L_1 is always equal to L_3 on these instances (see also Appendix B), while the corresponding times are considerably larger than those attained by the dynamic column generation methods proposed for model (14)–(16) (see Tables 5 and 6). As far as integer solutions are concerned, the Fixed and Cliques versions are better than the Original one, although much worse than the branch-and-bound algorithm based on model (14)–(16) (see Table 7).

In order to test the flexibility of the branch-and-bound algorithm based on model (14)–(16), we considered three additional classes of test instances. The first class is obtained by considering larger workloads for the original b112 case study, so as to handle instances involving a larger number of employees. The other two classes are obtained from the real life instances by changing the block and rest durations, and the workload in a random way, respectively.

We first considered *scaled instances* b112_1, ..., b112_10 obtained from instance b112 by multiplying the number e_{qj} of employees required on each day $j \in N$ and duty $q \in \{1, \dots, n_d\}$ by a factor δ, with $\delta = 1, \dots, 10$, respectively. Table 9 reports the corresponding results obtained with our branch-and-bound method. For each instance, the table reports the factor δ, L_0, L_1, L_3 (and the corresponding times $T(L_1)$ and $T(L_3)$, respectively), and z_1, nn and T as in Table 7. It is easy to show that L_0, L_1 and L_3 are proportional to the scaling factor δ. Table 9 shows that, as expected, $T(L_1)$ steeply grows with δ, while $T(L_3)$ is essentially independent of δ. The branch-and-bound method is able to solve all the instances to proven optimality within a computing time which is not increasing with δ.

We then considered instances with different block durations ($k = 4, 5, 6$, respectively), while keeping the workload equal to that of the real life case study. The values of b and s are defined as described above. The number and duration of the rests are defined as follows. The maximum rest duration is set to 5 days and at least one such rest is assigned (for $k = 6$, at least two such rests are assigned). For $k = 5$ and 6, the minimum rest duration is 2 days. We also impose that b, all entries of vector a and s are relatively prime numbers. Subject to these constraints, we choose the configuration in which the number n_r of different rest types is at least 3 and the nonzero entries of vector a (representing the number of rests of each duration) are as uniform as possible, i.e., the one that minimizes $\frac{\sum_{r \in R}(a_r - \overline{a})^2}{n_r}$, with $\overline{a} := b/n_r$. Note that for $k = 6$ and $n = 42$ no feasible configuration exists.

Table 8. Comparison of different versions of model (2)–(12).

		Original version				Fixed version				Cliques version			
Name	L_1	$T(L_1)$	z_1	nn	T	$T(L_1)$	z_1	nn	T	$T(L_1)$	z_1	nn	T
b42	94.50	21.32	*	160	3601.97	23.77	95	80	2737.85	70.45	95	0	1691.55
b56	94.50	40.17	*	30	3603.23	48.43	97	229	3604.63	173.47	103	7	3604.93
b70	94.50	76.47	*	10	3603.65	74.18	*	50	3605.82	446.70	*	0	3604.60
b84	90.72	87.58	*	0	3602.75	91.30	98	50	3605.68	930.13	*	0	3612.17
b98	91.24	154.13	*	0	3602.57	134.37	*	0	3604.63	3602.42	*	0	3602.42
b112	91.63	177.63	*	0	3603.70	177.07	*	0	3603.93	1744.57	*	0	3619.10

Table 9. Results for instances with scaled workload.

							Branch-and-bound		
Name	δ	L_0	L_1	$T(L_1)$	L_3	$T(L_3)$	z_1	nn	T
b112_1	1	89.60	91.63	177.07	91.63	5.63	92	62	43.02
b112_2	2	179.20	183.27	440.45	183.27	5.66	184	81	38.30
b112_3	3	268.80	274.91	782.38	274.91	5.91	275	101	113.38
b112_4	4	358.40	366.55	1154.20	366.55	6.10	367	100	50.40
b112_5	5	448.00	458.18	1660.58	458.18	6.06	459	98	49.02
b112_6	6	537.60	549.81	2341.17	549.81	6.01	550	292	906.08
b112_7	7	627.20	641.45	3105.40	641.45	5.71	642	102	57.71
b112_8	8	716.80	733.09	3945.47	733.09	5.71	734	107	80.44
b112_9	9	806.40	824.72	–	824.72	5.73	825	104	61.78
b112_10	10	896.00	916.36	–	916.36	6.11	917	106	99.83

Table 10. Characteristics of the instances with different block and rest durations, and different values of n.

Name	n	k	b	n_r	d	a	s
s42_4	42	4	7	3	1, 2, 5	3, 3, 1	4
s56_4	56	4	9	3	1, 2, 5	4, 3, 2	5
s70_4	70	4	11	4	1, 2, 3, 5	4, 3, 2, 2	6
s84_4	84	4	14	4	1, 2, 3, 5	6, 4, 3, 1	8
s98_4	98	4	16	3	1, 2, 5	7, 6, 3	9
s112_4	112	4	19	4	1, 2, 3, 5	8, 7, 3, 1	10
s126_4	126	4	21	4	1, 2, 3, 5	8, 7, 5, 1	12
s140_4	140	4	23	4	1, 2, 3, 5	9, 7, 5, 2	13
s154_4	154	4	26	4	1, 2, 3, 5	11, 8, 6, 1	14
s168_4	168	4	28	4	1, 2, 3, 5	10, 10, 7, 1	16
s182_4	182	4	30	4	1, 2, 3, 5	11, 10, 7, 2	17
s42_5	42	5	5	4	2, 3, 4, 5	1, 2, 1, 1	4
s56_5	56	5	7	4	2, 3, 4, 5	3, 2, 1, 1	5
s70_5	70	5	9	3	2, 3, 5	4, 4, 1	6
s84_5	84	5	11	3	2, 3, 5	6, 4, 1	8
s98_5	98	5	13	3	2, 3, 5	8, 4, 1	9
s112_5	112	5	15	3	2, 3, 5	10, 4, 1	10
s126_5	126	5	17	3	2, 3, 5	12, 4, 1	12
s140_5	140	5	19	3	2, 3, 5	14, 4, 1	13
s154_5	154	5	20	4	2, 3, 4, 5	10, 7, 2, 1	14
s168_5	168	5	22	4	2, 3, 4, 5	12, 7, 2, 1	16
s182_5	182	5	24	3	2, 3, 5	14, 8, 2	17
s56_6	56	6	6	3	2, 3, 5	2, 2, 2	5
s70_6	70	6	7	3	3, 4, 5	2, 3, 2	6
s84_6	84	6	9	3	2, 3, 5	3, 3, 3	8
s98_6	98	6	11	3	2, 3, 5	5, 4, 2	9
s112_6	112	6	12	4	2, 3, 4, 5	3, 4, 3, 2	10
s126_6	126	6	14	3	2, 3, 5	6, 5, 3	12
s140_6	140	6	15	3	2, 3, 5	5, 5, 5	13
s154_6	154	6	17	3	2, 3, 5	7, 6, 4	14
s168_6	168	6	19	3	2, 3, 5	9, 7, 3	16
s182_6	182	6	20	3	2, 3, 5	8, 7, 5	17

Table 10 gives the characteristics of the new instances, and Table 11 the results obtained with our branch-and-bound method. For each instance, Table 11 reports the values of L_0, L_1, L_3, $T(L_1)$, $T(L_3)$, z_1, nn and T as in Table 9, and the number $|\mathcal{P}|$ of feasible patterns. Table 11 shows that, also for these instances, no difference exists in the values of L_1 and L_3, although the computation of the former requires much longer.

Table 11. Results for instances with different block and rest durations, and with different values of n.

							Branch-and-bound		
Name	L_0	L_1	$T(L_1)$	$\|\mathcal{P}\|$	L_3	$T(L_3)$	z_1	nn	T
s42_4	90.00	90.72	20.50	576	90.72	0.10	91	28	8.95
s56_4	93.33	94.50	41.95	4632	94.50	0.40	95	48	3.03
s70_4	95.45	95.45	84.28	235590	95.45	1.08	96	46	7.66
s84_4	90.00	90.00	106.07	3073776	90.00	3.65	90	48	28.25
s98_4	91.87	92.84	100.68	3408930	92.84	8.30	94	1595	3618.13
s112_4	88.42	88.42	202.42	–	88.42	17.85	89	65	99.25
s126_4	90.00	90.00	215.02	–	90.00	30.28	90	61	433.66
s140_4	91.30	91.30	266.60	–	91.30	64.31	92	74	472.08
s154_4	88.84	88.84	381.43	–	88.84	94.96	89	78	657.38
s168_4	90.00	90.00	468.20	–	90.00	128.14	90	68	2984.68
s182_4	91.00	91.00	450.68	–	91.00	286.00	91	70	4523.95
s42_5	100.80	103.09	33.95	432	103.09	0.03	104	39	6.83
s56_5	96.00	97.54	71.08	1920	97.54	0.13	98	45	1.01
s70_5	93.33	95.69	103.33	2500	95.69	0.26	96	53	14.50
s84_5	91.63	93.52	132.73	10416	93.52	0.51	94	65	4.80
s98_5	90.46	92.03	183.92	24696	92.03	1.15	93	67	10.95
s112_5	89.60	90.94	254.90	50880	90.94	1.83	92	1339	3615.56
s126_5	88.94	90.11	274.33	116820	90.11	2.70	91	79	29.26
s140_5	88.42	89.46	381.23	200200	89.46	4.56	90	78	49.54
s154_5	92.40	93.96	477.20	–	93.96	8.61	94	82	337.85
s168_5	91.63	93.04	576.80	–	93.04	11.20	94	85	387.45
s182_5	91.00	92.71	597.45	–	92.71	16.58	93	82	280.29
s56_6	93.33	94.50	23.23	576	94.50	0.20	95	19	0.50
s70_6	100.00	102.16	96.65	1440	102.16	0.26	103	53	7.65
s84_6	93.33	94.50	47.30	13104	94.50	1.23	95	41	6.85
s98_6	89.09	91.24	176.77	33264	91.24	0.98	92	71	9.63
s112_6	93.33	94.50	265.70	1478848	94.50	2.88	95	69	45.70
s126_6	90.00	91.94	276.75	1065618	91.94	4.43	93	891	3600.81
s140_6	93.33	94.50	81.98	5231584	94.50	17.18	95	66	190.76
s154_6	90.58	92.40	468.68	17581080	92.40	10.75	93	85	154.05
s168_6	88.42	90.72	510.20	57500256	90.72	14.25	91	109	394.28
s182_6	91.00	92.71	632.68	–	92.71	29.73	93	105	675.46

The branch-and-bound algorithm is able to determine, within the time limit, the optimal solution of 28 out of 32 instances. For three of the four unsolved instances the solution found is within one unit from the optimum, while for instance s182_4 an optimal solution is found in 4523.95 seconds.

Finally we considered instances having the same block and rest durations as well as the same set of duties as those described in Table 4, but with a workload randomly generated for each day as follows. Initially, the workload of each day is empty. Then, for each duty $q \in \{1, \dots, n_d\}$ and each day $j \in N$, we randomly generate a number g_{qj} and impose that g_{qj} additional employees perform duty q for days $j, \dots, j+k-1$ (recalling that k is the unique duration of the blocks). These values are generated so as to guarantee that the number e_{qj} of employees that must perform duty q on day j is uniformly distributed in the ranges given in Table 12. In particular, the number f_j of employees that must work on day j ($j = 1, \dots, n$) is uniformly random in $[51, 75]$ for each day from Monday to Friday, in $[42, 63]$ for each Saturday, and in $[33, 51]$ for each Sunday. Note that the average workload from Monday to Friday and on Sunday is equal to the workload in the corresponding days of our real life case study, i.e., 63 and 42, respectively.

Table 13 gives the corresponding results, showing that these instances are easier than those having a regular workload (see Tables 6 and 7). Moreover, note that the trivial lower bound L_0 is much worse than L_3 and L_1, that coincide for these instances as well, with the exception of instance r42. The value of $|\mathcal{P}|$ is not given, since it coincides with that reported for the corresponding instances of Table 5.

In order to study the behavior of our iterative coloring procedure, we report in Tables 14, 15 and 16 the results for real life instances b42-b182 (see Table 4), scaled instances b112_1-b112_10 and random instances r42-r182, respectively. In this case, we set a time limit of 3600 CPU seconds for each execution of Step 1 and of 10000 CPU seconds for the overall iterative coloring procedure. Tables 14, 15 and 16 report, for each instance, z_1, i.e., the value of the optimal solution of the first execution of Step 1, z, i.e., the value of the final solution found by the iterative coloring procedure, nit, i.e., the number of iterations in which Steps 1 and 2 are performed, T_1, i.e., the overall computing time for Step 1, and T, i.e., the overall computing time of the method. For real life instances b42-b182 and for scaled instances b112_1-b112_10, the method always found a provably optimal solution, typically at the first iteration. As to random instances r42-r182, we solved to proven optimality 7 out of 11 instances, whereas the value of the heuristic solution found for the other 4 instances is one unit above the lower bound z_1.

As to instances with different block and rest durations (see Table 10), the iterative coloring procedure finds heuristic solution values within 4% and 3% from the lower bound z_1 for $k = 4$ and $k = 5$, respectively. For $k = 6$ the heuristic solution value is always equal to z_1. This behavior is due to the fact that block durations of 4 and 5 days are not suitable for our real life workload, which is composed of a constant term and a six day periodic term from Monday to Saturday.

Table 12. Minimum and maximum number of employees required in randomly generated instances r42–r182.

	duties 1-2-3	duties 4-5-6	f_j
Monday–Friday	[10, 14]	[7, 11]	[51, 75]
Saturday	[8, 12]	[6, 9]	[42, 63]
Sunday	[6, 9]	[5, 8]	[33, 51]

Table 13. Results for instances with the same block and rest durations as in the case study but with workload randomly generated for each day.

						Branch-and-bound		
Name	L_0	L_1	$T(L_1)$	L_3	$T(L_3)$	z_1	nn	T
r42	92.44	93.25	26.60	93.34	0.11	94	34	4.38
r56	92.42	93.52	40.15	93.52	0.38	94	44	1.50
r70	91.27	92.73	60.23	92.73	1.03	93	51	4.38
r84	87.74	90.29	76.15	90.29	1.56	91	57	8.30
r98	86.55	88.52	102.67	88.52	3.26	89	57	17.17
r112	88.28	90.19	127.25	90.19	6.33	91	67	30.23
r126	87.79	91.00	154.92	91.00	5.91	91	66	106.58
r140	89.84	91.38	235.78	91.38	43.83	92	69	191.92
r154	88.82	91.66	242.73	91.66	34.79	92	72	209.90
r168	87.29	91.00	282.87	91.00	18.35	91	69	557.93
r182	87.44	89.82	376.22	89.82	118.91	90	73	713.25

Table 14. Results for the iterative coloring procedure on real life instances.

Name	z_1	z	nit	T_1	T
b42	95	95	1	0.17	1.57
b56	95	95	1	2.02	3.12
b70	95	95	3	15.00	25.42
b84	91	91	1	13.05	14.85
b98	92	92	1	34.33	36.48
b112	92	92	1	43.02	45.50
b126	92	92	2	571.33	584.75
b140	93	93	1	131.95	135.32
b154	93	93	1	553.12	556.92
b168	91	91	1	1098.87	1102.92
b182	91	91	1	3813.65	3818.25

Table 15. Results for the iterative coloring procedure on scaled instances.

Name	z_1	z	nit	T_1	T
b112_1	92	92	1	43.02	69.22
b112_2	184	184	1	38.30	45.58
b112_3	275	275	1	113.38	120.97
b112_4	367	367	1	50.40	71.80
b112_5	459	459	1	49.02	81.55
b112_6	550	550	1	906.08	1001.63
b112_7	642	642	1	57.71	176.97
b112_8	734	734	1	80.44	238.42
b112_9	825	825	1	61.78	261.52
b112_10	917	917	1	99.83	359.60

Table 16. Results for the iterative coloring procedure on random instances.

Name	z_1	z	nit	T_1	T
r42	94	95	9	3639.47	3656.20
r56	94	94	3	38.98	46.22
r70	93	94	15	5058.13	5118.12
r84	91	91	9	63.87	110.25
r98	89	89	17	899.37	1006.40
r112	91	91	11	425.83	509.67
r126	91	91	9	4439.90	4524.42
r140	92	93	15	13603.33	13815.63
r154	92	92	12	4510.38	4687.38
r168	91	91	3	1140.48	1177.98
r182	90	91	4	9770.73	9855.42

Acknowledgements. We acknowledge the support given to this project by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) and the Consiglio Nazionale delle Ricerche (CNR), Italy. We are grateful to the anonymous referees whose tight remarks consistently improved, in our view, the paper. The computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (LabOR).

Appendix A

We consider *clique inequalities* for the ILP model (2)–(12), which are derived from the incompatibility relations among the x and w variables. We say that two binary variables are *incompatible* if they cannot both take the value 1 in a feasible solution.

Recall that $k_{\min} := \min_{t\in T} k_t$ and $d_{\min} := \min_{r\in R} d_r$. Considering a generic employee i, note that, due to constraints (6), (7) and (8), each variable x^t_{ij} $(i \in M, j \in N, t \in T)$ is incompatible with x^s_{ih} for $h \in \{j - k_s - d_{\min} + 1, \ldots, j + k_t + d_{\min} - 1\}$ and $s \in T$, as well as with w^r_{ih} for $h \in \{j - d_r + 1, \ldots, j + k_t - 1\}$ and $r \in R$. Symmetrically, each variable w^r_{ij} $(i \in M, j \in N, r \in R)$ is incompatible with x^t_{ih} for $h \in \{j - k_t + 1, \ldots, j + d_r - 1\}$ and $t \in T$, as well as with w^p_{ih} for $h \in \{j - d_p - k_{\min} + 1, \ldots, j + d_r + k_{\min} - 1\}$ and $p \in R$.

Taking into account the incompatibilities listed above, define the *incompatibility graph* $G = (V, E)$ where each vertex $v \in V$ corresponds to a binary variable x^t_{ij} or w^r_{ij} and each edge in E represents one of the incompatibilities above. Note that some additional incompatibilities between variables with respect to those mentioned above may arise, depending on the specific values of k_t and d_r $(t \in T, r \in R)$. Since the number of these incompatibilities is negligible, and taking them into account would make the structure of G much more complicated, we do not consider them in this section.

Every maximal clique of G defines a valid inequality for model (2)–(12), imposing that the sum of the variables corresponding to the vertices of the clique must not exceed z_i. We next show that G has a polynomial (and, in practice, reasonable) number of maximal cliques.

Proposition 7. *Each maximal clique of G is associated with one of inequalities (13).*

Proof. For convenience, we will call the vertices of G "variables" and identify each vertex with the name of the associated variable.

Given $h, j \in N$, we will write $h < j$ if $(j-h) \bmod n < (h-j) \bmod n$, i.e., the (cyclic) number of days between h and j is smaller than the (cyclic) number of days between j and h. We will (naturally) assume $\max_{t\in T} k_t + d_{\min} < n/2$ and $\max_{r\in R} d_r + k_{\min} < n/2$.

First of all, note that, given a (maximal) clique of G, another (maximal) clique is obtained by shifting all the day indices of the variables in the original clique by an arbitrary value. Let $s \in T$ and $p \in R$ be such that $k_s := k_{\min}$ and $d_p := d_{\min}$, respectively, and consider a maximal clique C. For every variable u, let $I(u)$ denote the set of *neighbors* of u in G, including u itself. Clearly, $C \subseteq \bigcap_{u\in C'} I(u)$ for every $C' \subseteq C$, and C is maximal if and only if $C \nsubseteq I(u)$ for every $u \notin C$.

We show that C has the structure associated with one of inequalities (13) by proving a sequence of facts.

Fact 1 *For every $t \in T$, if $x^t_{ij} \in C$ and $x^t_{ij'} \in C$ with $j' > j$, then $x^t_{ih} \in C$ for all $h = j, \ldots, j'$.*

Indeed, $C \subseteq I(x^t_{ij}) \cap I(x^t_{ij'}) \subseteq I(x^t_{ih})$ for all $h = j, \ldots, j'$. Symmetrically,

Fact 2 *For every $r \in R$, if $w^r_{ij} \in C$ and $w^r_{ij'} \in C$ with $j' > j$, then $w^r_{ih} \in C$ for all $h = j, \ldots, j'$.*

Fact 3 *If $x^s_{ij}, \ldots, x^s_{i(j+\ell)}$ are the variables in C associated with block type s, then $\ell \in \{-1, 0, \ldots, k_{\min} + d_{\min} - 1\}$ (if $\ell = -1$, no variable associated with s is in C).*

Indeed, x^s_{ij} and $x^s_{i(j+k_{\min}+d_{\min})}$ are compatible. Symmetrically (shifting the indices),

Fact 4 *If $w^p_{i(j-d_{\min}+\ell+1)}, \ldots, w^p_{i(j+k_{\min}-1)}$ are the variables in C associated with rest type p, then $\ell \in \{-1, 0, \ldots, k_{\min} + d_{\min} - 1\}$ (if $\ell = k_{\min} + d_{\min} - 1$, no variable associated with p is in C).*

We first consider the case in which $\ell \geq 0$ in Fact 3, i.e., C contains variables associated with block type s.

Fact 5 *If $x^s_{ij}, \ldots, x^s_{i(j+\ell)}$ are the variables in C associated with block type s, with $\ell \in \{0, \ldots, k_{\min} + d_{\min} - 1\}$, then the variables in C associated with block type t, $t \in T \setminus \{s\}$, are $x^t_{i(j-k_t+k_{\min})}, \ldots, x^t_{i(j+\ell)}$.*

To show that $x^t_{i(j-k_t+k_{\min})}, \ldots, x^t_{i(j+\ell)} \in C$, note that $C \subseteq \bigcap_{h=j}^{j+\ell} I(x^s_{ih}) \subseteq \bigcap_{h=j-k_t+k_{\min}}^{j+\ell} I(x^t_{ih})$. In order to complete the proof of Fact 5, we show that $x^t_{i(j-k_t+k_{\min}-1)} \notin C$ and $x^t_{i(j+\ell+1)} \notin C$. Suppose $x^t_{i(j-k_t+k_{\min}-1)} \in C$. In this case, $C \subseteq I(x^t_{i(j-k_t+k_{\min}-1)}) \cap I(x^t_{i(j+\ell)})$. Noting that $I(x^t_{i(j-k_t+k_{\min}-1)}) \cap I(x^t_{i(j+\ell)}) \subseteq I(x^s_{i(j-1)})$ and $x^s_{i(j-1)} \notin C$ contradicts the maximality of C analogously, supposing $x^t_{i(j+\ell+1)} \in C$, the contradiction follows from $C \subseteq I(x^t_{i(j-k_t+k_{\min})}) \cap I(x^t_{i(j+\ell+1)}) \subseteq I(x^s_{i(j+\ell+1)})$ and $x^s_{i(j+\ell+1)} \notin C$.

Fact 6 *If $x^s_{ij}, \ldots, x^s_{i(j+\ell)}$ are the variables in C associated with block type s, with $\ell \in \{0, \ldots, k_{\min} + d_{\min} - 1\}$, then the variables in C associated with rest type r, $r \in R$, are $w^r_{i(j-d_r+\ell+1)}, \ldots, w^r_{i(j+k_{\min}-1)}$.*

Indeed, $w^r_{i(j-d_r+\ell)}$ is compatible with $x^s_{i(j+\ell)}$, and $w^r_{i(j+k_{\min})}$ is compatible with x^s_{ij}. This shows that the variables associated with rest type r in C are contained in $\{w^r_{i(j-d_r+\ell+1)}, \ldots, w^r_{i(j+k_{\min}-1)}\}$. For the case $d_r = d_{\min}$ and $\ell = k_{\min} + d_{\min} - 1$, we have $j - d_r + \ell + 1 > j + k_{\min} - 1$ and the proof is complete. Otherwise, we have to show that $w^r_{i(j-d_r+\ell+1)} \in C$ and $w^r_{i(j+k_{\min}-1)} \in C$. This holds since all the resulting w variables in C are adjacent in G, and the x variables in C are $x^t_{i(j-k_t+k_{\min})}, \ldots, x^t_{i(j+\ell)}$, $t \in T$, which are all contained in $I(w^r_{i(j-d_r+\ell+1)}) \cap I(w^r_{i(j+k_{\min}-1)})$.

Facts 5 and 6 yield the proof of the proposition for the case $\ell \geq 0$. The remaining case $\ell = -1$ is symmetric to the case $\ell = k_{\min} + d_{\min} - 1$, by exchanging the role of blocks and rests. In particular, the symmetric counterparts of Facts 5 and 6 read:

Fact 7 *If $w^p_{i(j-d_{\min}+\ell+1)}, \ldots, w^p_{i(j+k_{\min}-1)}$ are the variables in C associated with rest type p, with $\ell \in \{-1, 0, \ldots, k_{\min} + d_{\min} - 2\}$, then the variables in C associated with rest type r, $r \in R \setminus \{p\}$, are $w^r_{i(j-d_r+\ell+1)}, \ldots, w^r_{i(j+k_{\min}-1)}$.*

Fact 8 *If $w^p_{i(j-d_{\min}+\ell+1)}, \ldots, w^p_{i(j+k_{\min}-1)}$ are the variables in C associated with rest type p, with $\ell \in \{-1, 0, \ldots, k_{\min} + d_{\min} - 2\}$, then the variables in C associated with block type t, $t \in T$, are $x^t_{i(j-k_t+k_{\min})}, \ldots, x^t_{i(j+\ell)}$.*

The number of cliques (13) is $n \cdot m$ $(k_{\min} + d_{\min} + 1)$, i.e., comparable with the number of constraints in model (2)–(12).

Appendix B

In order to explain why L_3 and L_1 are typically equal, we first describe an apparently strange variation of (14)–(16) whose LP relaxation is equivalent to that of (2)–(12). Let $\mathcal{Q}$ be the set of all (not necessarily feasible) patterns Q corresponding to an alternating sequence of blocks and rests that span $k(Q) \cdot n$ days for some positive integer $k(Q)$, and such that each block and rest has a duration among those specified on input (as usual, a block or a rest may be split between the first and the last days of the planning horizon). For each $Q \in \mathcal{Q}$, let $b(Q,t)$ $(t \in T)$ and $a(Q,r)$ $(r \in R)$ denote, respectively, the number of blocks of type t and rests of type r in pattern Q. Moreover, let $s(Q)$ be the number of working days j in pattern Q such that $j \bmod n$ is a special day in S. Finally, let $n(Q,j)$ $(j \in N)$ be the number of working days among $j, n+j, \ldots, (k(Q)-1)n+j$ in the pattern (i.e., the working days that are equal to $j \bmod n$). Consider the ILP

$$\min \sum_{Q \in \mathcal{Q}} k(Q)\, y_Q \tag{29}$$

$$\sum_{Q \in \mathcal{Q}} n(Q,j)\, y_Q \geq f_j \quad (j \in N) \tag{30}$$

$$\sum_{Q \in \mathcal{Q}} b(Q,t)\, y_Q = b_t \sum_{Q \in \mathcal{Q}} k(Q)\, y_Q \quad (t \in T) \tag{31}$$

$$\sum_{Q \in \mathcal{Q}} a(Q,r)\, y_Q = a_r \sum_{Q \in \mathcal{Q}} k(Q)\, y_Q \quad (r \in R) \tag{32}$$

$$\sum_{Q \in \mathcal{Q}} s(Q)\, y_Q \leq s \sum_{Q \in \mathcal{Q}} k(Q)\, y_Q \tag{33}$$

$$y_Q \geq 0, \text{integer} \quad (Q \in \mathcal{Q}). \tag{34}$$

Constraints (31)–(33) impose, respectively, that, on average, b_t blocks of type t, a_r rests of type r, and at most s working special days are given for each pattern of the solution spanning n days. Correspondingly, this ILP is a relaxation of both (14)–(16) and (2)–(12). On the other hand, we have

Proposition 8. *The LP relaxations of (29)–(34) and (2)–(12) are equivalent.*

Proof. Consider a solution $\overline{z}, \overline{x}, \overline{w}$ of the LP relaxation of (2)–(12). We will construct a solution $\overline{y}$ of the LP relaxation of (29)–(34) having the same value. Initially, set $\overline{y}_Q := 0$ for $Q \in \mathcal{Q}$.

Now concentrate on the variables associated with an employee $i \in M$. Construct the corresponding directed multigraph with one vertex for each day $j \in N$ and a *block arc* $(j, j+k_t)$ of capacity $\overline{x}^t_{ij}$ for each $\overline{x}^t_{ij} > 0$ as well as a *rest arc* $(j, j+d_r)$ of capacity $\overline{w}^r_{ij}$ for each $\overline{w}^r_{ij} > 0$. Due to constraints (7) and (8), for each vertex j, the overall capacity of the block arcs entering j is equal to the overall capacity of the rest arcs leaving j (and viceversa). Consider an arbitrary cycle that contains alternating block and rest arcs, and let c be the minimum capacity of an arc in the cycle. It is easy to see that this cycle corresponds to a pattern $Q \in \mathcal{Q}$. Increase the value of $\overline{y}_Q$ by c, decrease the capacity of the arcs in the cycle by c (removing the arcs of capacity zero), and iterate the procedure until no arc is remaining.

It is easy to check that the processing of the cycles for employee i increases $\sum_{Q \in \mathcal{Q}} k(Q)\, \overline{y}_Q$ by $\overline{z}_i$ (thanks to constraints (6)) and guarantees that constraints (31)–(33) are satisfied at the end of the processing. Hence, once this processing has been done for all employees, the final solution $\overline{y}$ has value $\sum_{i \in M} \overline{z}_i$ and is feasible for the LP relaxation of (29)–(34). In particular, $\overline{y}$ satisfies (30) thanks to constraints (3).

Conversely, given a feasible solution $\overline{y}$ of the LP relaxation of (29)–(34), it is easy to define a solution $\overline{z}, \overline{x}, \overline{w}$ of the LP relaxation of (2)–(12) of the same value. In particular, for each employee $i \in M$, one may define $\overline{z}_i := \frac{\sum_{Q \in \mathcal{Q}} k(Q)\, \overline{y}_Q}{m}$ and then define the variables $\overline{x}^t_{ij}, \overline{w}^t_{ij}$ so that they are associated with patterns $Q \in \mathcal{Q}$, each taken with value $\frac{\overline{y}_Q}{m}$ (further details are omitted).

The results in Section 5 suggest that, in practice, allowing the patterns to have the strange structure as those in $\mathcal{Q}$ and imposing the "global" constraints (31)–(33), does not change the LP value with respect to the case in which one pretends that each pattern is feasible.

We conclude by discussing a nontrivial example for which $\lceil L_3 \rceil > \lceil L_1 \rceil$. Recall the polynomial reduction in the proof of Proposition 1. It is easy to see that a perfectly analogous reduction works for the *Four Partitioning Problem* (4PP) (see [18] for a precise statement of this problem). Consider a 4PP instance with no feasible solution such that the LP relaxation of the associated "Set Partitioning" formulation has a feasible solution, say $\overline{\gamma}$. (This formulation has a binary variable for each four-tuple of items with sum exactly c and one equality constraint for each item.) Such an instance is described, e.g., in [23]. For the staff scheduling instance obtained by the reduction, it is easy to verify that $L_3 > 1$, whereas it is possible to show how to construct from $\overline{\gamma}$ a solution of the LP relaxation of (2)–(12) of value 1, implying $L_1 = 1$. The difficulty in constructing such a "bad" 4PP instance and the fact that apparently it must contain very large integer values may reflect the fact that for real life instances L_3 and L_1 coincide in almost all cases.

References

1. Aykin, T.: Optimal shift scheduling with multiple break windows. Management Science **42**, 591–602 (1996)
2. Aykin, T.: A comparative evaluation of modeling approaches to the labor shift scheduling problem. European Journal of Operational Research **125**, 381–397 (2000)
3. Balakrishnan, N., Wong, R.T.: A network model for the rotating workforce scheduling problem. Networks **20**, 25–42 (1990)
4. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Vance, P.H.: Crew scheduling. In R.W. Hall, editor, Handbooks of Transportation Science, pages 493–521, Norwell, MA, 1999. Kluwer Academic Publisher
5. Bartholdi III, J.J.: A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering. Operations Research **29**, 501–510 (1981)
6. Bartholdi III, J.J., Orlin, J.B., Ratliff, H.D.: Cyclic scheduling via integer programs with circular ones. Operations Research **28**, 1074–1085 (1980)
7. Beaumont, N.: Scheduling staff using mixed integer programming. European European Journal of Operational Research **98**, 473–484 (1997)
8. Bechtold, S.E., Jacobs, L.W.: The equivalence of general set-covering and implicit integer programming formulations for shift scheduling. Naval Research Logistics **43**, 233–250 (1996)
9. Billionnet, A.: Integer programming to schedule a hierarchical workforce with variable demands. European Journal of Operational Research **114**, 105–114 (1999)

10. Brusco, M.J., Jacobs, L.W.: Personnel tour scheduling when starting-time restrictions are present. Management Science **44**, 534–547 (1998)
11. Brusco, M.J., Jacobs, L.W.: Optimal models for meal-break and start-time flexibility in continuous tour scheduling. Management Science **46**, 1630–1641 (2000)
12. Brusco, M.J., Jacobs, L.W., Bongiorno, R.J., Lynos, D.V., Tang, B.: Improving personnel scheduling at airline stations. Operations Research **43**, 741–751 (1995)
13. Caprara, A., Fischetti, M., Toth, P., Vigo, D., Guida, P.L.: Algorithms for railway crew management. Mathematical Programming **79**, 125–141 (1997)
14. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. John Wiley & Sons, Inc, 1998
15. Cormen, T.H., Leiserson, C.E., Rivest, R.R.: Introduction to Algorithms. MIT Press, 1990
16. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: a review of applications, methods and models. European Journal of Operational Research. (to appear)
17. Gamache, M., Soumis, F., Marquis, G., Desrosiers, J.: A column generation approach for large-scale aircrew rostering problems. Operations Research **47**, 247–263 (1999)
18. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman, San Francisco, 1979
19. Geoffrion, A.M.: Lagrangean relaxation for integer programming. Mathematical Programming Study **2**, 82–114 (1974)
20. Jacobs, L.W., Brusco, M.J.: Overlapping start-time bands in implicit tour scheduling. Management Science **42**, 1247–1259 (1996)
21. Jarrah, A.I.Z., Bard, J.F., deSilva, A.: Solving large-scale tour scheduling problems. Management Science **40**, 1124–1144 (1994)
22. Jaumard, B., Semet, F., Vovor, T.: A generalized linear programming model for nurse scheduling. European Journal of Operational Research **107**, 1–18 (1998)
23. Marcotte, O.: An instance of the cutting stock problem for which the rounding property does not hold. Operations Research Letters **4**, 239–243 (1986)
24. Mehrotra, A., Murphy, K.E., Trick, M.A.: Optimal shift scheduling: a branch-and-price approach. Naval Research Logistics **47**, 185–200 (2000)
25. Thompson, G.M.: Improved implicit optimal modeling of the labor shift scheduling problem. Management Science **41**, 595–607 (1995)
26. Thompson, G.M.: Labor staffing and scheduling models for controlling service levels. Naval Research Logistics **44**, 719–740 (1997)
27. Tien, J.M., Kamiyama, A.: On manpower scheduling algorithms. SIAM Review **24**, 275–287 (1982)