Integer programming techniques for the nurse rostering problem — **Source link**

Haroldo Gambini Santos, Túlio Ângelo Machado Toffolo, Túlio Ângelo Machado Toffolo, Rafael Antonio Marques Gomes
...+1 more authors

Institutions: Universidade Federal de Ouro Preto, Katholieke Universiteit Leuven, Universidade Federal de Minas Gerais

Related papers:

- The State of the Art of Nurse Rostering

- Personnel scheduling: A literature review

- The first international nurse rostering competition 2010

- New approaches to nurse rostering benchmark instances

- A systematic two phase approach for the nurse rostering problem

Integer Programming Techniques for the Nurse Rostering Problem

**Haroldo G. Santos · Túlio A. M. Toffolo ·
Rafael A. M. Gomes · Sabir Ribas**

Abstract This work presents Integer Programming (IP) techniques to tackle the problem of the International Nurse Rostering Competition. Starting from a compact and monolithic formulation in which the current generation of solvers performs poorly, improved cut generation strategies and primal heuristics are proposed and evaluated. A large number of computational experiments with these techniques produced the following results: the optimality of the vast majority of instances was proved, the best known solutions were improved by up to 15% and strong dual bounds were obtained. In the spirit of reproducible science, all code was implemented using the Computational Infrastructure for Operations Research (COIN-OR).

1 Introduction

A significant amount of research has been devoted to the computational solution of the Nurse Rostering Problem [15]. Much of previous work, however, concentrates on specific case studies, focusing on the particularities of certain institutions. For these works, comparison between different search strategies is a very difficult task. The International Nurse Rostering Competition (INRC) [28] was organized to stimulate the research in this area. An instance set was proposed and a significant number of different algorithms has been empirically

Haroldo G. Santos[1] · Túlio A. M. Toffolo[1,2] · Rafael A. M. Gomes[1] · Sabir Ribas[3]
[1] Computing Department, Federal University of Ouro Preto, Brazil
[2] Department of Computer Science, KU Leuven, Belgium
[3] Department of Computer Science, Federal University of Minas Gerais, Brazil
E-mail: haroldo@iceb.ufop.br, rafael.amgomes@gmail.com, tulio@toffolo.com.br and sabirribas@gmail.com

evaluated using it. As a result, best known solutions have been updated since then.

In this work we present a monolithic compact Integer Programming formulation, i.e. a formulation with a polynomial number of constraints and variables, for the problem described in INRC. We propose and evaluate techniques for improving the performance of state-of-art integer programming (IP) solvers using this formulation.

The proposed techniques can be divided in two groups: the first group is devoted to the improvement of dual bounds. In this case we are not only interested in the quick production of high quality solutions but also interested in having a precise estimate for a lower bound on the optimal solution cost. This obviously incurs additional processing time but it is a critical step for methods that aim to prove the optimality. In the second group we present techniques to speed up the production of near optimal feasible solutions. These latter techniques can be applied alone for those interested in solving real world situations.

Our computational experiments showed that fast methods for the production of good primal and dual bounds can be obtained using the proposed techniques: our algorithms provided not only very competitive heuristics but we also proved the optimality for the vast majority of INRC instances and improved best known solutions for the remaining instances by up to 15%.

In the spirit of the reproducible science, the implementation of the cut generation procedures was made using the open source branch-and-cut software[24] of the COIN-OR Foundation, CBC[33]. We proposed alternative cut separation routines for two of the cut generators included on CBC and showed that these routines significantly outperform the included ones considering the required time to produce better lower bounds for INRC instances. These routines are being made available also as an open source project.

The paper is organized as follows: in section 2 an informal description of the problem is presented along with a brief description of previous algorithms proposed in the literature. In section 3 the NRP problem is formally stated using our proposed formulation. Sections 4 and 5 present our proposals for dual and primal bound improvements, respectively. Section 6 includes computational experiments to evaluate our proposals. Finally, in section 7, conclusions and future works are discussed.

2 The Nurse Rostering Problem

The nurse rostering problem can be described by a nurse-day view, a nurse-task view, or a nurse-shift pattern view [16]. In the nurse-day view, allocations are indexed for each nurse and each day. This way, a solution can be directly represented by a matrix where each cell $m_{i,j}$ contains a set of shifts to be performed by the nurse i in the day j. Broadly speaking this set may have any number of shifts, but it the INRC problem and most practical cases a nurse performs only one shift per day – which may include morning shift (M),

evening shift (E), night shift (N), day-off (-), among others. Table 1 presents part of a weekly roster which indicates the shifts allocated to the nurses, in a nurse-day view.

Table 1 Example of an NRP solution in a nurse-day view

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
N1	M	M	N	-	-	E	E
N2	E	N	E	E	M	-	-
N3	-	E	M	-	M	-	N

In the nurse-task view, the decision variable is indexed for each nurse and each task that the nurse performs in the scheduling period. This decision variable may assume a value of 1 if the nurse is assigned to the task, or 0 otherwise. In the nurse-shift pattern view, the decision variable is indexed for each nurse and each pattern of shifts available. Cheang et al. [16] presents a bibliographic survey of the many models and methodologies available to solve the NRP.

In this work, we address the problem defined in the first International Nurse Rostering Competition, sponsored by the leading conference in the Automated Timetabling domain, PATAT. Competitors were allowed to submit a specific technique for each instance type. There follows a brief description of the approaches that succeeded in the competition.

Valouxis et al. [43], winners of the challenge, developed a two phase strategy where in the first phase the workload for each nurse and for each day of the week was decided while in the second phase the specific daily shifts were assigned. Since the competition imposed quality and time constraint requirements, they partitioned the problem instances into sub-problems of manageable computational size which were then solved sequentially using Integer Programming. Also, they applied local optimization techniques for searching across combinations of nurses' partial schedules. This sequence was repeated several times depending on the computational time available.

Burke and Curtois [12] applied an ejection chain based method for sprint instances and a branch and price algorithm for medium and long instances. Problem instances have been converted to the general staff rostering model proposed and documented by the same team. Then, their software Roster Booster which included the above mentioned algorithmic approaches was used.

Bilgin et al. [6] applied a hyper-heuristic approach combined with a greedy shuffle heuristic. The hyper-heuristic consisted of a heuristic selection method and a move acceptance criterion. The best solution found was further improved by exploring swaps of partial rosters between nurses.

Nonobe [37] modified the general purpose constraint optimization solver presented in [38], which is based on a tabu search algorithm. Their main idea is to spend less time in developing algorithms since after reformulating the problem as a constraint optimization problem only user defined constraints have to be implemented.

More details about these approaches can be found on the site of the competition[1].

Another recent work developed by Burke et al. [13] is a hybrid multi-objective model that combines IP and variable neighborhood search (VNS) to deal with NRP. An IP formulation is first used to solve the subproblem which includes the full set of hard constraints and a subset of soft constrains. Next, a basic VNS follows as a post-processing procedure to further improve the IP's resulting solutions. The major focus of the VNS is the satisfaction of the excluded constraints from the preceding IP model.

2.1 Constraints

Combinatorial optimization problems generally carry hard and soft constraints. Roughly, the difference is that hard constraints must be met and soft constraint violations should be avoided. A single violation of a hard constraint renders the solution infeasible. In this work, we considered the problem defined in the INRC, which has the following hard constraints (I-II):

I) a nurse cannot work more than one shift per day;
II) all shift type demands during the planning period must be met.

   Constraints 1-13 are the soft constraints:

1) minimum/maximum number of shifts assigned to a nurse;
2) minimum/maximum number of consecutive free days;
3) minimum/maximum number of consecutive working days;
4) maximum number of working weekends in four weeks;
5) minimum/maximum number of consecutive working weekends;
6) number of days off after a series of night shifts;
7) complete weekends: if a nurse has to work only on some days of the weekend then penalty occurs;
8) no night shift before free weekend: if a nurse does not work on the weekend, then Friday night should also be free, otherwise penalty occurs;
9) identical shift types during the weekend: assignments of different shift types to the same nurse during a weekend are penalized;
10) alternative skill: if an assignment of a nurse to a shift type requiring a skill that she does not have occurs, then the solution is penalized accordingly;
11) unwanted patterns: an unwanted pattern is a sequence of assignments that is not in the preferences of a nurse based on her contract;
12) day on/off request: requests by nurses to work or not to work on specific days of the week should be respected, otherwise solution quality is compromised;

---

[1] https://www.kuleuven-kulak.be/nrpcompetition/competitor-ranking

13) shift on/off request: similar to the previous but now for specific shifts on certain days;

In the next section our compact Integer Programming formulation for the INRC problem is presented.

3 An Integer Programming Formulation for the INCR Problem

In this section we present an Integer Programming formulation which successfully models all constraints, explicitly or implicitly, considered in instances of the International Nurse Rostering Competition. In the INRC model, nurses are hired in different contracts. Even though much of the input data is contract related in this formulation we present most of the sets and parameters in a more generic organization, as nurse related, in order to simplify notation.

3.1 Input Data

N: set of nurses;

S: set of shifts, with $\tilde{S} \subset S$ being the set of night shifts and $\hat{S} \subset S$ the set of day shifts;

D: set of days with elements sequentially numbered from 1;

$\tilde{r}_{ds}$: number of required nurses at day d and shift s;

Π: set of all ordered pairs $(d_1, d_2) \in D \times D : d_1 \leq d_2$ representing windows in the planning horizon;

$\tilde{W}_n$: set of weekends in the planning horizon according to the weekend definition for nurse n, with elements numbered from 1 to $\tilde{w}_n$;

$\tilde{D}_{in}$: set of days in the i-th weekend of nurse n;

l_{in}: last day before the i-th weekend of nurse n;

$\acute{P}_n$: set of unwanted working shift patterns for nurse n;

$\hat{P}_n$: set of unwanted working days patterns for nurse n;

$\underline{\gamma}_n^i, \overline{\gamma}_n^i$: lower and upper limit for valid values for ranged soft constraint i for nurse n;

$\underline{\omega}_n^i, \overline{\omega}_n^i$: weight for violating the lower and upper limit of soft constraint i for nurse n;

ω_n^i: weight for violating the i-th logical constraint for nurse n;

$\sigma_{nd_1d_2}$: pre-computed penalty for the continuous work of nurse n in days $d_1, \ldots, d_2$ considering soft constraints 3 and 7;

$\tau_{nd_1d_2}$: pre-computed penalty for the continuous rest of nurse n in days $d_1, \ldots, d_2$ considering soft constraint 2;

$\psi_{ni_1i_2}$: pre-computed penalty for the continuous work of nurse n in weekends $i_1, \ldots, i_2$;

ν_{nsd}: pre-computed penalty for the work of nurse n at shift s on day d considering soft constraints 10 and 13.

We divide soft constraints in two groups. In the first group, denoted here by *Ranged Soft Constraints,* we include constraints which state a range of

valid integer values for a variable in the format $\underline{v} \leq v \leq \overline{v}$. Values outside this range need to be penalized through slack variables (or through coefficients in decision variables) according to their distance to the closest valid value.

	Ranged Soft Constraints
1	minimum/maximum number of shifts assigned to a nurse
2	minimum/maximum number of consecutive free days
3	minimum/maximum number of consecutive working days
4	maximum number of working weekends in four weeks
5	minimum/maximum number of consecutive working weekends
6	number of days off after a series of night shifts

	Logical Soft Constraints
7	complete weekends
8	no night shift before free weekend
9	identical shift types during the weekend
10	alternative skill
11	unwanted patterns
12	day on/off request
13	shift on/off request

Table 2 Indexes for ranged and logical soft constraints

In the second group, the *Logical Soft Constraints*, are those constraints which are satisfied or not, i.e. the maximum distance to feasibility is one.

In Table 2 each soft constraint is associated with an index, as defined in section 2.1. This index will be used to express constants which state the minimum and maximum limit for a given ranged soft constraint i and a nurse n, which will be denoted here by $\underline{\gamma}_n^i$ and $\overline{\gamma}_n^i$, respectively. The weight for violating the i-th minimum and maximum limit of these constraints is denoted by $\underline{\omega}_n^i$ and $\overline{\omega}_n^i$, respectively. Finally, for logical soft constraints the weight of penalizing them is defined by ω_n^i.

Some specific sequences of working shifts (soft constraint 11) may be unwanted, e.g.: Late, Evening, Late. The set of these patterns for nurse n is specified in $\acute{P}_n$ and each pattern $\acute{p} \in \acute{P}_n$ has a size $\tilde{s}(\acute{p})$ and contents $\acute{p}[1], \ldots, \acute{p}[\tilde{s}(\acute{p})] \in S$. Day related patterns are also considered in soft constraint 12: sequences of working/resting days should be avoided, e.g.: not working on Friday and working on the succeeding weekend. The set of these patterns is defined by $\hat{P}_n$ with elements $\hat{p} \in \hat{P}_n$ with size $\tilde{s}(\hat{p})$. To specify which days from the pattern represent the "not working" option we define a set of virtual days $\dot{D}$ with negative numbered days representing this option, so that pattern elements $\hat{p}[1], \ldots, \hat{p}[\tilde{s}(\hat{p})]$ are restricted to be in $D \cup \dot{D}$.

3.2 Decision variables

The main decision variables are the three indexed x_{nsd} binary variables:

$$x_{nsd} = \begin{cases} 1 & \text{if nurse } n \text{ is allocated to shift } s \text{ and day } d \\ 0 & \text{otherwise} \end{cases}$$

additionally, there are the following auxiliary variables:

$$y_{ni} = \begin{cases} 1 & \text{if nurse } n \text{ works at weekend } i \\ 0 & \text{otherwise} \end{cases}$$

$$w_{nd_1 d_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from day } d_1 \text{ until day } d_2 \\ 0 & \text{otherwise} \end{cases}$$

$$r_{nd_1 d_2} = \begin{cases} 1 & \text{if nurse } n \text{ rests from day } d_1 \text{ until day } d_2 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ni_1 i_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from weekend } i_1 \text{ until weekend } i_2 \\ 0 & \text{otherwise} \end{cases}$$

and the following slack variables:

$\underline{\alpha}_i^n \in \mathbb{Z}^+$, $\overline{\alpha}_i^n \in \mathbb{Z}^+$ slack variables for measuring violations in the lower and upper limit of soft constraint i for nurse n, additional indexes may be used in specific soft constraints;

$\alpha_i^n \in \{0,1\}$ binary variables indicating if logical soft constraint i is violated for nurse n ($\alpha_i^n = 1$) or not ($\alpha_i^n = 0$), additional indexes may be used in specific soft constrains.

To simplify the statement of constraints we consider additional variables y_{n0}, which are always fixed to zero.

There are $O(n \times d^2)$ variables w and r, and so the number of variables of the model is more affected by the extension of the planing horizon than by the number of nurses. In practice, the number of variables is not expected to increase considerably since the planning horizon is usually one month in most hospitals. The same holds for variables z but, instead of the number of days, the number of weekends is considered.

3.3 Objective Function

Before presenting the objective function we remark that some slack variables (and their respective constraints) do not need to be explicitly included. This is the case of constraints which are directly linked to the selection of a specific working/resting window from the set Π by activating variables $w_{nd_1d_2}$ and $r_{nd_1d_2}$, respectively. This is obviously the case for soft constraints 2 and 3 (Table 2) and also the case for soft constraint 7, since every activation of $w_{nd_1d_2}$ finishing/starting in the middle of a weekend must be penalized. We denote by $\sigma_{nd_1d_2}$ and $\tau_{nd_1d_2}$ the weighted penalty of all violations incurred from working (resting) continuously in a block starting at day d_1 and finishing at day d_2 for nurse n, respectively. Soft constraints 10 and 13 are also directly penalized in x_{nsd} variables with coefficients ν_{nsd}. To make it clear how constraints 13 can be implicitly evaluated just by the activation of a x_{nsd} variable we recall that in INRC instances shifts and skills are directly related. Analogously, violations of soft constraints 5 are penalized in variables $z_{ni_1i_2}$ with coefficients $\psi_{ni_1i_2}$. Given this notation the objective function is formulated as follows:

Minimize:

$$
\sum_{n \in N}
\begin{bmatrix}
\displaystyle\sum_{(d_1d_2)\in\Pi} (\sigma_{nd_1d_2} w_{nd_1d_2} + \tau_{nd_1d_2} r_{nd_1d_2}) + \\[2em]
\displaystyle\sum_{s\in S}\sum_{d\in D} \nu_{nsd} x_{nsd} + \overline{\omega}_n^1 \overline{\alpha}_n^1 + \underline{\omega}_n^1 \underline{\alpha}_n^1 + \\[2em]
\displaystyle\sum_{i\in\{1\dots\tilde{w}_c\}} (\overline{\omega}_n^4 \overline{\alpha}_{ni}^4 + \omega_n^8 \alpha_{ni}^8 + \\[2em]
\displaystyle\sum_{d_1\in\tilde{D}_{in}}\sum_{d_2\in\tilde{D}_{in}:d_2>d_1} \omega_n^9 \alpha_{nd_1d_2}^9) + \\[2em]
\displaystyle\sum_{i_1,i_2\in\tilde{W}_n:i_2\geq i_1} \psi_{ni_1i_2} z_{ni_1i_2} + \sum_{d\in D} \underline{\omega}_n^6 \underline{\alpha}_n^6 + \\[2em]
\displaystyle\sum_{\acute{p}\in\acute{P}_n}\sum_{d\in\{1,\dots,|D|-\tilde{s}(\acute{p})+1\}} \omega_n^{11} \alpha_{n\acute{p}d}^{11} + \\[2em]
\displaystyle\sum_{\hat{p}\in\hat{P}_n} \omega_n^{12} \alpha_{n\hat{p}}^{12}
\end{bmatrix}
$$

3.4 Constraints

Constraints are presented next. Constraints 1 and 2 model the two hard constraints of the INRC problem : to provide sufficient coverage of nurses for every day and shift and to limit working shifts for nurses to a maximum of one per day. Constraints 3 and 4 link the activation of variables x with the activation of y variables. Constraints from 5 to 9 ensure that every working window activation (w variables) is immediately followed by the activation of a r variable with the corresponding resting window and vice versa. This implies the selection of contiguous working and resting periods of different sizes for the whole planning horizon.

The remaining constraints are all soft-constraints, which means that they can be violated since they include a slack variable (variables α) which will be penalized in the objective function when activated. Ranged constraints 10 model the minimum and maximum working days in the planning horizon. Constraints 11 limit the maximum number of working weekends in four weeks. Constraints 12 impose a minimum number of resting days after a sequence of night shifts. Constraints 13 ensure that a nurse is not allocated to a night shift in a day preceding a free weekend. For a weekend, allocated shifts should be equal for every working day, as stated in constraints 14 and 15. Undesired patterns for shifts and days and are modeled in constraints 16 and 17, respectively. Constraints 18 ensure that at most one pattern expressed in z variables covers one weekend. Constraints 19 forbid the occurrence of two consecutive patterns expressed in z variables, since these two should be combined into a larger one. Finally, constrains 20 link variables y and z. We recall that many soft constraints are directly penalized in coefficients of decision variables and do not need to be explicitly considered here, as stated before.

$$\sum_{n \in N} x_{nsd} = \tilde{r}_{ds} \ \forall d \in D, s \in S \ (1)$$

$$\sum_{s \in S} x_{nsd} \leq 1 \ \forall n \in N, d \in D \ (2)$$

$$y_{ni} \geq \sum_{s \in S} x_{nsd} \ \forall n \in N, i \in \tilde{W}_n, d \in \tilde{D}_{in} \ (3)$$

$$y_{ni} \leq \sum_{s \in S, d \in \tilde{D}_{in}} x_{nsd} \ \forall n \in N, i \in \tilde{W}_n \ (4)$$

$$\sum_{s \in S} x_{nsd} = \sum_{(d_1,d_2) \in \Pi \ : \ d \in \{d_1,\dots,d_2\}} w_{nd_1 d_2} \ \forall n \in N, d \in D \ (5)$$

$$\sum_{s \in S} x_{nsd} = 1 - (\sum_{(d_1,d_2) \in \Pi \ : \ d \in \{d_1,\dots,d_2\}} r_{nd_1 d_2}) \ \forall n \in N, d \in D \ (6)$$

$$\sum_{(d_1,d_2) \in \Pi \ : \ d \in \{d_1 \dots d_2\}} (w_{nd_1 d_2} + r_{nd_1 d_2}) = 1 \ \forall n \in N, d \in D \ (7)$$

$$\sum_{d' \in \{1,\dots,d\}} w_{nd'd} + \sum_{d'' \in D : d'' \geq d+1} w_{n,d+1,d''} \leq 1 \ \forall n \in N, d \in D \ (8)$$

$$\sum_{d' \in \{1,\dots,d\}} r_{nd'd} + \sum_{d'' \in D : d'' \geq d+1} r_{n,d+1,d''} \leq 1 \ \forall n \in N, d \in D \ (9)$$

$$\underline{\gamma}_n^1 - \underline{\alpha}_n^1 \leq \sum_{s \in S, d \in D} x_{nsd} \leq \overline{\gamma}_n^1 + \overline{\alpha}_n^1 \ \forall n \in N (10)$$

$$\sum_{i' \in \{i,\dots,i+3\}} y_{ni'} \leq \overline{\gamma}_n^4 + \overline{\alpha}_{ni}^4 \ \forall n \in N, i \in \{1,\dots,\tilde{w}_n - 3\} (11)$$

$$\sum_{s' \in \tilde{S}} \underline{\gamma}_n^6 x_{ns'd} + \sum_{s \in \hat{S}, d' \in \{d+1,\dots,d+\underline{\gamma}_n^6\}} x_{nsd'}$$

$$\leq \underline{\gamma}_n^6 + \underline{\alpha}_{nd}^6 \ \forall n \in N, d \in D : d \leq |D| - \underline{\gamma}_n^6 (12)$$

$$\sum_{s \in \tilde{S}} x_{nsl_{in}} - y_{ni} \leq \alpha_{ni}^8 \ \forall n \in N, i \in \tilde{W}_n (13)$$

$$\alpha_{nd_1 d_2}^9 \geq x_{nsd_1} - x_{nsd_2} \ \forall n \in N, s \in S, i \in \tilde{W}_n, d_1, d_2 \in \tilde{D}_{in} : d_1 < d_2 (14)$$

$$\alpha_{nd_1 d_2}^9 \geq x_{nsd_2} - x_{nsd_1} \ \forall n \in N, s \in S, i \in \tilde{W}_n, d_1, d_2 \in \tilde{D}_{in} : d_1 < d_2 (15)$$

$$\sum_{j \in \{1,\dots,\tilde{s}(\acute{p})\}} x_{n,\acute{p}[j],d+j-1}$$

$$\leq \tilde{s}(\acute{p}) - 1 + \alpha_{n\acute{p}d}^{11} \ \forall n \in N, \acute{p} \in \acute{P}_n, d \in \{1,\dots,|D| - \tilde{s}(\acute{p}) + 1\} (16)$$

$$\sum_{s \in S} \sum_{j \in \{1,\dots,\tilde{s}(\hat{p}) : \hat{p}[j] \geq 1\}} x_{n,s,\hat{p}[j]} +$$

$$\sum_{j \in \{1,\dots,\tilde{s}(\hat{p}) : \hat{p}[j] \leq -1\}} (1 - \sum_{s \in S} x_{n,s,-\hat{p}[j]}) \leq \tilde{s}(\hat{p}) - 1 + \alpha_{n\hat{p}}^{12} \ \forall n \in N, \hat{p} \in \hat{P}_n (17)$$

$$\sum_{i' \in \{1 \dots i\}} \sum_{i'' \in \{i \dots \tilde{W}\}} z_{n,i',i''} \leq 1 \ \forall n \in N, i \in \tilde{W} (18)$$

$$\sum_{i' \in \{1 \dots i\}} z_{n,i',i} + \sum_{i'' \in \{i+1 \dots \tilde{W}\}} z_{n,i,i''} \leq 1 \ \forall n \in N, i \in \{1 \dots |\tilde{W}| - 1\} (19)$$

$$y_{n,i} = \sum_{i' \in 1 \dots i} \sum_{i'' \in i \dots \tilde{W}} z_{n,i',i''} \ \forall n \in N, i \in \tilde{W} (20)$$

4 Dual Bound Improvement : Cutting Planes

The performance of MIP solvers on a given formulation strongly depends on how tight is the bound obtained by the linear programming relaxation (the dual bound) to the optimal solution[44]. This bound can be improved by the dynamic inclusion of violated inequalities, using the method of the cutting planes. In this section we will discuss the structure of the problem as well some strong inequalities which can be generated to improve the dual bound.

The problem considered contains mostly binary variables linked by several GUB (generalized upper bound) constraints. Constraints of this type define an implicit conflict graph [3] indicating the set of pairs of variables whose simultaneous activation is forbidden. Linear programming relaxations for these problems can be significantly strengthened by the inclusion of inequalities derived from the set packing polytope (SPP) [39]. The most common classes of cuts for SPP are the clique cuts and the odd-hole cuts. A clique inequality for a set C of conflicting variables has the form $\sum_{j \in C} x_j \leq 1$ and an odd-hole inequality with conflicting variables C can be defined as: $\sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor$. It is well known that in practice clique cuts are by far the most important ones [7]. The impact of these cuts has been explored for some hard timetabling problems [4,14]. Considering generic clique separation routines, the most common ones are the star clique and the row clique method [21,29,7]. These are fast separation routines which are used in the current version of the COIN-OR Cut Generation Library.

Our algorithm proposal considers aggressive clique separation: instead of searching for *the* most violated clique inequality we search for *all* violated clique inequalities. Some previous results indicate that this is the best strategy. In [14], for example, although authors used a branch-and-bound code to search for the most violated clique, computational results motivated the inclusion of non-optimally violated cuts found during the search. This result is consistent with reports of application of other cuts applied to different models, such as Chvàtal-Gomory cuts [23]. The option for inserting a large number of violated inequalities at once is also responsible for reviving the gomory cuts importance [17].

Our proposed clique separation routine has two main components:

1. a module to separate all violated cliques in the conflict subgraph induced by the fractional variables;
2. a lifting module which extends generated cliques considering the original conflict graph.

The clique separation module was implemented using an improved version of the Bron-Kerbosch algorithm [11]. This version implements an optimized pivoting rule [10] to speed up the discovery of maximal cliques with large weight. This rule assigns the highest priority for visiting first nodes with large modified degree (summation of node degree and of its neighbors) and weight. Although this algorithm has an exponential worst case performance, the heuristic pivot rules make the algorithm suitable not only for running in the enumeration con-
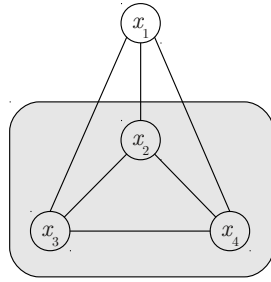
Fig. 1 Example of a k_3 which could be lifted to a k_4

text but also for executing with restricted times, since larger violated cliques tend to be discovered first. Nevertheless, our experiments showed that all violated inequalities for all instances can be enumerated in a fraction of a second using our implementation. It is also important to remark that even if a subset of cliques is inserted, the optimal solution would not be missed, branching would take care of the rest. The importance of lifting clique inequalities can be explained with the conflict graph in Figure 1. Nodes inside the gray area indicate variables with non-zero values in the fractional solution. In this solution, only nodes $x_2, \ldots, x_4$ could contribute to define a maximally violated clique inequality. Nevertheless, subsequent linear programming relaxations could include three different violated $k_3{}^2$ cliques by alternating the inactive variable. If the k_4 clique inequality were inserted during the separation of the first fractional solution, additional re-optimizations of the linear program could be saved. Furthermore, a less dense constraint matrix may be obtained with the insertion of these dominant constraints first.

It is well known that the separation of odd-holes contributes only marginally for lower bound improvement [7,35]. Nevertheless, its inclusion in the branch-and-cut procedure is cheap, since these inequalities can be separated in polynomial time using shortest path algorithms [25]. Odd hole inequalities can be strengthened by the inclusion of a wheel center, such as variable x_6 in the conflict graph presented in Figure 2. In fact, for an odd hole with variables C and W being the set of candidates to be included as wheel centers of C, the following inequality is valid:

$$\sum_{j \in W} \lfloor \frac{|C|}{2} \rfloor x_j + \sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor \tag{21}$$

The conflict graph is built by the analysis of the constraint matrix. Although the presented formulation is complete for modeling the INRC problem, we observed that solvers can detect a larger conflict graph, and thus produce stronger cuts, if the following valid inequalities are inserted:
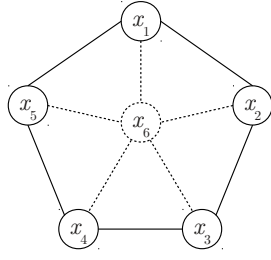
---

^{2} a clique with three nodes

Fig. 2 Example of an odd hole and its possible extension to a wheel

$$\sum_{d' \in \{1..d_1\}} r_{nd'd_1} + \sum_{d'' \in \{d_2..|D|\}} w_{nd_2d''} \leq 1 \ \forall n \in N, (d_1, d_2) \in \Pi \ : \ d_2 - d_1 = 2\}$$

$$(22)$$

$$\sum_{d' \in \{1,...,d_1\}} w_{nd'd_1} + \sum_{d'' \in \{d_2,...,|D|\}} r_{nd_2d''} \leq 1 \ \forall n \in N, (d_1, d_2) \in \Pi \ : \ d_2 - d_1 = 2\}$$

$$(23)$$

Constraints 22 simply state that no working window can start exactly two days after any resting window. This would imply the existence of an intermediate window which could not be a working or a resting window, since each one can only be followed by the other one. Constraints 23 state the same but in the inverse order. Both of these constraints are redundant considering constraints 8 and 9.

We also observed that one subset of variables is directly linked to most of the costs in the objective function: variables $w_{nd_1d_2}$ and $r_{nd_1d_2}$. In the optimal solution of the linear programming relaxation these variables often appear with fractional values, weakening the quality of the dual bound. Since the number of these active variables per nurse is quite limited, we opted for a specific cut separation for these variables. Our routine, which separates the fractional value for a restricted group of these variables per nurse was implemented using the Fenchel cutting planes [8,9]. These cuts will be called hereafter *Window* cuts.

5 Primal Bound Improvement : MIP heuristics

The use of MIP (Mixed Integer Programming) solvers in a heuristic context, i.e. for producing good quality solutions in very restricted times is a growing trend in optimization [34,42]. A pervasive term in this area is *subproblem optimization*. To speed up the improvement of feasible solutions, solvers work on smaller problems performing local search. Subproblems can be defined either with soft-fixation of variables, as in *Local Branching* and similar methods

[22,27], or with hard fixation of variables as in *Relaxation Induced Neighborhood Search* (RINS)[19]. This latter work presented better results in tests with MIPLIB[31] instances.

The proposed MIP heuristic employs a hybrid heuristic subproblem optimization scheme. Heuristic rules are used to create subproblems $\mathcal{P}'(\mathcal{H})$, defined by hard fixation of a given set of variables $\mathcal{H}$ of the original problem $\mathcal{P}$. The algorithm consists basically of two subsequent phases: construction phase and local search phase. The construction phase builds a feasible initial solution using simple heuristic rules, outside the MIP framework. MIP search is used in all the remaining time for exploring large neighborhoods until a local minimum is found. The procedure returns either the local optimum solution of all the neighborhood structures or the best solution found within max_{time} seconds.

Before presenting the MIP heuristic developed, we present a simple procedure to build feasible solutions which will be used in our experiments.

5.1 A Greedy Constructive Algorithm

This method builds an allocation matrix $M_{|N| \times |D|}$, initializing all m_{ij} cells with days off. Sequentially, for each day d and shift s, the demand $\tilde{r}_{ds}$ is satisfied by selecting, one by one, a nurse n for which this new allocation incurs the smallest increase in the objective function considering augmented partial solution defined in $M_{|N| \times |D|}$. This process is repeated until all the demand units are allocated. The algorithm has time complexity of $O(|N|^2 \times |D|)$.

5.2 Neighborhood structures

The local search phase explores the search space through several neighborhoods, using a VND (Variable Neighborhood Descent) [26] scheme. Considering the results obtained with recent uses of RINS heuristics [18], we proposed two different neighborhood structures that are based on the resolution of small partitions of the original problem to optimality. The differences between the neighborhoods lie on the rules considered to generate such subproblems.

Given a feasible solution S_0, a neighbor is obtained basically by (i) defining a set of nurse allocations that will be fixed, according to the solution S_0 and (ii) solving to optimality the NRP subproblem obtained with the fixations. In preliminary experiments, two of the evaluated neighborhoods presented much better results. The following paragraphs describe these two neighborhoods.

5.2.1 *Fix Days neighborhood structure*

In the *Fix Days* neighborhood structure, the NRP subproblems are generated by fixing all the nurse allocations of $|D| - ndays$ days of the month, where *ndays* is a parameter of the neighborhood.

In the first iteration ($iter = 0$), a subproblem is created from a solution S by fixing every nurse allocation except the ones on the days from 1 to $ndays$. The subproblem is then solved to optimality. If the solution is improved, S is updated. In the next iteration, another subproblem is generated by fixing all nurse allocations except those on the days between day_A and day_B (equations (24) and (25)):

$$day_A = 1 + (iter \times step) \tag{24}$$

$$day_B = ndays + (iter \times step) \tag{25}$$

The equations (24) and (25) calculate, respectively, the beginning and the end of a time window in which the nurse allocations remain unfixed. In these equations, $iter$ is the number of the current iteration and $step$ is a parameter that defines the number of days between two consecutive subproblems. If the value of day_A or day_B is greater than the number of days, $|D|$, we consider the day to be the remainder of its value divided by $|D|$. The algorithm proceeds until $|D|/step$ consecutive iterations without improvement are reached, which indicates that a local optimum for the neighborhood was found. Figure 3 shows how the search on the solution space is performed by the *Fix Days* neighborhood structure.

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed
N1	M	M	N	–	–	E	E	M	M	N
N2	E	N	E	E	M	–	–	E	N	E
N3	–	E	M	–	M	–	N	–	E	M

Fig. 3 *Fix Days* neighborhood subproblems with $ndays = 3$ and $step = 2$.

The neighborhood has two parameters, *step* and *ndays*. As said before, the first one indicates the number of days between two consecutive subproblems. The smaller the value, the greater the number of different subproblems in the neighborhood is. The other parameter, *ndays*, defines the size of each subproblem and so is a critical one. Small values may create subproblems that do not contain any better solution and large values may create unmanageable subproblems.

5.2.2 *Fix Shifts* neighborhood structure

In the *Fix Shifts* neighborhood structure, the subproblems are created by fixing all the allocations of $|S| - 1$ shifts. On the first iteration, only the allocations of the first shift remain unfixed. On the second iteration, only the allocations

of the second shift are not fixed, and so on. Since the number of different subproblems generated in this neighborhood is equal to $|S|$, the algorithm proceeds until $|S|$ consecutive iterations without improvement are reached. The neighborhood does not have any parameter.

Given that an average instance has from 3 to 5 shifts, it may seem that the subproblems of this neighborhood are hard to solve. But such subproblems can actually be solved in very small time, as seen in section 6.

5.3 Mathematical programming heuristic

In the latter section, we presented the neighborhood structures used by our mathematical programming heuristic (MPH). The heuristic works in a VND fashion, searching each one of the neighborhoods until their local minima are found. We decided to use the following neighborhoods in our algorithm:

$\mathcal{N}_1^m$: *Fix Days* neighborhood structure with $ndays = 2m$ and $step = m$
$\mathcal{N}_2$: *Fix Shifts* neighborhood structure

First, the algorithm performs a complete search on the neighborhoods $\mathcal{N}_1^m$ and $\mathcal{N}_2$. After that, the algorithm increases the value of m by 1, searching for the best solution on the neighborhood $\mathcal{N}_1^{m+1}$. If any improvement is produced by the latter search, the algorithm searches the neighborhood $\mathcal{N}_2$ before incrementing the value of m. Otherwise, the algorithm just increases the value of m by 1, moving to the neighborhood $\mathcal{N}_1^{m+1}$. This procedure is repeated until $m > |D|/2$ or until the time limit is reached.

Figure 4 shows the pseudocode of the proposed heuristic. In this figure, the procedures $\mathcal{N}_1^m(S_0)$ and $\mathcal{N}_2(S_0)$ return, respectively, the best neighbors of S_0 in the neighborhoods $\mathcal{N}_1^m$ and $\mathcal{N}_2$. If no better solution than S_0 is found, then S_0 is returned.

Require: S_0, m
1: $S^* \leftarrow \mathcal{N}_1^m(S_0)$
2: $S^* \leftarrow \mathcal{N}_2(S^*)$
3: $m = m + 1$
4: **while** $m \leq |D|/2$ and time limit not reached **do**
5:    $S \leftarrow \mathcal{N}_1^m(S^*)$
6:    **if** S is a better solution than S^* **then**
7:      $S^* \leftarrow \mathcal{N}_2(S)$
8:    **end if**
9:    $m = m + 1$
10: **end while**
11: **return** S^*;

Fig. 4 Pseudocode of MPH Algorithm

It is important to note that on a standard VND, it is typical to choose a specific order of neighborhoods to be searched. If one neighborhood is able

to improve the solution, VND moves back to the first neighborhood, restarting the search. Our algorithm always increments the value of m. We decided to do so because we observed that searching again the neighborhood $\mathcal{N}_1^m$ almost never improves the solution, but takes considerable processor time. Since neighborhood $\mathcal{N}_1^m$ is smaller than $\mathcal{N}_1^{m+1}$, and most neighbors of $\mathcal{N}_1^m$ are also in the neighborhood $\mathcal{N}_1^{m+1}$, looking again for a better solution in $\mathcal{N}_1^m$ can be a waste of time.

Another interesting thing to note is that if a time limit is not set, the original problem will be solved in the last iteration of the heuristic. This occurs because finding the local minimum of any solution S in the neighborhood $\mathcal{N}_1^{|D|}$ is the same as solving the original problem itself.

6 Computational Experiments

Our code was written in C++ using the open source COIN-OR libraries. This approach allowed us to deeply integrate our routines with the COIN-OR MIP solvers [24, 41, 32] and also communicate with commercial solvers through the Open Solver Interface (OSI)[40]. Closed source solvers can also have additional code integrated by using callbacks, but this approach is ultimately limited by which callbacks are available and how many decisions they delegate. Thus, all cut generation routines were implemented and tested using the COIN Branch-and-Cut solver (CBC) [24], which is the fastest open source mixed integer programming solver available [36].

The code was compiled on GCC/G++ version 4.6. We ran all the experiments on several Core i7 3.4GHz computers with 16Gb of RAM memory running Linux Ubuntu 10.10 64-bits. We used CPLEX version 12.2.0 and COIN-OR CBC 2.7.6.

The instance set used within the experiments was the same used during the INRC, including the harder *hidden* instances. Further information about these instances can be found in [28] or at the competition website[3].

Supplementary material related to this article, as well as the source code, the instances and the best known solutions can be downloaded in our website[4].

Before proceeding to the evaluation of our proposals, we present some experiments with a state-of-art integer programming solver. The objective is to determine how powerful these solvers are handling the INRC instances with the proposed formulation and the application of only small additional settings, if any.

6.1 Standalone solvers

We included experiments with the commercial CPLEX [30] standalone solver.

---

[3] http://www.kuleuven-kulak.be/nrpcompetition

[4] http://www.goal.ufop.br/nrp

In Table 3 the results of CPLEX running with different optimization emphasis with execution time restricted to 10 minutes and one hour are included. The final lower and upper (lb/ub) bounds are presented with the computed gap $\frac{ub-lb}{lb} \times 100$.

Results are summarized per instance group, including the maximum value, average and standard deviation (m/a/s.d.) gap values.

Since the production of a feasible solution for INRC instances can be done very quickly using a greedy heuristic (see section 5.1) and CPLEX standalone solver can read initial solutions, we also included experiments where CPLEX starts from an already produced feasible solution (columns with $s = gr(.)$). For sprint instances CPLEX always found the optimal solution in a few minutes, so we did not report results for these instances.

Results in Table 3 indicate that although there are large instances which are easy for the standalone solver, so that optimality was proven in less than 10 minutes, there are several instances where the solver alone (columns $s = \emptyset$) could not reach *any* feasible solution in one hour of processing time, even with the activation of the heuristic emphasis. Even though entering one initial solution (columns $s = gr(.)$) solves the feasibility problem, the final solution quality after one hour of processing time is still far from acceptable, with gaps of about 70% appearing in some cases. These results show that in spite of the progresses in generic MIP solvers, in many cases of real world applications the hybridization of these solvers with methods which consider problem specific information is still very important and in many cases absolutely necessary.

6.2 Cutting planes

The objective of the separation procedure is to speed up the improvement of the lower bound and consequently to prove the optimality faster. In the first experiment we ran several rounds of cut separation using our proposed clique separation procedure, named here as `eclq` and the clique separation routine included in the COIN-OR Cut Generation Library, denoted here as `cgl`, restricted by the following time limit: 100 seconds for sprint instances and 600 seconds for larger instances. To measure the improvements we computed for each instance and time instant the relative distance (gap) to best upper bound: the optimal solution or best known solution. Let a given lower bound be lb and an upper bound be ub, then the gap is calculated as $\frac{ub-lb}{ub} \times 100$. In Figure 5 the evolution of the average gap for groups of instances in time is presented. It can be observed that the inclusion of our lifted inequalities allows a faster reduction in the gap. Furthermore, `eclq` cuts still make progress when `cgl` cuts cannot perform any significant change in the dual limit. The separation of odd holes showed no surprises for us: as previous works say, they have no significant impact for dual bound improvement. One reason for this is that most violated odd-holes found are k_3, so that the clique separation routines already finds it. Violated odd holes of size 5 or more are scarce in the root node relaxation. Nevertheless, these are safe cuts (i.e. they do not

Table 3 Results of the standalone commercial solver CPLEX

	CPLEX12.2 default settings												CPLEX12.2 heuristic emphasis											
	10min / $s=\emptyset$			10min / $s=gr(.)$			1hour / $s=\emptyset$			1hour / $s=gr(.)$			10min / $s=\emptyset$			10min / $s=gr(.)$			1hour / $s=\emptyset$			1hour / $s=gr(.)$		
Instance	LB	UB	gap	LB	UB	gap	LB	UB	gap	LB	UB	gap	LB	UB	gap	LB	UB	gap	LB	UB	gap	LB	UB	gap
long — early																								
01	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0
02	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0
03	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
04	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0
05	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0
long — hidden																								
01	319	∞	∞	319	487	34.5	337	615	45.2	334	374	10.7	319	∞	∞	319	487	34.5	337	602	44.0	334	380	12.1
02	81	∞	∞	81	111	27.0	81	153	47.1	86	92	6.5	81	∞	∞	81	111	27.0	81	143	43.4	86	93	7.5
03	17	∞	∞	18	61	71.0	17	∞	∞	25	55	55.2	17	∞	∞	18	61	71.0	26	∞	∞	26	51	49.9
04	14	∞	∞	14	44	68.8	14	∞	∞	19	44	56.1	14	∞	∞	14	44	68.8	19	∞	∞	14	42	67.2
05	36	∞	∞	36	130	72.7	40	∞	∞	41	41	0.0	36	∞	∞	36	130	72.7	40	∞	∞	40	46	14.1
long — late																								
01	212	∞	∞	204	385	46.9	231	∞	∞	232	237	2.1	212	∞	∞	204	385	46.9	231	∞	∞	233	267	12.7
02	214	∞	∞	208	409	49.2	229	282	18.8	229	229	0.0	214	∞	∞	208	409	49.2	229	517	55.7	229	229	0.0
03	213	∞	∞	212	391	45.8	218	250	12.8	218	221	1.4	213	∞	∞	212	391	45.8	216	295	26.8	218	225	3.1
04	196	∞	∞	197	310	36.5	213	∞	∞	213	230	7.4	196	∞	∞	197	310	36.5	213	∞	∞	212	234	9.3
05	79	635	87.5	79	270	70.9	79	542	85.4	80	229	65.3	79	635	87.5	79	270	70.9	79	545	85.5	80	269	70.4
(m / a / s.d.)	(∞ / 65.8 / 48.3)			(72.7 / 34.9 / 29.0)			(∞ / 47.3 / 45.1)			(65.3 / 13.6 / 23.7)			(∞ / 65.8 / 48.3)			(72.7 / 34.9 / 29.0)			(∞ / 50.4 / 43.8)			(70.4 / 16.4 / 24.8)		
medium — early																								
01	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
02	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
03	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	**236**	0.0
04	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0
05	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0
medium — hidden																								
01	60	1093	94.5	59	314	81.3	62	287	78.3	61	227	73.0	60	1093	94.5	59	314	81.33	66	1060	93.8	64	220	71.1
02	183	∞	∞	180	352	48.9	188	312	39.6	188	258	27.0	183	∞	∞	180	352	48.9	189	291	35.2	188	291	35.3
03	22	449	95.0	22	93	76.0	23	93	74.9	24	93	74.7	22	449	95.0	22	93	75.99	24	60	60.3	24	49	51.9
04	58	∞	∞	59	136	57.0	60	97	38.6	63	96	34.8	58	∞	∞	59	136	56.99	62	96	35.8	62	109	42.8
05	56	∞	∞	56	233	75.9	84	488	82.9	78	233	66.7	56	∞	∞	56	233	75.89	84	427	80.4	82	205	60.0
medium — late																								
01	148	171	13.5	145	215	32.5	152	157	3.1	152	157	2.9	149	166	10.5	147	212	30.77	151	158	4.4	151	157	3.5
02	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0
03	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0
04	33	35	4.4	34	37	9.4	34	35	2.9	35	35	0.0	35	35	0.0	33	37	9.8	35	35	0.0	35	35	0.0
05	104	114	9.2	103	120	13.9	106	107	1.0	107	107	0.0	104	123	15.8	103	216	52.55	107	107	0.0	107	107	0.0
(m / a / s.d.)	(∞ / 34.4 / 46.6)			(81.3 / 26.3 / 32.4)			(82.9 / 21.4 / 32.5)			(74.7 / 18.6 / 29.4)			(∞ / 34.4 / 46.7)			(81.3 / 28.8 / 32.9)			(93.8 / 20.7 / 32.8)			(71.1 / 17.6 / 26.4)		

lower bound improvement - sprint instances

lower bound improvement - medium instances

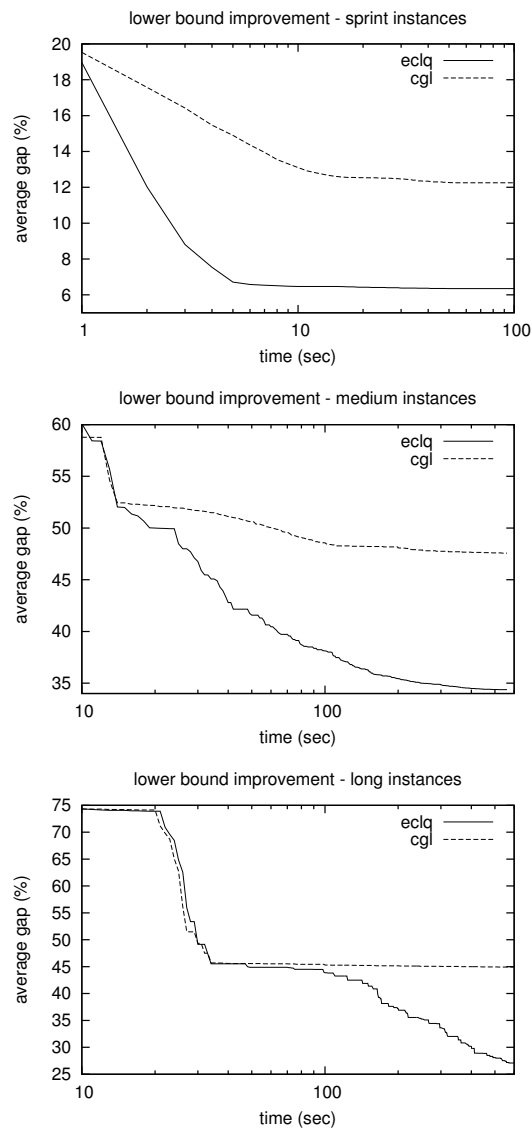lower bound improvement - long instances

Fig. 5 Dual bound improvement for COIN-OR built in cut generator (`cgl`) and the proposed cut separation procedure (`eclq`)

depend on rounding numbers computed with limited floating point accuracy) which can be instantly separated, so they are worth keeping in branch-and-cut procedure even if they have a marginal contribution observed in the root node relaxation.

		Window	Zero-Half	Gomory	RedSplit	TwoMIR
sprint	max.	50.1	51.5	93.7	52.9	52.8
	min.	1.9	3.7	3.7	3.7	3.7
	av.	19.7	21.6	26.2	22.3	22.6
	std.dev	15.2	15.3	23.1	15.8	15.9
medium	max.	100.0	100.0	100.0	100.0	100.0
	min.	0.0	0.0	0.0	0.0	0.0
	av.	50.1	51.2	51.9	50.7	41.1
	std.dev	46.6	46.3	47.7	46.7	47.9
long	max.	100.0	100.0	58.9	100.0	36.5
	min.	0.0	0.0	0.0	0.0	0.0
	av.	37.5	38.7	14.5	38.0	5.9
	std.dev	39.1	39.3	17.8	40.3	9.3

Table 4 Contribution of different cuts to improve root node relaxation lower bound

After a series of experiments with all cuts available in the COIN-OR Cut Generation Library (CGL), we found out that the following generic cutting planes were also useful in improving the dual bound: Mixed Integer Gomory[5], Two-Step Mixed Integer Rounding[20], RedSplit[1] and the Zero-Half $(0/\frac{1}{2})$[2] cuts. Zero-Half cuts are not available yet in the latest formal CGL release, but authors kindly offered the code for our experiments. The contribution of all these additional cuts applied jointly with our clique cuts for improving the lower bound for each group of instances is shown in Table 4. Considering the linear programming relaxation limit (lp) and the lower bound obtained at the end of the root node cut application in CBC (lb), we computed for each instance the improvement as $min\{\frac{lb-lp}{lp+\epsilon} \times 100, 100\}$, where ϵ is a small constant to avoid division by zero. A summary of these results is presented in Table 4. As it can be seen, although gomory cuts are of crucial importance for small instances, its relevance diminishes in larger instances. The reason is that these cuts tend to produce very dense constraints for large linear programs and are probably discarded by the branch and cut code of CBC in large instances. The proposed Window cuts, on the other hand, appear to be more important in larger instances.

6.3 Mathematical Programming Heuristic

The *MPH* uses CPLEX within the algorithm to solve the subproblems. Parallel mode was disabled, so both the heuristic and CPLEX ran sequentially. All 60 instances from the INRC [28] were tested with the parameter m assuming values from 1 to 9. The results are reported in Table 5.

In this table, the column BKS shows the best known solutions, including the ones found in this work (marked with a ⊛). The following columns show the best results found in the literature (PUB), the best results by CPLEX, the best results obtained by the *MPH* with m in the range of 1 to 9 and the best results obtained in each one of these. For each result, the table reports

the best upper bound (ub) obtained and the gap between this upper bound
(*ub*) and the best known solution (BKS): $\frac{ub-\text{BKS}}{ub} \times 100$.

The results for "sprint" instances were not reported in Table 5 because,
for all of them, the heuristic was able to find the optimal solution within 3
minutes with m starting with values in the range [1,9].

Some comments about the results shown in Tables 5:

- The impact of the parameter m is very perceptible and the best average
  result was obtained when $m = 1$.
- The upper bounds provided by the MPH are certainly very good, especially
  when $m = 1$ in the first iteration. Considering all the runs (m starting
  from 1 to 9), the heuristic was able to outperform the best solution in the
  literature for 7 instances. Considering the "sprint" instances, the gap was
  greater than 0 only for 7 out of 60 instances. The average gap from the
  best known solution was also very low: 0.5% for "long" instances and 0.6%
  for "medium" instances, with a maximum gap of 5.3% considering all the
  instances, which is specifically related to one unit on the objective function
  value of instance medium_late02.
- Since the *MPH* was able to robustly find good solutions in up to 10 minutes
  of sequential processor time, we decided not to report results of longer runs.
  Such decision is easy to be explained, since running the heuristic for longer
  times may result in solving the original problem (see section 5.3), which
  can take a very long time for some instances.
- The comparison with the upper bounds of the method and the best results
  from literature should take into account the difference of order of magni-
  tude in the running times for "long" instances. While *MPH* running times
  are limited to 600 seconds, Valouxis et al. [43] report times of up to 36,000
  seconds on these instances. The comparison is more fair when considering
  the "medium" instances, since running times are similar.

Figure 6 shows the improvement in time with regard to the upper bound by
MPH using different values for the initial m. From this figure we can conclude
that for values lower than 7 for the initial m, the *MPH* is capable to produce
good solutions in the early stages of the search. The figure also shows that,
as the value of the initial m becomes larger, the heuristic takes more time to
generate better solutions. Such additional time is not worthwhile, since the
final solution is still worse than the ones produced by *MPH* with smaller m
in the first iteration.

6.4 Best Results

The best results obtained from all experiments are presented in Table 6. Table
cells marked with ⊛ indicate some improvement over the best known solution
as reported in the INRC site at time of the writing of this work. It is impor-
tant to remember that this site has received updates in the years following the
competition, so the previous best known solutions (column PUB) were already

		Instance	BKS	PUB UB	PUB gap	CPLEX (m = 14) UB	CPLEX gap	best MPH (m ∈ [1,9]) UB	best MPH gap	m = 1 UB	gap	m = 2 UB	gap	m = 3 UB	gap	m = 4 UB	gap	m = 5 UB	gap	m = 6 UB	gap	m = 7 UB	gap	m = 8 UB	gap	m = 9 UB	gap
long	early	01	197	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0
		02	219	219	0.0	219	0.0	219	0.0	219	0.0	220	0.5	220	0.5	220	0.5	220	0.5	219	0.0	219	0.0	219	0.0	219	0.0
		03	240	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0
		04	303	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0
		05	284	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0
	hidden	01	⊛ 346	363	4.7	487	29.0	346	0.0	347	0.3	348	0.6	346	0.0	346	0.0	359	3.6	346	0.0	355	2.5	534	35.2	506	31.6
		02	⊛ 89	90	1.1	111	19.8	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	96	7.3
		03	38	38	0.0	61	37.7	39	2.6	39	2.6	40	5.0	40	5.0	41	7.3	41	7.3	42	9.5	130	70.8	119	68.1	118	67.8
		04	22	22	0.0	44	50.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	37	40.5	41	46.3	112	80.4
		05	41	41	0.0	130	68.5	41	0.0	43	4.7	45	8.9	42	2.4	43	4.7	43	4.7	43	4.7	41	0.0	49	16.3	174	76.4
	late	01	235	235	0.0	385	39.0	239	1.7	239	1.7	241	2.5	239	1.7	241	2.5	242	2.9	244	3.7	244	3.7	244	3.7	257	8.6
		02	229	229	0.0	409	44.0	235	2.6	235	2.6	235	2.6	237	3.4	235	2.6	239	4.2	242	5.4	235	2.6	242	5.4	243	5.8
		03	220	220	0.0	391	43.7	220	0.0	221	0.5	225	2.2	221	0.5	220	0.0	233	5.6	239	7.9	233	5.6	220	0.0	380	42.1
		04	221	221	0.0	310	28.7	222	0.5	225	1.8	231	4.3	228	3.1	224	1.3	229	3.5	227	2.6	222	0.5	228	3.1	233	5.2
		05	83	83	0.0	270	69.3	83	0.0	86	3.5	83	0.0	83	0.0	83	0.0	83	0.0	83	0.0	92	9.8	159	47.8	240	65.4
	gap (avg./std. deviation)			(0.4/1.2)		(28.6/24.7)		(0.5/0.9)		(1.2/1.5)		(1.8/2.6)		(1.1/1.6)		(1.3/2.2)		(2.1/2.5)		(2.3/3.3)		(9.1/20.0)		(15.1/22.7)		(26.0/31.6)	
medium	early	01	240	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0
		02	240	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0
		03	236	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0
		04	237	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0
		05	303	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0
	hidden	01	⊛ 111	130	14.6	314	64.6	111	0.0	116	4.3	111	0.0	118	5.9	118	5.9	116	4.3	117	5.1	118	5.9	120	7.5	307	63.8
		02	221	221	0.0	352	37.2	221	0.0	221	0.0	223	0.9	224	1.3	227	2.6	223	0.9	235	6.0	221	0.0	226	2.2	256	13.7
		03	⊛ 34	36	5.6	93	63.4	34	0.0	34	0.0	40	15.0	41	17.1	38	10.5	36	5.6	38	10.5	36	5.6	36	5.6	41	17.1
		04	⊛ 78	80	2.5	136	42.6	80	2.5	80	2.5	80	2.5	84	7.1	81	3.7	82	4.8	81	3.7	81	3.7	80	2.5	147	46.9
		05	⊛ 119	122	2.5	233	48.9	119	0.0	121	1.7	120	0.8	123	3.3	121	1.7	120	0.8	119	0.0	127	6.3	121	1.7	232	48.7
	late	01	⊛ 157	158	0.6	215	27.0	157	0.0	161	2.5	160	1.9	160	1.9	161	2.5	161	2.5	157	0.0	159	1.3	161	2.5	164	4.3
		02	18	18	0.0	18	0.0	19	5.3	19	5.3	19	5.3	22	18.2	23	21.7	24	25.0	20	10.0	20	10.0	20	10.0	24	25.0
		03	29	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0
		04	35	35	0.0	37	5.4	35	0.0	35	0.0	36	2.8	35	0.0	36	2.8	36	2.8	37	5.4	37	5.4	35	0.0	36	2.8
		05	107	107	0.0	120	10.8	108	0.9	108	0.9	114	6.1	113	5.3	118	9.3	114	6.1	114	6.1	119	10.1	109	1.8	112	4.5
	gap (avg./std. deviation)			(1.7/3.9)		(20.0/24.9)		(0.6/1.5)		(1.1/1.7)		(2.4/4.0)		(4.0/6.0)		(4.1/6.0)		(3.5/6.4)		(3.1/3.8)		(3.2/3.7)		(2.2/3.1)		(15.1/21.3)	

Table 5 Results of the Mathematical Programming Heuristic
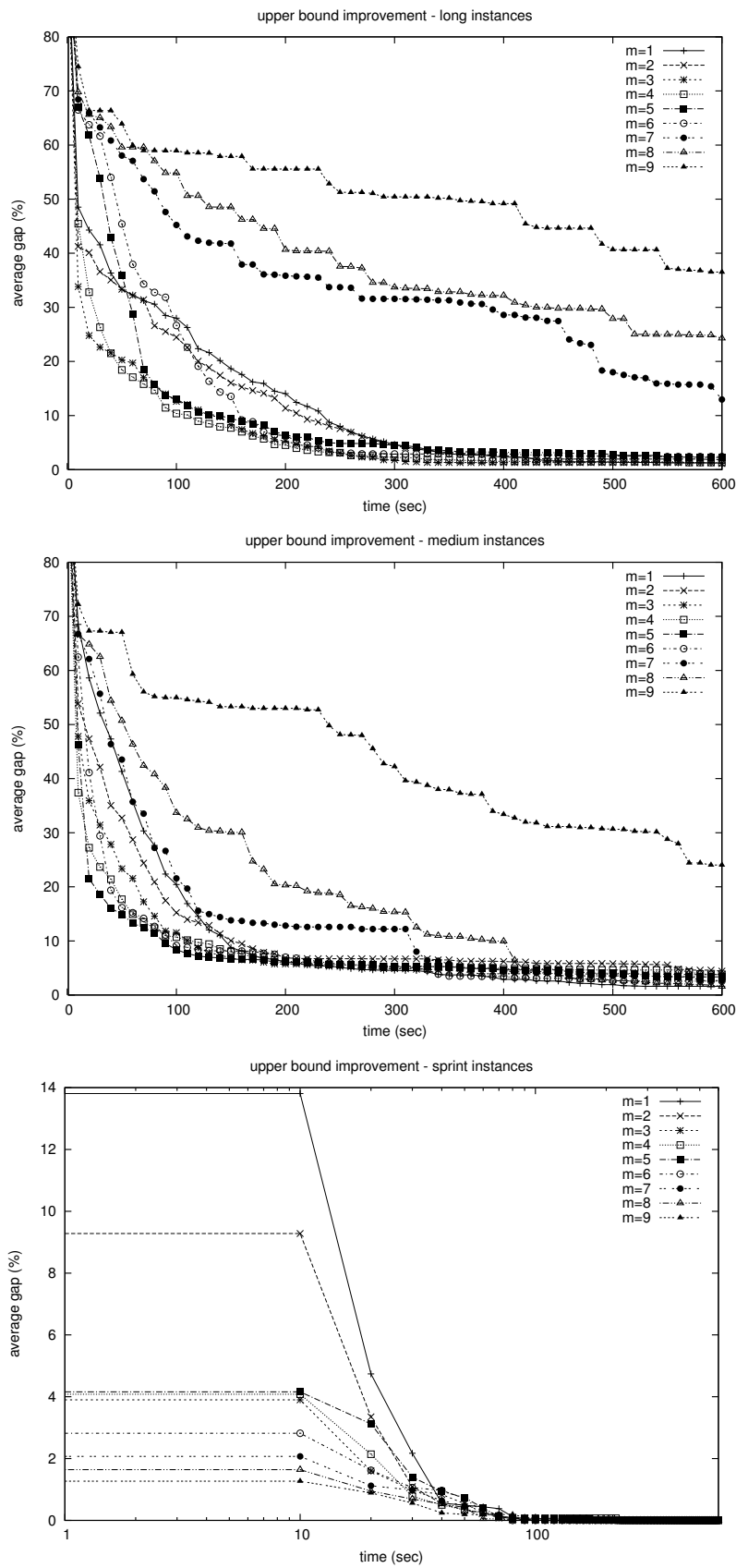
Fig. 6 Improvement in time graphic for MPH with different values for the initial m.

Instance			LB	PUB	UB	GAP
long	early	01	197.0	197	197	0.0
		02	219.0	219	219	0.0
		03	240.0	240	240	0.0
		04	303.0	303	303	0.0
		05	284.0	284	284	0.0
	hidden	01	341.0	363	⊛ **346**	1.4
		02	86.0	90	⊛ **89**	3.4
		03	35.3	38	38	7.1
		04	19.0	22	22	13.8
		05	41.0	41	41	0.0
	late	01	232.0	235	235	1.3
		02	229.0	229	229	0.0
		03	219.0	220	220	0.5
		04	214.6	221	222	3.3
		05	83.0	83	83	0.0
medium	early	01	240.0	240	240	0.0
		02	240.0	240	240	0.0
		03	236.0	236	236	0.0
		04	237.0	237	237	0.0
		05	303.0	303	303	0.0
	hidden	01	87.2	130	⊛ **111**	21.5
		02	196.6	221	221	11.1
		03	27.7	36	⊛ **34**	18.5
		04	72.8	80	⊛ **78**	6.7
		05	90.8	122	⊛ **119**	23.7
	late	01	155.7	158	⊛ **157**	0.8
		02	18.0	18	18	0.0
		03	29.0	29	29	0.0
		04	35.0	35	35	0.0
		05	107.0	107	107	0.0

Instance			LB	PUB	UB	GAP
sprint	early	01	56.0	56	56	0.0
		02	58.0	58	58	0.0
		03	51.0	51	51	0.0
		04	59.0	59	59	0.0
		05	58.0	58	58	0.0
		06	54.0	54	54	0.0
		07	56.0	56	56	0.0
		08	56.0	56	56	0.0
		09	55.0	55	55	0.0
		10	52.0	52	52	0.0
	late	01	32.0	33	⊛ **32**	0.0
		02	32.0	32	32	0.0
		03	62.0	62	62	0.0
		04	66.0	67	⊛ **66**	0.0
		05	59.0	59	59	0.0
		06	130.0	134	⊛ **130**	0.0
		07	153.0	153	153	0.0
		08	204.0	209	⊛ **204**	0.0
		09	338.0	338	338	0.0
		10	306.0	306	306	0.0
	hidden	01	37.0	37	37	0.0
		02	42.0	42	42	0.0
		03	48.0	48	48	0.0
		04	73.0	75	⊛ **73**	0.0
		05	44.0	44	44	0.0
		06	42.0	42	42	0.0
		07	42.0	42	42	0.0
		08	17.0	17	17	0.0
		09	17.0	17	17	0.0
		10	43.0	43	43	0.0

Table 6 previous upper bound (PUB), updated lower (LB) and upper (UB) bounds

very hard to find. Most instances were solved to optimality and the proposed heuristic was able to improve the best known solutions even for the hidden instances. So far, the harder among the unsolved instances is medium_hidden05, where the lower bound distance is now at 23.7%.

7 Conclusions and Future Works

This work presented Integer Programming techniques for the Nurse Rostering Problem. Although there are several detailed results published in the literature for heuristics evaluated using the INRC instance set, we believe that this is the first work which also devotes a considerable attention to the computational production of strong dual bounds obtained from the linear programming relaxation. These bounds allowed us to prove the optimality for many instances, and its importance is not restricted to exact methods. Improved dual bounds allow a more effective pruning of nodes in the search tree, which is useful to speed up MIP heuristic search in large neighborhoods, as the one presented in this work. The large number of experiments made using the open source CBC solver allowed us to spot existing previously unknown CBC bugs. These bugs were subsequently fixed by CBC developers. We proposed and implemented a much better clique cut generator for CBC, showing that it can produce better dual bounds in the early stages of the search. This code will also be released as open source. This work was not enough to make CBC competitive with the best commercial solvers when solving INRC instances, since to develop a competitive MIP heuristic we still needed to rely on CPLEX. Nevertheless, we believe that this is a step toward validating and improving this important open source integer programming solver.

The proposed MIP heuristic, built upon the presented formulation and evaluated with the state-of-art CPLEX solver, improved several best known solutions, requiring very short computing times and still being competitive with the best heuristics for this problem. We believe that the new improved primal and dual bounds will allow a more precise evaluation of the quality of available heuristics for this problem.

8 Acknowledgements

References

1. Andersen, K., Cornuejols, G., Y., L.: Reduce-and-split cuts: Improving the performance of mixed integer gomory cuts. Management Science **51**, 1720–1732 (2005)
2. Andreello, G., Caprara, A., Fischetti, M.: Embedding cuts in a branch and cut framework: a computational study with 0,1/2-cuts. INFORMS Journal on Computing 1 **19**(2), 229–238 (2007)
3. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.P.: Conflict graphs in solving integer programming problems. European Journal of Operational Research **121**, 40–55 (2000)
4. Avella, P., Vasil'ev, I.: A computational study of a cutting plane algorithm for university course timetabling. Journal of Scheduling **8**, 497–514 (2005)
5. Balas, E., Ceria, S., Cornuejlos, G., Natra, N.: Gomory cuts revisited. Operations Research Letters **19**, 1–10 (1996)
6. Bilgin, B., Demeester, P., Mısır, M., Vancroonenburg, W., Berghe, G., Wauters, T.: A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10) (2010)
7. Borndorfer, R.: Aspects of set packing, partitioning, and covering. Ph.D. thesis, Faculty of Mathematics at Technical University of Berlin (1998)
8. Boyd, E.: Fenchel cutting planes for integer programming. Operations Research **42**, 53–64 (1992)
9. Boyd, E.: Solving 0/1 integer programs with enumeration cutting planes. Annals of Operations Research **50**, 61–72 (1994)
10. Brito, S., Santos, H.G.: Pivoting in the bron-kerbosch algorithm for maximum-weight clique detection (in portuguese). In: Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional (2011)
11. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM **16**(9), 575–577 (1973)
12. Burke, E., Curtois, T.: An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10) (2010)
13. Burke, E., Li, J., Qu, R.: A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. European Journal of Operational Research **203**(2), 484–493 (2010)
14. Burke, E., Mareček, K., Parkes, A.J., Rudová, H.: A branch-and-cut procedure for the udine course timetabling problem. Annals of Operations Research pp. 1–17 (2011)

15. Burke, E.K., De Causmaecker, P., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. Journal of Scheduling **7**, 441–499 (2004)
16. Cheang, B., Li, H., Lim, A., Rodrigues, B.: Nurse rostering problems–a bibliographic survey. European Journal of Operational Research **151**(3), 447–460 (2003)
17. Cornuéjols, G.: Revival of the gomory cuts in the 1990´s. Annals of Operations Research **149**(1), 63–66 (2007)
18. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve mip solutions. Tech. rep., ILOG (2003)
19. Danna, E., Rothberg, E., Le Pape, C.: Integrating mixed integer programming and local search: A case study on job-shop scheduling problems. In: Proceedings Third International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR'03) (2003)
20. Dash, S., Goycoolea, M., Gunluk, O.: Two step MIR inequalities for mixed-integer programs. INFORMS Journal on Computing (2009)
21. Eso, M.: Parallel branch-and-cut for set partitioning. Ph.D. thesis, Cornell University Ithaca, NY, USA (1999)
22. Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**, 23–47 (2003)
23. Fischetti, M., Lodi, A.: Optimizing over the first Chvàtal closure. Mathematical Programming B **110**(1), 3–20 (2007)
24. Forrest, J., Lougee-Heimer, R.: CBC user guide. INFORMS Tutorials in Operations Research. pp. 257–277 (2005)
25. Grotschel, M., Lovasz, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer (1993)
26. Hansen, P., Mladenović, N.: Variable neighborhood search. Computers & Operations Research **24**(11), 1097–1100 (1997)
27. Hansen, P., Mladenović, N., Urosević, D.: Variable neighborhood search and local branching. Computers & Operations Research **33**(10), 3034–3045 (2006)
28. Haspeslagh, S., De Causmaecker, P., Stolevik, M., A., S.: First international nurse rostering competition 2010. Tech. rep., CODeS, Department of Computer Science, KU Leuven Campus Kortrijk. Belgium (2010)
29. Hoffman, K., Padberg, M.: Solving airline crew scheduling problems by branch-and-cut. Management Science **39**(6), 657–682 (1993)
30. IBM: CPLEX 12.2 User's Manual (2011)
31. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**, 103–163
32. Linderoth, J.T., Ralphs, T.K.: Noncommercial software for mixed-integer linear programming. In: J. Karlof (ed.) Integer Programming: Theory and Practice, *Operations Research Series*, vol. 3 (2005)
33. Lougee-Heimer, R.: The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. IBM Journal of Research and Development **47**(1), 57–66 (2003)
34. Martins, A.X., Souza, M.C., Souza, M.J., Toffolo, T.A.M.: GRASP with hybrid heuristic-subproblem optimization for the multi-level capacitated minimum spanning tree problem. Journal of Heuristics **15**, 133–151 (2009)
35. Méndez-Díaz, I., Zabala, P.: A cutting plane algorithm for graph coloring. Discrete Applied Mathematics **156**, 159–179 (2008)
36. Mittelmann, H.: Benchmarks for optimization software (2012). URL http://plato.asu.edu/bench.html
37. Nonobe, K.: INRC2010: An approach using a general constraint optimization solver. The First International Nurse Rostering Competition (INRC 2010) (2010)
38. Nonobe, K., Ibaraki, T.: A tabu search approach to the constraint satisfaction problem as a general problem solver. European Journal of Operational Research **106**(2-3), 599–623 (1998)
39. Padberg, M.: On the facial structure of set packing polyhedra. Mathematical Programming **5**(1), 199–215 (1973)
40. Ralphs, T., Saltzman, M., Ladnyi, L.: The COIN-OR Open Solver Interface: Technology Overview (2004). URL http://www.coin-or.org/Presentations/CORS2004-OSI.pdf

41. Ralphs, T.K., Gzelsoy, M.: The symphony callable library for mixed integer programming. In: B. Golden, S. Raghavan, E. Wasil, R. Sharda, S. Vo (eds.) The Next Wave in Computing, Optimization, and Decision Technologies, *Operations Research/Computer Science Interfaces Series*, vol. 29, pp. 61–76. Springer US (2005)
42. Uchoa, E., Toffolo, T.A.M., de Souza, M.C., Martins, A.X., Fukasawa, R.: Branch-and-cut and hybrid local search for the multi-level capacitated minimum spanning tree problem. Networks **59**(1), 148–160 (2012)
43. Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., Housos, E.: A systematic two phase approach for the nurse rostering problem. European Journal of Operational Research (2012)
44. Wolsey, L.: Integer Programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley (1998)