Integer Programming Techniques for the Nurse Rostering Problem

Santos, H.G., Toffolo, T.A.M., Ribas, S., Gomes, R.A.M.

Received: date / Accepted: date

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 on 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 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).

Keywords Nurse Rostering · Integer Programming · Cutting Planes · Heuristics

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 in particularities of certain institutions. For these works, comparison between different search strategies is a very difficult task. Recently, 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 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 vari-

Computing Department
Federal University of Ouro Preto
Ouro Preto, Minas Gerais, Brazil 35400-000

ables, for the problem described in INRC. We propose and evaluate techniques for improving the performance of state-of-art integer programming 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 aiming at proving the optimality. In the second group we present techniques to speedup 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 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	E	N
N2	E	E	M	-	-				
N3	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. Here follow brief descriptions of approaches that succeeded in the competition.

Valouxis et al. [44], 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 partitionated the problem instances into sub-problems of manageable computational size which were then solved sequentially using Integer Mathematical Programming. Also, they applied local optimization techniques for searching across combinations of nurses' partial schedules. This sequence was repeated several times depending on the available computational time.

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 problem tabu search based solver presented in [38]. 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 consulted on the site of the competition[1].

Another recent work developed by Burke et al. [13] is a hybrid multi-objective model that combines integer programming (IP) and variable neigh-

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

bourhood 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 postprocessing 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 constraint 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 following hard constraints, as defined in INRC:

– a nurse can not work more than one shift per day;
– all shift type demands during the planning period must be met.

and the following soft constraints:

– minimum/maximum number of shifts assigned to a nurse;
– minimum/maximum number of consecutive free days;
– minimum number of consecutive working days;
– maximum number of consecutive working weekends;
– number of days off after a series of night shifts;
– maximum number of working weekends in four weeks;
– complete weekends: if a nurse has to work only on some days of the weekend then penalty occurs;
– identical shift types during the weekend: assignments of different shift types to the same nurse during a weekend are penalized;
– 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;
– shift on/off request: similar to the previous but now for specific shifts on certain days;
– unwanted patterns: an unwanted pattern is a sequence of assignments that is not in the preferences of a nurse based on her contract;
– 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;
– unwanted patterns not involving specific shift types;
– unwanted patterns involving specific shift types.

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

3 An Integer Programming Formulation for the INCR Problem

In this section we present an Integer Programming formulation which successfully models all constraints considered in instances of the International Nurse Rostering Competition.

3.1 Input Data

N set of nurses
C set of contracts
c_n contract of nurse n
S set of shifts
S^N set of night shifts
D set of days with elements sequentially numbered from 1
H set of all ordered pairs $(d_1, d_2) \in D \times D : d_1 < d_2$ representing windows in the planning horizon
W_c set of weekends in the planning horizon according to the weekend definition of contract c, with elements numbered from 1 to w_c
D_{ic} set of days in the i-th weekend of contract c
r_{ds} number of required nurses at day d and shift s
P_c set of unwanted working shift patterns for contract c
P_c set of unwanted working days patterns for contract c

The configuration of soft constraints depends on each contract c. Thus, each contract has an associated weight (which may be null) for penalizing the violation of each soft constraint. Limits informing how tight is a given soft constraint are also contract related.

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 $v < v < v$. Values outside this range need to penalized in slack variables according to its distance to the closest valid value.

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. This index will be used to express constants which state the minimum and maximum limit for a given ranged soft constraint i and a contract c, which will be denoted here by γ_c and γ_c, respectively. The weight for violating the i-th minimum and maximum limit of these constraints is denoted by w_c and w_c, respectively. For logical soft constraints the weight of penalizing them is defined by w_c. Finally, we denote by a_n,a_n a_n the slack variables associated with the the violation of lower/upper limit of ranged and logical soft constraints i for nurse n, respectively. Additional indexes in the a_n variables may be used, for example, when this violation must be computed for a specific location, so that a^7_{nk} is the slack variable related to the violation of the complete weekends soft constraint for nurse n at the k-th weekend.

Ranged Soft Constraints	
1	total number of allocations
2	contiguous working days
3	contiguous resting days
4	total number of working weekends in four weeks
5	consecutive working weekends
6	number of resting days after a night shift

Logical Soft Constraints	
7	complete weekends
8	no night shift before free weekend
9	same shift on weekends
10	alternative shifts
11	undesired working shifts pattern
12	undesired working days pattern
13	undesired shifts and days

Table 2 Indexes for ranged and logical soft constraints

Some specific sequences of working shifts (soft constraint 11) may be unwanted, e.g.: Late, Evening, Late. The set of these patterns for contract c is specified in $\acute{P}_c$ and each pattern $\acute{p} \in \acute{P}_c$ has a size $\breve{s}(\acute{p})$ and contents $\acute{p}[1],\ldots,\acute{p}[\breve{s}(\acute{p})] \in S$. 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 $\hat{P}_c$ with elements $\hat{p} \in \hat{P}_c$ with size $\breve{s}(\hat{p})$. To specify which days from the pattern represent the "not working" option we define a set of virtual days $\breve{D}$ with negative numbered days representing this option, so that pattern elements $\hat{p}[1],\ldots,\hat{p}[\breve{s}(\hat{p})]$ are restricted to be in $D \cup \breve{D}$.

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:

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

$$w_{nd_1d_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from day } d_1 \text{ until day } d_2 \\ 0 & \text{otherwise} \end{cases}$$

(

1 if nurse n rests from day d1 until day d2
rnd1d2 = 0 otherwise

$$\sum_{(d_1 d_2) \in \Pi} (\sigma_{c_n d_1 d_2} w_{n d_1 d_2} + \tau_{c_n d_1 d_2 m d_1 d_2}) +$$

$$\sum_{s \in S} \sum_{d \in D} v_{nsd} x_{nsd} + \omega_{1 c_n a_{1n}} + \omega_{1 c_n a_{1n}} +$$

$$\sum_{i \in \{1 \ldots w_c\}} (\omega_{4 c_n} a_{4ni} + \omega_{8 c_n a_{8ni}} + \omega_{9 c_n a_{9ni}}) +$$

$$\sum_{n \in N}$$

$$\sum_{i_1, i_2 \in W_{c_n} : i_2 \geq i_1} \psi_{ni_1 i_2} z_{ni_1 i_2} +$$

$$\sum_{d \in D} \left(a_{6_n n} + \sum_{p \in P_{c_n}} a^{11}_{np} + \sum_{p \in P_{c_n}} a^{12}_{np} \right)$$

(

$$0z_{ni_1i_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from weekend } i_1 \text{ until weekend } i_2 \\ & \text{otherwise} \end{cases}$$

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

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 $_{\sigma_{cd_1d_2}}$ and $_{\tau_{cd_2}}$ 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 nurses of contract c, respectively. Soft constraints 10 and 13 are also directly penalized in $_{x_{nsd}}$ variables with coefficients $_{v_{nsd.}}$ Analogously, soft constraint 5 is penalized in variables $_{z_{ni_1i_2}}$ with coefficients $\psi_{ni_1i_2}$.

Minimize:

3.4 Constraints

Constraints are presented in the following. 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 which indicate working weekends. 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 following 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 consider the maximum number of consecutive weekends. Constraints 13 impose a minimum number of resting days after a sequence of night shifts. Constraints 14 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 15 and 16. Undesired patterns for days and shifts are modeled in constraints 17 and 18.

$$\sum_{\nu \in N} x_{\nu\sigma\delta} = r_{\delta\sigma} \quad \forall d \in D, s \in S \tag{1}$$

$$\sum_{\sigma \in \Sigma} x_{\nu\sigma\delta} \leq 1 \quad \forall n \in N, d \in D \tag{2}$$

$$\sum_{\sigma \in \Sigma} x_{\nu\sigma\delta} \geq y_{\nu\iota} \quad \forall n \in N, i \in [w_{\chi_n}, d \in [D_{\iota_{\chi_n}} \tag{3}$$

$$\sum_{\sigma\in\Sigma,\delta\in [\Delta i_{c_n}} x_{\nu\sigma\delta} < y_{\nu\iota} \quad \forall n \in N, i \in [w_{\chi_n} \tag{4}$$

$$\sum_{\sigma\in\Sigma} x_{\nu\sigma\delta} = \sum_{(\delta 1,\delta 2)\in\angle : \delta\in\{\delta 1,\ldots,\delta 2\}} w_{\nu\delta 1\delta 2} \quad \forall n \in N, d \in D \tag{5}$$

$$\sum_{\sigma\in\Sigma} x_{\nu\sigma\delta} = 1 - \left(\sum_{(\delta 1,\delta 2)\in\angle :\ \delta\in\{\delta 1,\ldots,\delta 2\}} r_{\nu\delta 1\delta 2} \right) \quad \forall n \in N, d \in D \tag{6}$$

$$\sum_{(\delta 1,\delta 2)\in\angle :\delta\in\{\delta 1\ldots\delta 2\}} (w_{\nu\delta 1\delta 2} + r_{\nu\delta 1\delta 2}) = 1 \quad \forall n \in N, d \in D \tag{7}$$

$$\sum_{\delta^0\in\{1,\ldots,\delta\}} w_{\nu\delta^0\delta} + \sum_{\delta''\in\Delta:\delta''\geq\delta+1} w_{\nu,\delta+1,\delta''} < 1 \quad \forall n \in N, d \in D \tag{8}$$

$$\sum_{\delta^0\in\{1,\ldots,\delta\}} r_{\nu\delta'\delta} + \sum_{\delta''\in\Delta:\delta''\geq\delta+1} r_{\nu,\delta+1,\delta''} < 1 \quad \forall n \in N, d \in D \tag{9}$$

$$\gamma^1_{\chi_n} \leq \sum_{\sigma\in\Sigma,\ \delta\in\Delta} x_{\nu\sigma\delta} < \gamma_{1\chi_n} + \alpha^1_\nu \quad \forall n \in N \tag{10}$$

$$\sum_{\iota^0\in\{\iota,\ldots,\iota+3\}} y_{\nu\iota''} < \gamma_{4\chi_n} + \alpha_{4\nu\iota} \quad \forall n \in N, i \in \{1, \ldots, w_{\chi_n} - 3\} \tag{11}$$

$$\sum_{\iota_0 \in \{\iota,\ldots,\iota+\mathrm{©}5\ \tilde{}c(n)\}} y_{\nu\iota}\,\delta\ \gamma 5\tilde{}\chi_n + \alpha^5{}_{\nu\iota} \in n \in N,\ i \in \{1,\ldots,\ \tilde{}w\tilde{}\chi_n \in \gamma 5\tilde{}\chi_n\} \quad (12)$$

$$\mathrm{E}\gamma^6{}_{\Sigma}\tilde{}\chi_n\ \sigma_0\ x\sigma\ {}_{0\iota} + \sum_{\sigma \in \Sigma.\therefore\ \tilde{}\Sigma,\delta^0 \in \{\delta+1,\ldots,\delta+\mathrm{©}6\ \tilde{}c(n)\}} x\nu\sigma\delta^0$$

$$\delta\ \gamma 6\tilde{}\chi_n + \alpha^6{}_{\nu\delta} \in n \in N,\ d \in D : d\ \delta|D| \in \gamma 6\tilde{}\chi(\nu) \quad (13)$$

$$\sum_{\sigma \in \Sigma\ \tilde{}\ \delta \in\ \tilde{}\Omega r\tilde{}_{\sigma\iota}:\delta\varepsilon 2\ni\delta\delta^{\dagger}\in\delta^0 \in\ \tilde{}\Omega r\tilde{}cn} x\nu,\sigma,\delta \in 1 + y_{\nu\iota}\ \delta\ 1 + \alpha^8{}_{\nu\iota} \in n \in N,\ i \in\ \tilde{}W\iota\tilde{}\chi_n \quad (14)$$

$$\alpha^9{}_{\nu}\ \varepsilon\ x\nu\sigma\delta 1 \in x\nu\sigma\delta 2 \in n \in N,\ s \in S,\ i \in\ \tilde{}W\tilde{}\chi_n,\ d1,\ d2 \in\ \tilde{}D\iota\tilde{}\chi(\nu):d1 < d2 \quad (15)$$

$$\alpha 9\ \nu\ \varepsilon\ x\nu\sigma\delta 2 \in x\nu\sigma\delta 1 \in n \in N,\ s \in S,\ i \in\ \tilde{}W\tilde{}\chi_n,\ d1,\ d2 \in\ \tilde{}D\iota\tilde{}\chi(\nu):d1 < d2 \quad (16)$$

$$\sum_{\varphi \in \{1,\ldots,\ \tilde{}\sigma(\acute{}\pi)\}} x\nu,\acute{}\pi[1],\delta+\varphi \in 1\ \ \tilde{}s(\acute{}p)+\alpha^{11}{}_{\nu\acute{}\pi} \in n \in N,p\acute{} \in\ \acute{}P\tilde{}\chi_n,\ d \in \{1,\ldots,|D| \in\ \tilde{}s(\acute{}p) + 1\} \quad (17)$$

$$\sum_{\sigma \in \Sigma}\ \sum_{\varphi \in \{1,\ldots,\ \tilde{}\sigma(\hat{}\pi):\hat{}\pi[\varphi]\varepsilon 1\}} x\nu,\sigma,\hat{}\pi[\varphi]+ \sum_{\varphi \in \{1,\ldots,\ \tilde{}\sigma(\hat{}\pi):\hat{}\pi[\varphi]\delta \in 1\}} (1^{X} \in\ x\nu,\sigma,\in\hat{}\pi[\varphi])\ \delta\ s(\hat{}p)+\alpha^{12}{}_{\nu} \in n \in N,\ p\hat{} \in\ P\tilde{}\chi_n \quad (18)$$

4 Dual Bound Improvement : Cutting Planes

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 $\sum_{i \in C} x_i \leq 1$ and an odd-hole inequality with conflicting variables C can be define as: $\sum_{i \in C} x_i \leq \frac{|C|-1}{2}$. 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 Chv`atal-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]. The 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 context 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 important to remark also that even if a subset of cliques were inserted, the optimal solution would not be missed, branching would take care of the rest. This situation does not occur in column generation: an interruption of the pricing algorithm before the optimal column to be discovered in the last iteration would make it impossible to prove the optimality of the discovered solution. In other words, for exact algorithms the cut separation problem can be hard, but column generation cannot, as pointed in [40]. 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 at the first fractional solution additional re-optimizations of the linear program could be saved, furthermore, a less dense constraint matrix will 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:

$$\sum_{j \in W} \frac{\beta |C|}{2} \chi x_j + \sum_{j \in C} \frac{x_j \delta \beta |C|}{2} \chi \qquad (19)$$

[2] a clique with three nodes

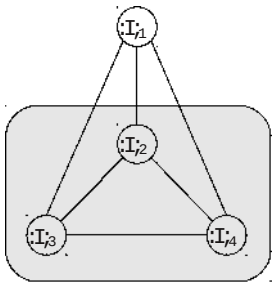

Fig. 1 Example of a k_3 which could be lifted to a k_4

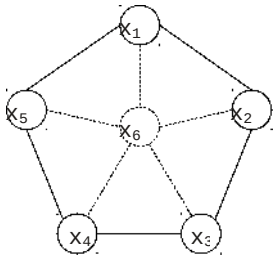

Fig. 2 Example of an odd hole and its possible extension to a wheel

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 if the following valid inequalities are inserted:

$$\sum_{d_1 \in \{1..d_1\}} \sum_{d_2 \in \{d_2..|D|\}} r_{nd_1d_1} + w_{nd_2d_2} < 1 \quad \forall n \in N, (d_1,d_2) \in \Pi : d_2 - d_1 = 2\} \tag{20}$$

$$\sum_{d_1 \in \{1,...,d_1\}} \sum_{d_2 \in \{d_2,...,|D|\}} w_{nd_1d_1} + r_{nd_2d_2} < 1 \quad \forall n \in N, (d_1,d_2) \in \Pi : d_2 - d_1 = 2\} \tag{21}$$

We also observed that one subset of variables is directly linked to most of the costs in the objective function: variables w_{nd1d2} and r_{nd1d2}. 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 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 [34,43]. A pervasive term in this area is *subproblem optimization. To* speedup 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 $\Pi^0(H)$, defined by hard fixation of a given set of variables$H$ of the original problem Π. 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 maxtime 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 $M|N|><|D|$, initializing all m_{ij} cells with days off. Sequentially, for each day d and shift s, the demand 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 $M|N|><|D|$. This process is repeated until all the demand units are allocated. The algorithm has time complexity of $O(|N|^2 \cdot |D|)$.

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.

s we t ee ue s d i h odi d b

5.2.1 Fix Days neighborhood structure

In the *Fix Days* neighborhood structure, the NRP subproblems are generated esroun e m aia asTesenoperoiscearp aa aper is organized as follows: n section an nfml description by fixing all the nurse allocations of |D| − ndays days of the month, where The pper is organized as follows inse blem is presented along a brief descriti ndays is a parameter of the neighborhood.

terature. In section 3 the NRP roblem is formaly statd usin

In the first iteration (iter =0), a subproblem is created from a solution prooed in literature. In secion 3 the NRP S by fixing every nurse allocation but the ones on the days from 1 to ndays.

ed formulaion. our proposed formulaton.

The subproblem is ten solved to optimality. If he solton is improved, S is ally statedusingthecompact isIntegerformallyProgrammingstatedusingformulationthecompacnt is work Afterwards, MIP neighborhood are pesented. Following, updated. In the next iteration, another subprblem is generated by fixng all used in this work. Afterwards, MIP neighbor nurse allocations but those on he days between day_A and day_B (equations our heuristic which was ued to we rovide present our initial heuriti feasibe whch soution. was used to p (22) and (23)).

ona experimen

$$day_A = 1 + (iter \times step) \qquad (22)$$

$$day_B = ndays + (iter \times step) \qquad (23)$$

The equations (22) and (23) calculate, respectively, the beginning and the end of the subproblem window ... until |D|/step consecutive iterations without improvement are reached, which may include morning shi (M), evening shift (E, night ... indicates that a local optimum for the neighborhood was fond.

Figure 3 shows how the search on the solution space is performed by the *Fix Days* neighborhood structure.

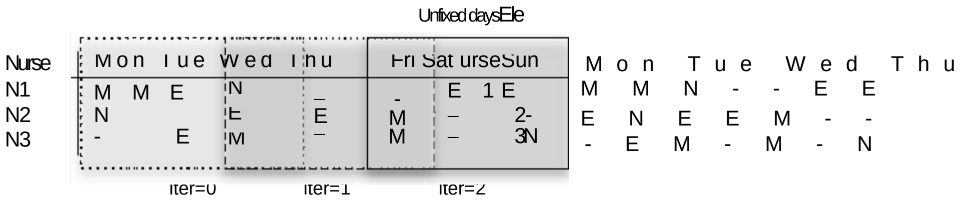

Fig. 3 *Fix Days* neighborhood subproblems with *ndays* = 3 and *step* = 2.

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 is the number of different subproblems in the neighborhood. 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 aren't 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 doesn't 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 on 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:

N_1^m:*Fix Days* neighborhood structure with *ndays* = 2*m* and *step* = *m*

N_2 : *Fix Shifts* neighborhood structure

First, the algorithm performs a complete search on the neighborhoods N_1^m and N_2. After that, the algorithm increases the value of *m* by 1, searching for the best solution on the neighborhood N_{1}^{m+1}. If any improvement is produced by the latter search, the algorithm searches the neighborhood N_2 before incrementing the value of *m*. Otherwise, the algorithm just increases the value of *m* by 1, moving to the neighborhood N_{1}^{m+1}. This procedure repeats 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 $N_1^m(S_0)$ and $N_2(S_0)$ return, respectively, the best neighbors of S_0 in the neighborhoods N_1^m and N_2. If no better solution than S_0 is found, then S_0 is returned.

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 N_1^m almost never improves the solution, but takes considerable processor time. Since neighborhood N_1^m is smaller than N_{1}^{m+1}, and most neighbors of N_1^m are also in the neighborhood N_{1}^{m+1}, looking again for a better solution on 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 $N_{1}^{|D|}$ is the same as solving the original problem itself.

```
Require: S0, m
1: S* ← N1^m(S0)
2: S* ← N2(S*)
3: m = m + 1
4: while m ≤ |D|/2 and time limit not reached do
5: S ← N1^m(S*)
6: if S is a better solution than S* then
7: S* ← N2(S)
8: end if
9: m = m + 1
10: end while
11: return S*;
```

Fig. 4 Pseudocode of MPH Algorithm

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 [24,42,32] and also communicate with commercial solvers through the Open Solver Interface (OSI)[41]. 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].

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

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

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 $s = \varnothing$) 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 ofabout 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 lb and an upper bound ub the gap is $\frac{ub-lb}{ub} \times 100$. 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 $k3$, 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 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.

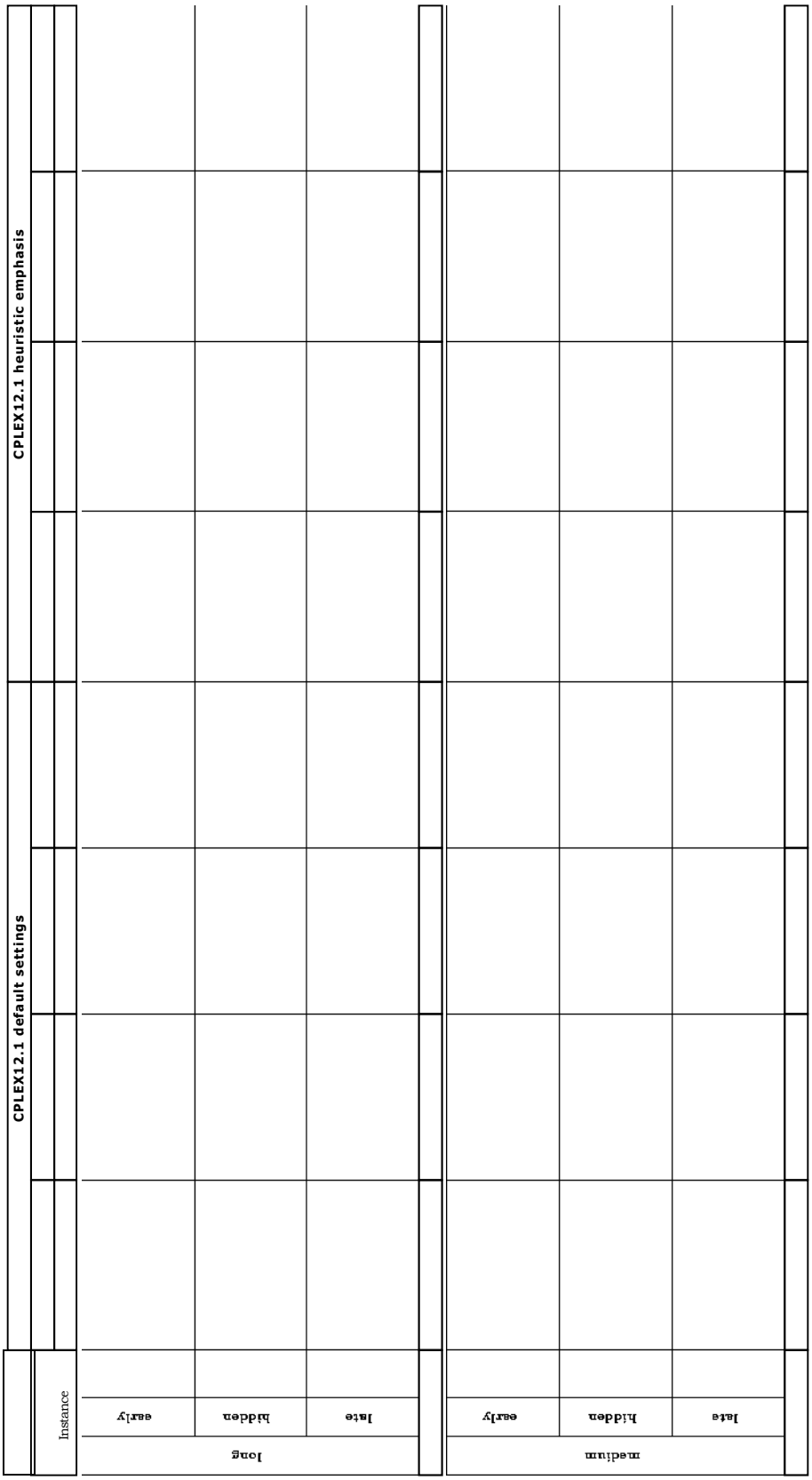

Table 3 Results of the standalone commercial solver CPLEX

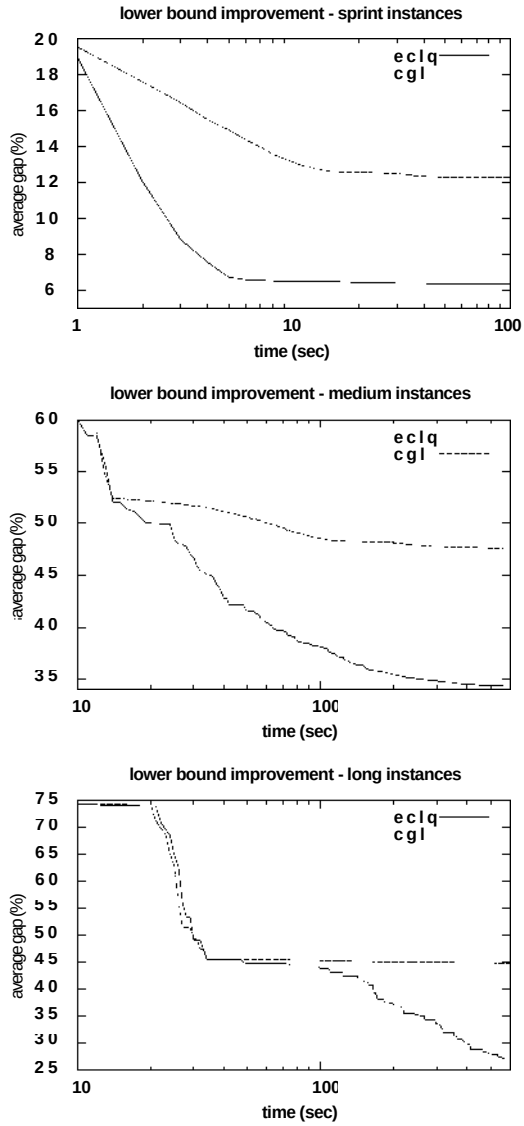

Fig. 5 Dual bound improvement for COIN-OR built in cut generator (`cgl`) and the proposed cut cut separation procedure (`eclq`)

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 ($0/^1_2$)[2] cuts. Zero-Half cuts are not available yet in the latest formal CGL release,

		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

but authors gently 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: $\min\{\frac{lb-lp}{lp+f} \times 100, 100\}$, where f 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): $\frac{ub-BKS}{ub} \times 100$.

The results for “sprint” instances weren’t 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:

instance			best		CPLEX (m=14)		best MMH (m ∈ [1, 9])		MMH results with different values for m													
									m=1		m=2		m=3		m=4		m=5		m=6		m=7		m=8		m=9	
			UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap
long	early	a	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
		b	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0	419	0.0
		d	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0	490	0.0
		e	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0
		f	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	a	346 @346	4.7	487	23.0	346	0.0	347	0.3	348	0.6	346	0.0	346	0.0	339	3.6	346	0.0	339	2.3	334	35.2	306	31.6
		b	89 @89	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
		d	38	0.0	61	37.7	39	2.6	40	5.0	40	5.0	41	7.5	41	7.5	41	7.5	41	7.5	130	70.8	119	68.1	118	67.8
		e	27	0.0	44	30.0	27	0.0	27	0.0	27	0.0	27	0.0	27	0.0	27	0.0	27	0.0	37	40.5	41	40.5	117	80.4
		f	41	0.0	130	68.5	41	0.1	43	4.7	43	4.7	43	4.7	43	4.7	43	4.7	43	4.7	41	0.0	43	10.5	174	76.4
	late	a	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
		b	229	0.0	409	44.0	235	2.6	235	2.6	237	3.4	235	2.6	235	2.6	239	4.2	242	5.4	239	7.9	242	5.4	243	5.8
		d	220	0.0	391	43.7	220	0.0	227	3.1	227	3.1	227	3.1	220	0.0	233	5.6	233	5.6	233	5.6	227	3.1	380	42.1
		e	177	0.0	310	42.7	177	0.0	178	0.6	178	0.6	178	0.6	178	0.6	179	1.1	177	0.0	177	0.0	177	0.0	233	24.0
		f	82	0.0	170	51.8	82	0.0	86	4.7	84	2.3	82	0.0	82	0.0	82	0.0	82	0.0	82	0.0	82	0.0	240	65.8
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	a	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
		b	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
		d	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
		e	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
		f	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0	305	0.0
	hidden	a	111 @111	14.6	314	64.6	111	0.0	116	4.3	111	0.0	116	3.9	116	3.9	116	4.3	117	5.1	118	5.9	120	7.5	307	63.8
		b	84 @84	5.6	332	74.7	77	0.0	77	0.0	77	0.0	77	0.0	77	0.0	77	0.0	77	0.0	77	0.0	77	0.0	229	13.7
		d	78 @78	2.5	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
		e	119 @119	2.5	136	42.6	80	2.5	80	2.5	80	2.5	84	7.1	82	5.1	82	4.8	81	3.7	81	3.7	80	2.5	147	46.9
		f	157 @157	2.5	233	48.9	119	0.0	121	1.7	120	0.8	125	3.3	121	1.7	120	0.8	119	0.0	127	6.3	121	1.7	232	48.7
	late	a	158 @157	0.6	213	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
		b	46 @46	0.0	46	0.0	43	6.5	47	8.5	47	8.5	47	8.5	43	4.7	44	25.0	40	15.0	40	15.0	40	15.0	74	62.0
		d	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
		e	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
		f	107 @107	0.0	120	10.8	105	0.9	105	0.9	114	6.1	113	5.3	116	6.9	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)		(3.1/38)		(3.2/3.7)		(22/3.1)		(15.1/21.3)	

Table 5 Results of the Mathematical Programming Heuristic

– The impact of the parameter m is very perceptible and the best average result were 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 on the literature for 6 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 magnitude in the running times for "long" instances. While *MPH* running times are limited to 600 seconds, Valouxis et al. [44] 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 important to remember that this site has received updates in the years following the competition, so the previous best known solutions (column PUB) were already very hard to find. Most instances were solved to optimality and the harder among the unsolved instances is medium hidden05 where the lower bound distance is now at 23.7%.

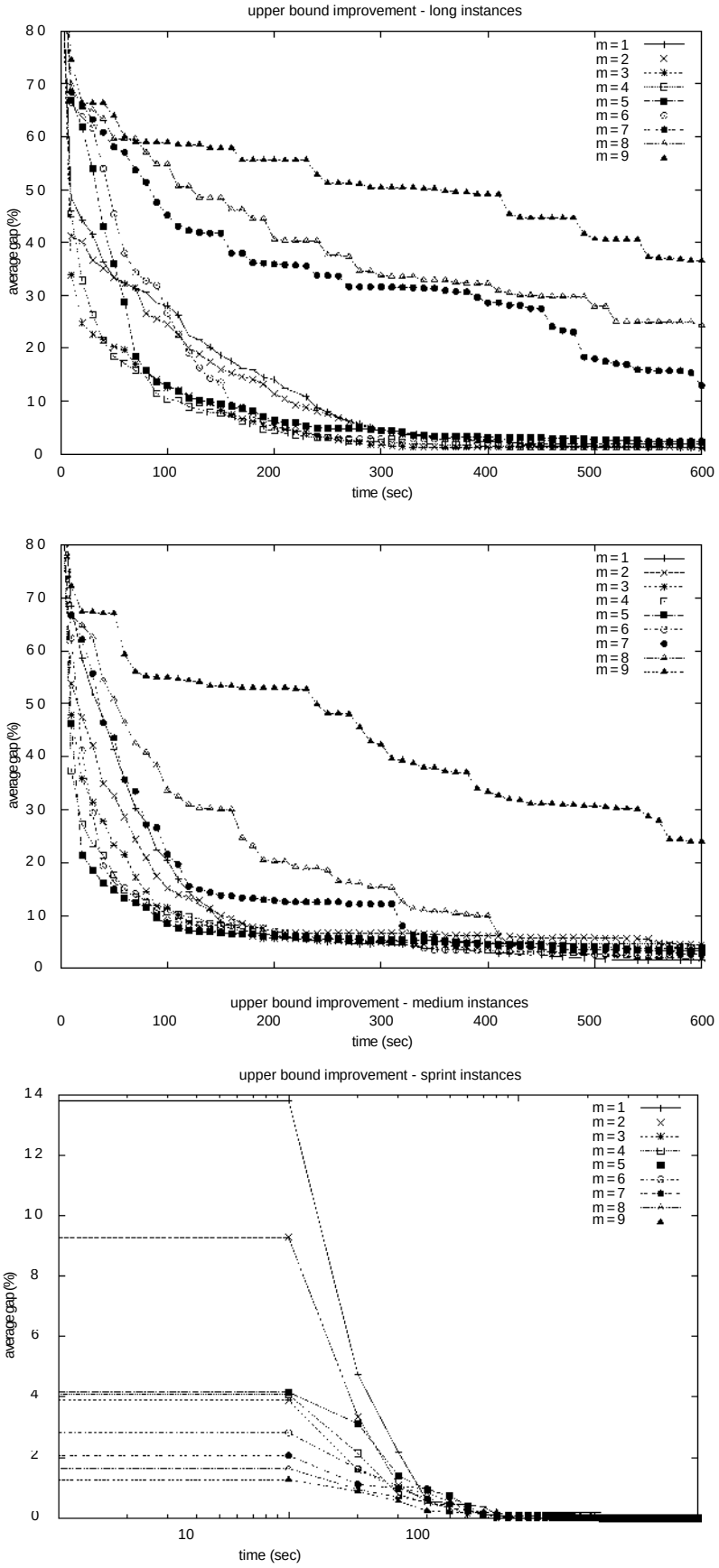

Fig. 6 Improvement in time graphic for MPH with different values for the initial *m*.

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

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 allows a more effective pruning of nodes in the search tree, which is useful to speedup 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 towards 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

The authors would like to thank FAPEMIG (grant APQ-01779-10) and CNPq (grant 480388/2010-5) for supporting the development of this research and the anonymous reviewers of this paper for the detailed suggestions and corrections.

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., Cornueljols, 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: the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT10)-the Nurse Rostering Competition (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 Simpsio Brasileiro de Pesquisa Operacional (2011)
11. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16,** 575–577 (1973). DOI http://doi.acm.org/10.1145/362342.362367. URL http://doi.acm.org/10.1145/362342.362367
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 2010 (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., Mareèek, K., Parkes, A.J., Rudová, H.: A branch-and-cut procedure for the udine course timetabling problem. Ann. Oper. Res. pp. 1–17 (2011). DOI http://dx.doi.org/10.1007/s10479-010-0828-5. URL http://cs.nott.ac.uk/~jxm/timetabling/patat2008-paper.pdf
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 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). DOI 10.1287
25. Grotschel, M., Lovasz, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer (1993)
26. Hansen, P., Mladenovi´c, N.: Variable neighborhood search. Computers and Operations Research 24(11), 1097–1100 (1997)
27. Hansen, P., Mladenovi´c, N., Urosevi´c, D.: Variable neighborhood search and local branching. Comput. Oper. Res. 33(10), 3034–3045 (2006)
28. Haspeslagh, S., De Causmaecker, P., Stolevik, M., A., S.: First international nurse roster-ing competition 2010. Tech. rep., CODeS, Department of Computer Science, KULeuven 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., Sal-vagnin, D., Steffy, D., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation 3, 103–163 (2011). URL http://dx.doi.org/10.1007/s12532-011-0025-9. 10.1007/s12532-011-0025-9
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). DOI 10.1007/s10732-008-9079-x. URL http://dl.acm.org/citation.cfm?id=1527562.1527566
35. Méndez-D´ıaz, 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. Poggi de Aragão, M., Uchoa, E.: Integer program reformulation for robust branch-and-cut-and-price. In: Annals of Mathematical Programming in Rio, pp. 56–61. Buzios, Brazil (2003)

41. 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
42. 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)
43. 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). DOI 10.1002/net.20485. URL http://dx.doi.org/10.1002/net.20485
44. 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)