Adaptive Neighborhood Search for Nurse Rostering

Zhipeng Lü[a,b], Jin-Kao Hao[b,*]

European Journal of Operational Research 218(3): 865-876, 2012

[a] *School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, P.R.China*
[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

Abstract

This paper presents an adaptive neighborhood search method (*ANS*) for solving the nurse rostering problem proposed for the First International Nurse Rostering Competition (*INRC-2010*). *ANS* uses jointly two distinct neighborhood moves and adaptively switches among three intensification and diversification search strategies according to the search history. Computational results assessed on the three sets of 60 competition instances show that *ANS* improves the best known results for 12 instances while matching the best bounds for 39 other instances. An analysis of some key elements of *ANS* sheds light on the understanding of the behavior of the proposed algorithm.

Keywords: Nurse rostering; intensification and diversification; adaptive switching mechanism; tabu search

1. Introduction

Nurse rostering is a research topic of increasing interest in recent decades that is encountered by many large modern hospitals around the world [23]. As a specific personnel scheduling problem, nurse rostering problem consists in generating daily schedules for nurses by assigning a number of daily demanding shifts to nurses with different skills subject to certain predefined (hard and soft) constraints. The general objective of the problem is to effectively utilize limited resources such that the hospitals' efficiency can be

*Corresponding author.
Email addresses: zhipeng.lv@hust.edu.cn, zhipeng.lui@gmail.com (Zhipeng Lü), hao@info.univ-angers.fr (Jin-Kao Hao)

improved without sacrificing the well-being and job satisfaction of nurses [26].

Due to the presence of many constraints and requirements of conflicting nature, nurse rostering in the real world are often complex and difficult to solve and present a great challenge for researchers in universities and personnel managers in hospitals. As described in [19], nurse rostering must consider issues like coverage demand, workload of nurses, consecutive assignments of shifts, day-off/on requirements, weekend-related requirements, preference or avoidance of certain shift patterns, etc.

Over the last few decades, nurse rostering has been extensively studied and a wide range of effective approaches have been reported in the literature. These techniques can be roughly classified into two main categories: exact algorithms and heuristics. Among the first category are several methods using mathematical programming techniques [6, 7, 20]. However, the high computational complexity of nurse rostering problems limits the application of exact methods only to small size instances. For larger instances, various effective metaheuristic algorithms have been designed to find suboptimal solutions of good quality in a reasonable time.

Burke *et al* [16] developed two hybrid tabu search algorithms, respectively with diversification and greedy shuffling heuristics, where several neighborhoods were defined: Moving a shift from one nurse to another on the same day, exchanging a part of the schedule of nurses and moves for exchanging assignments among every pair of nurses. Bellanti *et al* [8] introduced a tabu search procedure and an iterated local search for tackling a nurse rostering problem with various hard and soft constraints. They used four different neighborhoods operating on partial solutions completed by means of a greedy procedure so as to avoid the generation of infeasible solutions. In [15], Burke *et al* applied a Variable Neighborhood Search (*VNS*) on highly constrained real world nurse rostering data and they observed that *VNS* could help the search to effectively jump out of the local optima. Burke *et al* [13] proposed a hybrid heuristic ordering and variable neighborhood search method by combining heuristic ordering, *VNS* and back-tracking. The *VNS* is based on two types of neighborhood moves, which respectively assign a shift to a different nurse and swap the nurses assigned to each of a pair of shifts. The proposed algorithm significantly outperforms an existing genetic algorithm on commercial data. Valouxis and Housos [29] applied an approximate integer linear programming model to generate the initial solution of their nurse rostering problem and then further optimize it using a '2-opt' neighborhood local search procedure. Other representative approaches to solving nurse rostering problems also include simulated annealing [10], genetic algorithms [1, 2, 22], scatter search [14], memetic method [11], evolutionary algorithm [5] and estimation and distribution algorithm [3]. Interested readers are referred to [17] for a comprehensive survey of the advanced approaches for nurse rostering presented in recent decades.

The study reported in this paper concerns the nurse rostering problem recently presented in the First International Nurse Rostering Competition (*INRC-2010*). Building on the previous two timetabling competitions—*ITC-2002* and *ITC-2007* [24]—*INRC-2010* competition aims to further develop interest in timetabling and rostering by providing more challenging problems with an increased number of real world constraints. Moreover, the *INRC-2010* nurse rostering problem integrates additional real world constraints that were also missed in the previous nurse rostering literatures [19]. For this challenging problem, a number of solution procedures have been proposed by the participants of the competition. We now briefly review the methods proposed by the *INRC-2010* competition finalists.

Valouxis *et al* [28] tackled the problem by partitioning the original problem into sub-problems. Each sub-problem size is solved using mathematical programming. The approach consists of two phases: One is to assign nurses to working days and the other is to schedule the nurses assigned to each day to certain shifts. For the *Medium* and *Long* tracks of the competition, three additional local search techniques were incorporated into the first phase. It is noteworthy that this algorithm won all the three tracks of the *INRC-2010* competition. Burke and Curtois [12] applied two algorithms to solve the problem: The first algorithm is an ejection chain based method and it was applied to the *Sprint* instances. The second algorithm is a branch and price method which was applied to the *Medium* and *Long* instances. It has been shown that the second algorithm was generally able to solve many of the competition instances to optimality within the competition time limit. This algorithm was ranked the 2nd place for the *Medium* and *Long* tracks and the 4th place for the *Sprint* track. Nonobe [27] first modeled the problem into a constraint optimization problem (COP) and then used a general-purpose COP solver based on tabu search to solve it. The algorithm got the 2nd, 3rd and 4th places for the *Sprint*, *Medium* and *Long* tracks, respectively. Bilgin *et al* [9] proposed a hybrid algorithm which employs a hyper-heuristic followed by a greedy shuffle heuristic. In addition, the authors provided the computational results of integer linear programming (ILP) using IBM CPLEX. They got one 3rd place for the *Long* track and two 5th places for the *Sprint* and *Medium* tracks of the competition.

In this paper, we present ANS, an adaptive neighborhood search algorithm for solving the nurse rostering problem of the *INRC-2010*. Our ANS

algorithm incorporates an adaptive search mechanism which automatically switches among three search strategies, respectively called *intensive search*, *intermediate search* and *diversification search*. As such, the tradeoff between intensification and diversification is achieved in a flexible manner. The main contribution of the proposed algorithm can be summarized as follows:

- Compared with the top-ranked solvers of the *INRC-2010* competition like [9, 12, 28], the proposed algorithm, which is a pure neighborhood heuristic, remains conceptually simple. Indeed, while the reference solvers often apply *hybrid* methods (ILP, branch-and-price and heuristics) to tackle different tracks of the competition, our solver is based on a unified local search algorithm which is applied to solve all the competition instances.

- The proposed algorithm achieves a good performance by improving the previous best known results for 12 instances while matching the best known solutions in 39 other cases.

The remaining part of the paper is organized as follows. Section 2 presents the problem description and a mathematical formulation of the nurse rostering problem addressed in this paper. In Section 3, the main idea, framework and each component of our *ANS* algorithm for solving the nurse rostering problem are described. Sections 4 is dedicated to the computational results under both competition and relaxed timeout conditions. Section 5 investigates several essential components of the proposed *ANS* algorithm and concluding remarks are given in Section 6.

2. Problem Formulation of Nurse Rostering Problem

The nurse rostering problem considered in this paper consists of assigning shifts to nurses in accordance with a given set of constraints [19]. Usually, two types of constraints are defined: Those which must be strictly satisfied under any circumstances (hard constraints) and those which are not necessarily satisfied but whose violations should be desirably minimized (soft constraints). A schedule that satisfies all the hard constraints is called a *feasible* assignment. The objective of the nurse rostering problem is to minimize the total weighted soft constraint violations in a feasible assignment. Interested readers are referred to [19] for a detailed problem description of the *INRC-2010* problem .

We present below a mathematical formulation of the problem which is missing in the literature.

To introduce the hard and soft constraints, we state:

- a set $\mathcal{D}$ of days, during which nurses are to be scheduled, $|\mathcal{D}| = D$. Usually $\mathcal{D}$ is composed of four weeks, i.e., $D = 28$;
- a set $\mathcal{S}$ of nurses, each being associated with a set of available skills and working under exactly one contract, $|\mathcal{S}| = S$;
- a set $\mathcal{H}$ of shifts, each being characterized by a set of required skills, $|\mathcal{H}| = H$;
- a set $\mathcal{P}$ of patterns, each being the shift series that the nurse may not want to work in a row, $|\mathcal{P}| = P$;
- a set $\mathcal{C}$ of contracts, each being characterized by a number of regulations that should be respected by all the nurses working under this contract, $|\mathcal{C}| = C$.

We choose a direct solution representation for simplicity reasons. A candidate solution is represented by an $S \times D$ matrix $\mathcal{X}$ where $x_{i,j}$ corresponds to the shift type assigned for nurse s_i at day d_j. If there is no shift assigned to nurse s_i at day d_j, then $x_{i,j}$ takes the value "-1". With this representation we ensure that a nurse can only work at most one shift per day, meaning that the second hard constraint H_2 will never be violated.

A number of constant and variable symbols are presented in Table 1, where the constants are predefined by the problem instance or regulated by the working contract of nurses while the variables may take different values according to the current solution $\mathcal{X}$. The second column indicates in which constraint the corresponding constants or variables occur.

The 2 hard constraints are:

- H_1. **Coverage requirement**: For each day all demanded shifts must be assigned to nurses. $\forall d \in \mathcal{D}, h \in \mathcal{H}$,

$$\sum_{s=1}^{S} \chi(x_{s,d} = h) = sc(d, h)$$

  where χ is the truth indicator function which takes values of 1 if the given proposition is true and 0 otherwise.

- H_2. **Single shift per day**: A nurse can only work one shift per day, i.e., no two shifts can be assigned to the same nurse on a day. This hard constraint is always satisfied using our solution representation.

For any nurse $s \in \mathcal{S}$, the 18 soft constraints are:

- S_1. **Maximum assignment**: The maximum number of shifts that can be assigned to nurse s.

$$f_{s,1} = max(\sum_{d \in \mathcal{D}} \chi(x_{s,d} \neq -1) - shift(s)^+, 0)$$

- S_2. **Minimum assignment**: The minimum number of shifts that can be assigned to nurse s.

$$f_{s,2} = max(shift(s)^- - \sum_{d \in \mathcal{D}} \chi(x_{s,i} \neq -1), 0)$$

- S_3. **Maximum consecutive working days**: The maximum number of consecutive days on which a shift has been assigned to nurse s.

$$f_{s,3} = \sum_{i=1}^{n\_wksect(s)} max(len\_wksect(s,i) - work(s)^+, 0)$$

- S_4. **Minimum consecutive working days**: The minimum number of consecutive days on which a shift has been assigned to nurse s.

$$f_{s,4} = \sum_{i=1}^{n\_wksect(s)} max(work(s)^- - len\_wksect(s,i), 0)$$

- S_5. **Maximum consecutive free days**: The maximum number of consecutive days on which nurse s has no shift assigned.

$$f_{s,5} = \sum_{i=1}^{n\_frsect(s)} max(len\_frsect(s,i) - free(s)^+, 0)$$

- S_6. **Minimum consecutive free days**: The minimum number of consecutive days on which nurse s has no shift assigned

$$f_{s,6} = \sum_{i=1}^{n\_frsect(s)} max(free(s)^- - len\_frsect(s,i), 0)$$

- S$_7$. **Two free days after a night shift**: Nurse s should not be assigned any shift except a night shift during the following two days after a night shift.

$$f_{s,7} = \begin{array}{l} \sum_{i=1}^{D-2} \chi(x_{s,i} = Night \wedge ((x_{s,i+1} \neq Night \wedge x_{s,i+1} \neq -1) \vee (x_{s,i+2} \neq Night \wedge x_{s,i+2} \neq -1))) \\ +\chi(x_{s,D-1} = Night \wedge x_{s,D} \neq Night \wedge x_{s,D} \neq -1) \end{array}$$

- S$_8$. **Maximum consecutive working weekends**: The maximum number of consecutive weekends on which at leat one shift is assigned to nurse s.

$$f_{s,8} = \sum_{i=1}^{n\_wkdsect(s)} max(len\_wkdsect(s,i) - wkd(s)^{+}, 0)$$

- S$_9$. **Minimum consecutive working weekends**: The minimum number of consecutive weekends on which at leat one shift is assigned to nurse s.

$$f_{s,9} = \sum_{i=1}^{n\_wkdsect(s)} max(wkd(s)^{-} - len\_wkdsect(s,i), 0)$$

- S$_{10}$. **Maximum number of working weekends**: The maximum number of weekends in four weeks in which at least one shift is assigned to nurse s.

$$f_{s,10} = max(\sum_{i=1}^{n\_wkd} \chi(nwd(s,i) > 0) - n\_wkd(s)^{+}, 0)$$

- S$_{11}$. **Complete weekends**: Nurse s should work on all days of a weekend if nurse s works at least one day of the weekend.

$$f_{s,11} = \sum_{i=1}^{n\_wkd} CompWkdCost(s,i)$$

where

$$CompWkdCost(s,i) = \begin{cases} 4, & \text{if } nd(s) = 3 \wedge nwd(s,i) = 2 \wedge h\_wkd(s,i,2) = -1; \\ nd(s) - nwd(s,i), & \text{else if } 0 < nwd(s,i) < nd(s); \\ 0, & \text{otherwise.} \end{cases}$$

The first condition indicates that a higher penalty is raised if the working days at the weekend are not consecutive (this may happen when the weekend has 3 days according to the instance definition), i.e., if the working pattern of the weekend is X0X (X=working, 0=not working), the cost is equal to 4. In this case, only the patterns 000 and XXX do not incur any violation of this constraint. Note that the working pattern of a weekend is the shift series that the nurse works at a weekend.

- S_{12}. **Identical complete weekend shift type**: Nurse s should work the same shift types on the days of a complete working weekend.

$$f_{s,12} = \sum_{i=1}^{n\_wkd} IdentWkdCost(s,i)$$

where

$$IdentWkdCost(s,i) = \begin{cases} \sum_{h \in \mathcal{H}, nh(s,i,h)>0} (nd(s) - nh(s,i,h)), & \text{if } nwd(s,i) = nd(s); \\ 0, & \text{otherwise.} \end{cases}$$

- S_{13}. **Requested day on**: Nurse s requests to work on a specific day.

$$f_{s,13} = \sum_{d \in \mathcal{D}} \chi(day\_req(s,d) = on \wedge x_{s,d} = -1)$$

- S_{14}. **Requested day off**: Nurse s requests not to work on a specific day.

$$f_{s,14} = \sum_{d \in \mathcal{D}} \chi(day\_req(s,d) = off \wedge x_{s,d} \neq -1)$$

- S_{15}. **Requested shift on**: Nurse s requests to work a specific shift on a specific day.

$$f_{s,15} = \sum_{d \in \mathcal{D}} \sum_{h \in \mathcal{H}} \chi(sh\_req(s,d,h) = on \wedge x_{s,d} \neq h)$$

- S_{16}. **Requested shift off**: Nurse s requests not to work a specific shift on a specific day.

$$f_{s,16} = \sum_{d \in \mathcal{D}} \sum_{h \in \mathcal{H}} \chi(sh\_req(s,d,h) = off \wedge x_{s,d} = h)$$

- S_{17}. **Alternative skill**: Nurse s should work a shift for which all the required skills of the shift are possessed by nurse s

$$f_{s,17} = \sum_{d \in \mathcal{D}} \chi(x_{s,d} \neq -1 \wedge qual(s, x_{s,d}) = false)$$

- S_{18}. **Unwanted shift patterns**: Nurse s should not work a specific unwanted pattern in a row.

$$f_{s,18} = \sum_{p \in unwantp(s)} n\_unwp(s,p)$$

With the above formulation, we can then calculate the total soft constraint cost for a given candidate feasible solution $\mathcal{X}$ according to the cost function $f(\mathcal{X})$ defined in Formula (1).

$$f(\mathcal{X}) = \sum_{s=1}^{S} \sum_{i=1}^{18} w_{s,i} \cdot f_{s,i} \tag{1}$$

where $w_{s,i}$ is the weight associated to the soft constraint S_i for nurse s, regulated by the contract of nurse s. Note that different weights may be assigned to different soft constraints in an attempt to produce solutions that are more appropriate for their particular needs. $w_{s,i}$ could be 0 if the corresponding soft constraint is not considered. The objective is then to find a feasible solution $\mathcal{X}^*$ such that $f(\mathcal{X}^*) \leq f(\mathcal{X})$ for all $\mathcal{X}$ in the feasible search space.

3. Adaptive Neighborhood Search Algorithm

3.1. Main Framework

Starting from an initial feasible solution generated by a constructive heuristic (see Section 3.2), our *ANS* algorithm (Algorithm 1), which adaptively switches among three search strategies according to a diversification level *dl*, is used to optimize the solution as far as possible until the solution cannot be improved within a certain number of iterations (lines 7-15). When the local search stops, the search is restarted from an elite solution, whereupon a new round of adaptive local search is again launched (lines 19-20). In the following subsections, the main components of our *ANS* algorithm are described in detail.

Algorithm 1 Pseudo-code of the *ANS* algorithm for nurse rostering

1: **Input**: Problem instance I
2: **Output**: The best roster assignment $\mathcal{X}^*$ obtained
3: $\mathcal{X}^0 \leftarrow$ Initial_Solution( ) (see Section 3.2)
4: $\mathcal{X}^* \leftarrow \mathcal{X}^0$; diversification level parameter $dl \leftarrow 0.0$
5: **repeat**
6:     $\mathcal{X} \leftarrow \mathcal{X}^0$; $\mathcal{X}' \leftarrow \mathcal{X}^0$ //$\mathcal{X}$ and $\mathcal{X}^{'}$ denote the current and the best solution in the current round of local search, respectively
    //Lines 7-15: local search procedure
7:     **while** *the improvement cutoff of local search is not reached* **do**
8:         $mv \leftarrow$ neighborhood move selected from $M(\mathcal{X})$ (see Alg. 2, Section 3.4)
9:         $\mathcal{X} \leftarrow \mathcal{X} \oplus mv$
10:         **if** $f(\mathcal{X}) < f(\mathcal{X}^{'})$ **then**
11:             $\mathcal{X}^{'} \leftarrow \mathcal{X}$
12:         **end if**
13:         $dl \leftarrow Parameter\_Updating(dl)$ (see Alg. 3, Section 3.5)
14:     **end while**
    //Lines 16-18: record the best solution $\mathcal{X}^*$ found so far
15:     **if** $f(\mathcal{X}^{'}) < f(\mathcal{X}^*)$ **then**
16:         $\mathcal{X}^* \leftarrow \mathcal{X}^{'}$
17:     **end if**
    //Lines 19-20: restart the search from an elite solution (see Section 3.6)
18:     $\mathcal{X}^0 \leftarrow \mathcal{X}^*$ or $\mathcal{X}^0 \leftarrow \mathcal{X}^{'}$ with equal probability
19:     $dl \leftarrow 1.0$
20: **until** a stop criterion is met

3.2. Initial Solution

Our *ANS* algorithm generates a feasible initial solution satisfying the two hard constraints (H_1 and H_2). As mentioned above, the second hard constraint H_2 is automatically satisfied with our solution representation. Thus, we consider only the first hard constraint H_1, i.e., the daily shift coverage requirement. This is achieved by a sequential heuristic starting from an empty roster, from which roster assignments are constructed by inserting one appropriate shift into the roster at each time. We repeat this procedure for $\sum_{d \in \mathcal{D}} \sum_{h \in \mathcal{H}} sc(d,h)$ times until all the daily shift coverage requirements are met.

At the beginning, the unassigned shift h of day d is equal to $sc(d,h)$. At each step, we carry out two distinct operations: One is to randomly select an unassigned shift h' of a specific day d' where $sc(d',h') > 0$, the other is to randomly choose a nurse who is free for this shift. After this, $sc(d',h')$ is decreased by one. This procedure repeats until $sc(d,h)$ becomes zero for any shift h and any day d. In this way, the first hard constraint

H_1 is guaranteed to be satisfied and a feasible roster is constructed. Let us mention that our algorithm does not consider any soft constraint in this initial solution generation procedure. Our experiments demonstrated that the quality of the initial solution has little influence on the performance of our *ANS* algorithm.

3.3. Moves and Neighborhood

Given a solution $\mathcal{X}$, a neighboring solution can be obtained by applying a move mv to $\mathcal{X}$, denoted by $\mathcal{X} \oplus mv$. Let $M(\mathcal{X})$ be the set of moves which can be applied to $\mathcal{X}$, then the neighborhood of $\mathcal{X}$ is defined by:

$$N(\mathcal{X}) = \{\mathcal{X} \oplus mv | mv \in M(\mathcal{X})\} \tag{2}$$

In this paper, we use a combined neighborhood jointly defined by moving one shift at a specific day to a different nurse (*One-Move*) and swapping the two shifts assigned to a pair of nurses at a specific day (*Two-Swap*). Notice that these two moves never break the feasibility of the solutions.

Specifically, a *One-Move* in solution $\mathcal{X}$, denoted by $mv_1(d, s_1, s_2)$, consists in assigning the value of $x_{s_1,d}$ to $x_{s_2,d}$, i.e., $x_{s_2,d} = x_{s_1,d}$ and $x_{s_1,d} = -1$. Formally,

$$M_1(\mathcal{X}) = \{mv_1(d, s_1, s_2) | \forall d \in \mathcal{D}, x_{s_1,d} \neq -1 \wedge x_{s_2,d} = -1\} \tag{3}$$

Applying the *Two-Swap* move, denoted by $mv_2(d, s_1, s_2)$, to two different shifts $x_{s_1,d}$ and $x_{s_2,d}$ at day d in solution $\mathcal{X}$ consists in assigning the value of $x_{s_1,d}$ to $x_{s_2,d}$ and inversely the value of $x_{s_2,d}$ to $x_{s_1,d}$. Formally,

$$M_2(\mathcal{X}) = \{mv_2(d, s_1, s_2) | \forall d \in \mathcal{D}, x_{s_1,d}, x_{s_2,d} \neq -1 \wedge x_{s_1,d} \neq x_{s_2,d}\} \tag{4}$$

In our *ANS* algorithm, a combination of both $M_1(\mathcal{X})$ and $M_2(\mathcal{X})$ moves is used. At each local search iteration, $M_1(\mathcal{X})$ is applied with probability q, while $M_2(\mathcal{X})$ is employed at a (1 - q) rate. In this paper, we empirically set $q = 1 - \varphi \cdot dens$, where $\varphi = 0.4$ and $dens = \frac{\sum_{d \in \mathcal{D}} \sum_{h \in \mathcal{H}} sc(d,h)}{S \cdot D} \times 100\%$ represents the density of the problem instance. This combined neighborhood $M(\mathcal{X})$ is defined in Eq. (5), where $r[0, 1)$ represents a random number between 0 and 1.

$$M(\mathcal{X}) = \begin{cases} M_1(\mathcal{X}), & \text{if } r[0,1) < q; \\ M_2(\mathcal{X}), & \text{otherwise.} \end{cases} \tag{5}$$

3.4. Move Selection Strategies

To explore the above combined neighborhood, we introduce three move selection strategies leading respectively to an *Intensive Search*, an *Intermediate Search* and a *Diversification Search*. The *ANS* algorithm uses an adaptive mechanism to switch among them such that a suitable exploitation/exploration balance is reached.

Intensive Search:

In this search strategy, we employ a tabu search (TS) algorithm to explore the whole neighborhood $M(\mathcal{X})$. TS typically incorporates a *tabu list* as a "recency-based" memory structure to forbid each performed move to be reconsidered within a certain span of iterations (tabu tenure [21]).

More precisely, when using move $mv_1(d, s_1, s_2) \in M_1(\mathcal{X})$, if the shift $x_{s_1,d}$ is moved from nurse s_1 to nurse s_2, then reassigning shift $x_{s_1,d}$ to nurse s_1 at day d is declared tabu and thus forbidden. On the other hand, when it comes to move $mv_2(d, s_1, s_2) \in M_2(\mathcal{X})$, if the two shifts $x_{s_1,d}$ and $x_{s_2,d}$ are exchanged by nurses s_1 and s_2 at day d, it is forbidden to reassign shift $x_{s_1,d}$ ($x_{s_2,d}$) back to nurse s_1 (s_2) at day d within the next $TabuTenure$ iterations. In our experiments, we set the tabu tenure as: TabuTenure=tl + rand(3) where *tl* is a given constant and *rand*(3) takes a random value from 1 to 3. We empirically set $tl = \lfloor 0.8 \cdot S \rfloor$ for all the tested instances.

At each iteration, our TS algorithm then restricts consideration to moves which are not forbidden by the tabu list, and selects a move that produces the largest improvement in terms of the objective $f(\mathcal{X})$, breaking ties randomly. Together with this rule, a simple aspiration criterion is applied which allows a move to be performed in spite of being tabu if it leads to a solution better than the current best solution.

Intermediate Search:

As mentioned above, *Intensive Search* systematically chooses the best move among *all the feasible moves* in the neighborhood, while our *Intermediate Search* picks a move from those limited to a subset of nurses. Specifically, at each iteration our *ANS* algorithm randomly selects a subset $\mathcal{S}^*$ of nurses ($\mathcal{S}^* \subseteq \mathcal{S}$) and all the moves concerning the nurses in $\mathcal{S}^*$ are considered. That is to say, we only take into account a subset of the whole neighborhood $M(\mathcal{X})$ defined by Eq. (5), represented by

$$M(\mathcal{X})(\mathcal{S}^*) = \{mv(d, i, j) | mv(d, i, j) \in M(\mathcal{X}) \wedge i \in \mathcal{S}^* \wedge j \in \mathcal{S}^*\} \quad (6)$$

where we empirically set $|\mathcal{S}^*| = \lfloor S/2 \rfloor$ in our experiments.[1]

[1] This is the main difference to the *INRC-2010* competition version of our solver where

At each iteration, we select the best move in $M(\mathcal{X})(\mathcal{S}^*)$ to perform. However, if the best move is the most recently visited move, we select the second best move (in terms of solution quality) with a probability 0.5. This strategy of selecting the second best move is borrowed from SAT solvers [4] and it can help the search to avoid traversing already visited search regions and improve the search robustness to some extent. In our implementation, we utilize a memory structure called *recency* to record the iteration at which a move is recently performed. More precisely, each time a shift type h is moved away from nurse s at day d, the current local search iteration index (the *iter* number in Algorithm 1) is assigned to the associated record *recency(s,d,h)*. Thus, we could easily identify whether the best move in $M(\mathcal{X})(\mathcal{S}^*)$ is the recently performed one by retrieving the value of *recency*. This strategy is used to increase the diversification of the algorithm when the best move in $M(\mathcal{X})(\mathcal{S}^*)$ cancels a recent move. Let us comment that the *Intermediate Search* can be considered as a search strategy lying between the *Intensive Search* and the *Diversification Search* described below.

Diversification Search:

The objective of this search strategy is to diversify the search when a stagnation behavior is detected. Similar to the *Intermediate Search*, at each iteration a subset $\mathcal{S}^*$ of nurses are randomly selected and all the moves concerning these nurses are considered. In other words, from the neighborhood defined in Eq. (7), we identify a subset of promising moves $M'(\mathcal{X})(\mathcal{S}^*)$ of $M(\mathcal{X})(\mathcal{S}^*)$ such that each move in $M'(\mathcal{X})(\mathcal{S}^*)$ can improve at least one of the soft constraints S_1 to S_{18}.

$$M'(\mathcal{X})(\mathcal{S}^*) = \{mv | mv \in M(\mathcal{X})(\mathcal{S}^*) \wedge \exists j, \Delta f_j(mv) < 0\} \tag{7}$$

where $\Delta f_j(mv)$ denotes the objective difference for the jth soft constraint S_j incurred by the move mv. We call a move in $M'(\mathcal{X})(\mathcal{S}^*)$ the sub-promising move, i.e., this kind of move improves at least one of the 18 soft constraints. If there exist such sub-promising moves for the subset $\mathcal{S}^*$ of nurses, i.e., $|M'(\mathcal{X})(\mathcal{S}^*)| > 0$, our algorithm randomly selects one of such moves to perform. Otherwise, our algorithm picks a move from $M(\mathcal{X})(\mathcal{S}^*)$ at random.

Given the three search strategies with different intensification and diversification capability, we choose one of these three search strategies according to a parameter *dl* called "diversification level". This parameter is dynam-

---

we set $|\mathcal{S}^*| = 2$. Our experiments show that the *Intermediate Search* with $|\mathcal{S}^*| = 2$ has much less intensification capability than $|\mathcal{S}^*| = \lfloor S/2 \rfloor$. In addition, we employ somewhat different parameter settings in the current version.

ically adjusted to allow the search to adaptively switch among the three search strategies, as described in Algorithm 2. Specifically, if $dl \in [0, \beta_1)$ meaning a strong exploitation is needed, we perform the *Intensive Search* (lines 3-9), while the *Intermediate Search* is employed if $dl \in [\beta_1, \beta_2)$ (lines 10-17) ($0 < \beta_1 < \beta_2 < 1$). Otherwise ($dl \in [\beta_2, 1)$), the *Diversification Search* is used (lines 18-27). In our experiments, we empirically set $\beta_1 = 0.3$ and $\beta_2 = 0.7$ for all the benchmark instances.

Algorithm 2 Neighborhood move selection for *ANS* (dl)

1: **Input**: Current solution $\mathcal{X}$ and feasible moves $M(\mathcal{X})$, diversification level dl
2: **Output**: The selected neighborhood move mv
3: **if** $dl \in [0, \beta_1)$ **then**
4:     **if** *the TS aspiration criterion is satisfied* **then**
5:         $mv \leftarrow$ the best move in $M(\mathcal{X})$
6:     **else**
7:         $mv \leftarrow$ the best move in $M(\mathcal{X})$ except those forbidden by the tabu list
8:     **end if**
9: **end if**
10: **if** $dl \in [\beta_1, \beta_2)$ **then**
11:     Randomly choose a subset $\mathcal{S}^*$ of nurses ($|\mathcal{S}^*| = \lfloor S/2 \rfloor$)
12:     **if** *the best move in* $M(\mathcal{X})(\mathcal{S}^*)$ *is the most recent and* $rand[0, 1) < 0.5$ **then**
13:         $mv \leftarrow$ the second best move in $M(\mathcal{X})(\mathcal{S}^*)$
14:     **else**
15:         $mv \leftarrow$ the best move in $M(\mathcal{X})(\mathcal{S}^*)$
16:     **end if**
17: **end if**
18: **if** $dl \in [\beta_2, 1)$ **then**
19:     Randomly choose a subset $\mathcal{S}^*$ of nurses ($|\mathcal{S}^*| = \lfloor S/2 \rfloor$)
20:     Identify the set $M'(\mathcal{X})(\mathcal{S}^*)$ of sub-promising moves in $M(\mathcal{X})(\mathcal{S}^*)$
21:     **if** $|M'(\mathcal{X})(\mathcal{S}^*)| > 0$ **then**
22:         $mv \leftarrow$ a randomly selected move in $M'(\mathcal{X})(\mathcal{S}^*)$
23:     **end if**
24:     **if** $|M'(\mathcal{X})(\mathcal{S}^*)| = 0$ **then**
25:         $mv \leftarrow$ a randomly selected move in $M(\mathcal{X})(\mathcal{S}^*)$
26:     **end if**
27: **end if**
28: **return** mv

3.5. Adaptive Diversification Level Adjustment

ANS employs a mechanism firstly proposed in [4] to adaptively adjust the diversification level dl according to the search history. dl is first set at a level low enough ($dl = 0.0$) such that the objective function can be quickly improved. When the search process cannot improve the solution quality during a given number of iterations, dl is increased to reinforce the diversification until the search process overcomes the stagnation. Meanwhile, the diversification level is gradually decreased when the search begins to improve the current objective value (Algorithm 3).

Specifically, we record at each adaptive step the current iteration number *iter* and the objective value of the current solution. Then, if this objective value is not improved over the last θ steps (empirically set $\theta = \lfloor \frac{S \cdot H}{10} \rfloor$ for all our experiments), the search is supposed to be stagnating (line 4). At this point, the diversification level parameter dl is *increased* (line 5). Similarly, dl is kept unchanged until another stagnation situation is detected or the objective value is improved (line 8). In the latter case, dl is *decreased* (line 9). Note that the values 6 and 10 in lines 5 and 9 are directly borrowed from [4] and it is observed that they are very appropriate for our solver too.

Algorithm 3 Adaptive adjustment mechanism for diversification level dl

1: $iter = 0$; $adap\_iter = 0$;
2: **repeat**
3:    $iter \leftarrow iter + 1$
4:    **if** $iter - adap\_iter > \theta$ **then**
5:       $dl \leftarrow dl + (1 - dl)/6$
6:       $adap\_f = f$; $adap\_iter = iter$
7:    **else**
8:       **if** $f < adap\_f$ **then**
9:          $dl \leftarrow dl - dl/10$
10:          $adap\_f = f$; $adap\_iter = iter$
11:       **end if**
12:    **end if**
13: **until** Stop condition is satisfied

3.6. Elite Solution Restarting Mechanism

When the current local search cannot improve the solution quality within a given number of iterations, we employ an elite based restart mechanism to diversify the search. Precisely, we restart our local search either from the best solution found so far ($\mathcal{X}^*$) or the best solution of the current round of local search ($\mathcal{X}'$) (lines 18-19 in Algorithm 1). The purpose of alternating between $\mathcal{X}^*$ and $\mathcal{X}'$ for restarts is to favor a diversified intensification search.

For each restart, we set the diversification level at a high value ($dl = 1.0$) in order to allow the search to perform a series of *Diversification Search* moves during the first iterations of the new round of local search. Our experiments demonstrate that this simple restarting mechanism is quite effective for our problem studied in this paper.

Finally, we have also tried the conventional pure random restart strategy and tested our algorithm without restart at all. It is observed that the proposed elite solution restart strategy slightly outperforms these variants in almost all the cases.

3.7. Discussion

Our *ANS* algorithm is mainly based on a combined neighborhood structure and a adaptive switching mechanism among three different search strategies. Like our *ANS* algorithm, many studies on the nurse rostering problem in the literature take into account the issues of intensification and diversification. On the one hand, our combined neighborhood shares some common features with the Variable Neighborhood Search approach [25], which uses a transition scheme that progressively cycles through higher level neighborhoods and always returns to the simplest neighborhood when improvement occurs. Our *ANS* algorithm nevertheless randomly transits between two neighborhoods at each search step.

On the other hand, our adaptive search strategy switching mechanism borrows some ideas from hyper-heuristics [18] which are high level search strategies manipulating a number of low level heuristics. Like hyper-heuristics, our switching mechanism utilizes the strengths of different search heuristics in order to reach a tradeoff between exploration and exploitation. However, there is an obvious difference that *ANS* adopts an adaptive noise updating strategy utilized in the SAT problem [4] to the nurse rostering problem which seems missing in the previous hyper-heuristic algorithms for nurse rostering and other problems.

4. Computational Results

In this section, we report intensive experimental results of the *ANS* algorithm on 60 instances used in the first nurse rostering competition (*INRC-2010*) and compare them with the best known results found so far[2] and the results obtained by the *INRC-2010* competition finalists.

[2] Our best results: http://www.info.univ-angers.fr/pub/hao/ANS_NRP.html

4.1. INRC-2010 Competition and Test Instances

The *INRC-2010* competition is composed of three tracks, respectively called *Sprint*, *Medium* and *Long* tracks (also called tracks 1, 2 and 3, respectively). Although the problem formulation for all the three tracks is the same throughout the competition, these tracks differ from each other in terms of the allowed CPU time and the size and the characteristics of the proposed instances. The three tracks require a solution within approximately 10 seconds, 10 minutes and 10 hours on a modern PC, respectively, corresponding to different computational environments in real applications.

Each track has three sets of instances, called *Early*, *Late* and *Hidden* instances. The *Early* instances are published when the competition begins. The *Late* instances are available two weeks prior to the deadline of the competition. The *Hidden* instances are kept unavailable to competitors until the end of the competition. The three tracks consist of 30, 15 and 15 instances and named as *Sprint*, *Medium* and *Long*, respectively. All these competition instances[3] and the best known results found so far by all the competitors and researchers[4] are available at the competition web site. Note that the preliminary version of our solver ranks the third and fourth places in the *Sprint* and *Medium* tracks of *INRC-2010* competition, respectively.

4.2. Experimental Protocol

Our algorithm is programmed in C and compiled using GNU GCC on a Cluster with each node running Linux with Intel(R) Xeon(R) E5440 (4 cores) 2.83GHz CPU and 2.0Gb RAM. We report the computational results of our *ANS* algorithm under two timeout conditions: One is the *INRC-2010* competition timeout condition; the other is a relaxed time condition for the first two tracks. For the *INRC-2010* competition timeout condition, the timeout following the competition on our computer is about 15.86, 1051.6 and 51295 seconds for tracks 1, 2 and 3, respectively. This time out is obtained by running a benchmark program available at the *INRC-2010* competition web site. Under the relaxed timeout condition, we employ a time limit of 1000 seconds, 5000 seconds and 20 hours for tracks 1, 2 and 3, respectively.

All the computational results of our *ANS* algorithm were obtained without special tuning of the parameters. The only parameter that varies its value is the local search improvement cutoff α. Under the competition timeout condition, we empirically set $\alpha = 500, 1000$ and 10000 for the three

[3]Competition instances: http://www.kuleuven-kortrijk.be/nrpcompetition/instances

[4]Best known results: http://www.kuleuven-kortrijk.be/nrpcompetition/instances-results (up to April 10th, 2010)

tracks, respectively. Under the relaxed timeout condition, we set $\alpha = 1000$, 3000 and 20000 for tracks 1, 2 and 3, respectively. Table 2 gives the descriptions and settings of the other parameters used in *ANS*, where the last two columns respectively denote the values used in this paper and the preferable value regions.

Given the stochastic nature of the *ANS* algorithm, each problem instance is independently solved 1000, 200 and 20 times for instances of the *Sprint*, *Medium* and *Long* tracks, respectively, under the competition timeout condition and 200, 50 and 5 times for instances of the *Sprint*, *Medium* and *Long* tracks, respectively, under the relaxed timeout condition.

4.3. Results Under INRC-2010 Competition Timeout Condition

Table 3 shows the computational statistics of the *ANS* algorithm on the *INRC-2010* competition instances of the *Sprint*, *Medium* and *Long* tracks. Column 2 gives the previous best known solutions (BKS) uploaded to the *INRC-2010* competition web site by all the researchers. The remaining columns give the results of the *ANS* algorithm according to five criteria: (1) the best objective value (f_{best}), (2) the average objective value (f_{avr}), (3) the standard deviation, σ, over multiple runs, (4) the number of local search iterations, $iter$, for reaching the best objective value f_{best} and (5) the CPU time, t_{best} (in seconds), for reaching the best result f_{best}. The previous best solutions are indicated in bold and the new best solutions found in this paper are indicated in italic.

Table 3 discloses that our *ANS* algorithm obtains quite competitive results on the set of *Sprint* track instances. Specifically, *ANS* can stably reach the previous best known solutions for all the 30 instances under the competition timeout condition. In particular, our *ANS* algorithm improves the previous best known results for 5 instances (*sprint_late04* and *sprint_hidden01, 04, 06 and 08*). Furthermore, *ANS* can reach high quality solutions very stably (with a standard deviation σ less than 2.0 for 23 out of the 30 instances) and the CPU time to reach the best solution is within 5.0 seconds for 27 out of the 30 instances. It shows that only the last five hidden instances (*sprint_hidden06∼10*) present some challenge for *ANS* (with a relative large standard deviation σ). However, *ANS* can even improve the previous best known results for two of these five instances while equalling the other three best ones, demonstrating the efficacy of our algorithm.

For the *Medium* track instances, one finds that our *ANS* algorithm also reaches competitive results on this set of benchmark instances. Specifically, except for 5 *Late* and one *Hidden* instances out of the 15 ones, *ANS* reaches or improves the previous best known results for the left 9 ones.

Particularly, *ANS* can obtain new best solutions for 3 *Hidden* instances (*medium_hidden01, 03 and 05*). In addition, *ANS* obtains high quality solutions with a relative small standard deviation value (less than 2.0) for 10 out of the 15 instances.

Finally, we test the *ANS* algorithm on the set of 15 large *Long* track instances. Our *ANS* algorithm can reach or improve the previous best known results for 10 out of the 15 instances while reaching worse results for 5 other ones. Moreover, *ANS* can obtain new best results for 2 instances (*long_hidden01,02*). These results further provide evidence of the benefit of our *ANS* approach.

4.4. Comparison with the INRC-2010 Competition Finalists

In this section, we compare our *ANS* algorithm with other *INRC-2010* competition finalists under the competition time limit. Table 4 shows the best results obtained by *ANS* and five reference algorithms on the 40 *Early* and *Late* instances. These reference algorithms include a two-phase hybrid solver by the competition winner [28], a branch and price algorithm by Burke and Curtois [12], a general COP solver by Nonobe [27], a hyper-heuristic algorithm by Bilgin *et al* [9] and an ILP algorithm using ILOG CPLEX by Bilgin *et al* [9]. Note that the results in the last column marked with * are proven to be optimal. As before, column 2 also indicates the best known results uploaded to the *INRC-2010* web site.

Table 4 discloses that the best results obtained by our *ANS* algorithm are quite competitive with respect to those of the reference algorithms (best results for each instance are indicated in bold). For the 23 instances whose optimal solutions are known, *ANS* can match 22 of them. One finds that only the solver by Burke and Curtois [12] can reach all of them. The competition winner's solver can obtain 21 of them.

For the 20 hidden instances, the comparison is based on the best known objective values and the winner's solutions which are the only results available to us. From Table 4, one observes that our algorithm improves the previous best known results for 9 out of 20 instances while matching the best known results in 9 other cases, which shows the advantage of our algorithm on these hidden instances.

4.5. Results Under Relaxed Timeout Condition

In this section, we report computational results of our *ANS* algorithm for the *Sprint* and *Medium* tracks under the relaxed timeout condition, as shown in Table 5. The notations are the same as those in previous tables. In this experiment, only the results of the *Late* and *Hidden* instances are listed, since the best known solutions for the *Early* instances can be easily and stably obtained under the competition time limit and cannot be further improved even with more computational resource since all the 20 *Early* instances have been solved to optimality by ILP as shown in Table 4.

For the 20 *Sprint* track instances, our best results cannot be improved with more computational resource. However, both the average solution quality (f_{avr}) and the standard deviation value (σ) are significantly improved. For the 10 *Medium* track instances, our results can be further improved in 6 cases (*medium_late04, 05 and medium_hidden01, 02, 04, 05*), showing the search potential of our *ANS* algorithm under this relaxed time limit condition. Particularly, we obtain new upper bounds for 4 instances (*medium_hidden01, 02, 04, 05*) under this relaxed timeout condition.

Finally, we mention that with a relaxed time limit of 20 hours for the long instances, our algorithm can still improve our result for one instance (long_late03), matching the current best bound.

5. Analysis and Discussion

We now turn our attention to discussing and analyzing several important features of the proposed *ANS* algorithm.

5.1. Importance of Neighborhood Combination

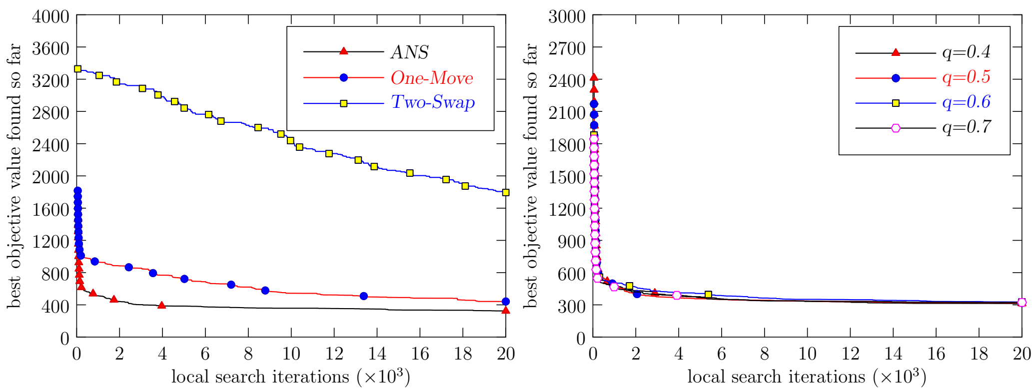

Figure 1: Comparison between different q values (Left: the combined neighborhood and single neighborhoods; Right: different q values from 0.4 to 0.7)

As indicated in Section 3.3, the *ANS* algorithm employs a neighborhood combination strategy to probabilistically select a one-move or two-swap move to perform at each iteration. In order to be sure this combination strategy makes a meaningful contribution, we conduct additional experiments to compare this strategy with the one-move and two-swap neighborhoods alone.

We keep other ingredients unchanged in the *ANS* algorithm and set the value q in Eq. (5) to be $q = 1.0$ and $q = 0.002$ to represent the single one-move and two-swap neighborhoods, respectively. The stopping criterion is the number of local search iterations which is limited to 20,000. The experiments are presented on the medium size instance *medium_hidden02* (which seems to be one of the most difficult instances). Similar results are observed on other instances. The three algorithms are denoted by *ANS*, *One-Move* and *Two-Swap*, respectively. The reason why we use $q = 0.002$ instead of $q = 0.0$ to represent the two-swap neighborhood lies in the fact that the pure two-swap neighborhood ($q = 0.0$) works much worse than with $q = 0.002$.

Figure 1 (left) shows how the best objective value (averaged over 10 independent runs) evolves with the local search iterations. We see that *ANS* converges more quickly towards high quality solutions than with the *One-Move* or *Two-Swap* neighborhood alone. In addition, *ANS* preserves better results than the *One-Move* and *Two-Swap* neighborhoods when the search progresses. This experiment provides an empirical justification of the joint use of the two move operators in the *ANS* algorithm.

In addition, we further compare different values of the important parameter q chosen from 0.4 to 0.7, as shown in Figure 1 (right). The experimental protocol is the same as above. This figure shows that when the search progresses, there is no clear difference between these different q values, implying that q can be arbitrarily chosen from a long range and the performance of ANS will not fluctuate drastically. More generally, we observed that whenever q is not close to 0, ANS performs similarly. This shows that the *One-Move* operator plays a more critical role than the *Two-Swap* move even though a joint use of both moves leads to a better performance. As described in Section 3.3, q is set to be $1 - 0.4 \cdot dens$ which has shown to be robust enough for all the tested instances in this paper.

5.2. Significance of Adaptive Switching Mechanism

In order to evaluate the importance of the adaptive switching mechanism, we compare it with the *Intensive Search* and *Intermediate Search* strategies alone, by setting $\beta_1 = \beta_2 = 1.0$ and $\beta_1 = 0.0$, $\beta_2 = 1.0$, respectively. The experimental protocol is the same as above.

Figure 2 shows how the current objective value (left) and the best objective value (right) evolve with the number of local search iterations. We observe that *ANS* obtains higher quality solutions than both *Intensive* and *Intermediate Searches* during the first iterations in terms of both the current and the best objective values. Furthermore, *ANS* can continuously improve the solution quality when the search progresses, while both *Intensive* and *Intermediate Searches* can only slightly improve the solution quality after the first iterations.

It is noteworthy that there are a lot of "big jumps" during the searching process of the *ANS* algorithm (left figure), which represents the *Diversification* moves in our approach. One observes that it is these "big jumps" (or noises) that allow our *ANS* algorithm to jump out of the local optimum traps, thus guiding the search to explore new search areas. In other words, the diversification process introduced in our approach allows the algorithm to benefit from a better exploration of the search space and prevents the search from stagnating in poor local optima. This experiment also confirms the importance of introducing "noises" to enhance the search power of traditional local search algorithms.

5.3. Tradeoff between Local Search and Restarting Mechanism

We study now another important aspect of the proposed algorithm, i.e., the tradeoff between local search and the restarting mechanism. In fact, the performance and the behavior of *ANS* are influenced by the value of the *improvement cutoff* α of the local search procedure. Under a limited computational resource, the *improvement cutoff* α reflects the relative proportion of restarting and local search in the algorithm. In this section, we analyze the influence of the parameter α on the performance of the *ANS* algorithm. To implement this experiment, we consider 4 different values of

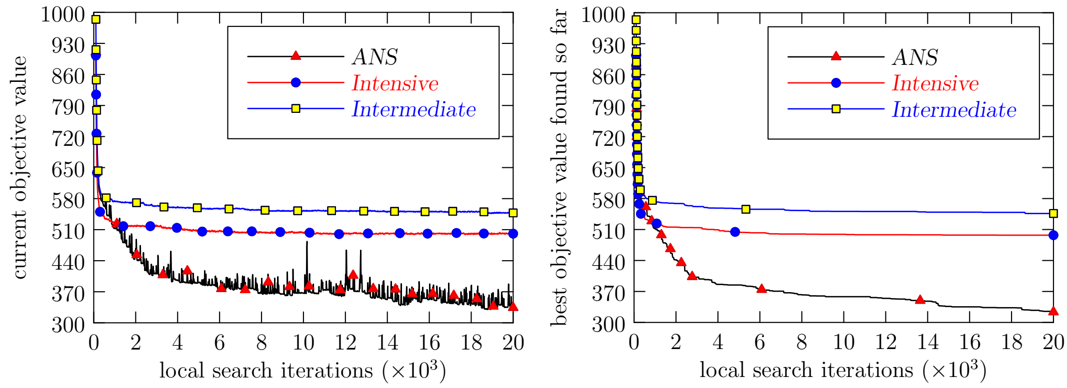

Figure 2: Significance of the adaptive switching mechanism

the parameter α: $\alpha = 1000, 2000, 5000$ and 20000. Figure 3 shows the average evolution of the best solutions during the search obtained with these different values for α.

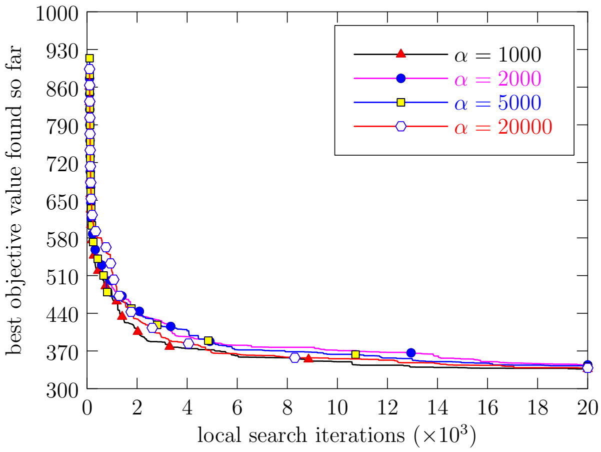

Figure 3: Influence of the improvement cutoff value α

From Figure 3, we notice that these different settings lead to quite similar performance. This phenomenon can be explained by the fact that our local search procedure has very strong diversification capability such that it can automatically switch to the *Diversification Search* once it detects a stagnation behavior, which is equivalent to restarting the search by setting a high diversification level $dl = 1.0$. Thus, this experiment shows a clear advantage that our *ANS* algorithm itself has reached a relatively strong balance between intensification and diversification and thus can be considered as a robust solver.

6. Conclusions

In this paper, we have dealt with the nurse rostering problem which constitutes the topic of the First International Nurse Rostering Competition. In addition to providing a mathematical formulation of the problem, we have presented a unified adaptive neighborhood search algorithm, which integrates a number of original features, to solve this challenging problem. The efficacy of the proposed algorithm is demonstrated on three sets of totally 60 instances used in the *INRC-2010* competition, in comparison with the previous best known results and the winner algorithm of the competition. In particular, we have found new upper bounds for 12 out of the 60 competition instances, as well as matching the previous best known results for 39 instances.

Furthermore, several essential parts of our proposed algorithm are investigated. We have first conducted experiments to demonstrate the significance of the random union combination of the two neighborhoods. In addition, we have carried out experiments to show the importance of the adaptive mechanism based on three search strategies (*Intensive*, *Intermediate* and *Diversification Searches*. Finally, we have shown that our solver is robust and is not very sensitive to the only parameter α.

Given that the adaptive neighborhood search ideas introduced in this paper are independent of the nurse rostering problem, it would be valuable to establish a methodology of this mechanism and to examine its application to other constraint satisfaction and combinatorial optimization problems.

Acknowledgment

We would like to thank the anonymous referees for their helpful comments and questions. Our special thanks go to the *INRC-2010* competition organizers for defining this challenging problem. The work is partially supported by the regional RaDaPop (2009-2013), LigeRo projects (2009-2013).Foundation of China (Grant No. 61100144).

[1] U. Aickelin, K. Dowsland, Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem, Journal of Scheduling 3 (2000) 139–153.

[2] U. Aickelin, K. Dowsland, An indirect genetic algorithm for a nurse scheduling problem, Computers & Operations Research 31 (2004) 761–778.

[3] U. Aickelin, J. Li, An estimation of distribution algorithm for nurse scheduling, Annals of Operations Research 155 (2007) 289–309.

[4] H. Hoos, An adaptive noise mechanism for WalkSAT, in: Proceedings of AAAI-2002, 655–660.

[5] R. Bai, E.K. Burke, G. Kendall, B. McCollum, An evolutionary approach to the nurse rostering problem, IEEE Transactions on Evolutionary Computation 14(4) (2010) 580–590.

[6] J. Bard, H.W. Purnomo, Preference scheduling for nurses using column generation, European Journal of Operational Research 164 (2005) 510–534.

[7] N. Beaumont, Scheduling staff using mixed integer programming, European Journal of Operational Research 98 (1997) 473–484.

[8] F. Bellanti, G. Carello, F. Della Croce, R. Tadei, A greedy-based neighborhood search approach to a nurse rostering problem, European Journal of Operational Research 153 (2004) 28–40.

[9] B. Bilgin, P. Demeester, M. Mısır, W. Vancroonenburg, G. Vanden Berghe, T. Wauters, A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition, PATAT 2010.

[10] M.J. Brusco, L.W. Jacobs, A simulated annealing approach to the cyclic staff-scheduling problem, Naval Research Logistics 40 (1993) 69–84.

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

[12] E.K. Burke, T. Curtois, An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition–2010, PATAT 2010.

[13] E.K. Burke, T. Curtois, G. Post, R. Qu, B. Veltman, A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem, European Journal of Operational Research 188 (2008) 330–341.

[14] E.K. Burke, T. Curtois, R. Qu, G. Vanden Berghe, A scatter search for the nurse rostering problem, Journal of the Operational Research Society, 61 (2010) 1667–1679.

[15] E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe, Variable neighborhood search for nurse rostering problems, Metaheuristics: Computer Decision-Making, Kluwer, 2004, pp. 153–172.

[16] E.K. Burke, P. De Causmaecker, G. Vanden Berghe, A hybrid Tabu Search algorithm for the nurse rostering Problem, Lecture Notes in Computer Science 1585 (1998) 187–194.

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

[18] E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, S. Schulenburg, Hyper-heuristics: An emerging direction in modern search technology, Handbook of Metaheuristics, Kluwer, 2003, pp. 457–474.

[19] S. Haspeslagh, P. De Causmaecker, M. Stølevik, A. Schaerf, First International Nurse Rostering Competition 2010. PATAT 2010.

[20] C.A. Glass, R.A. Knight, The nurse rostering problem: A critical appraisal of the problem structure, European Journal of Operational Research 202(2) (2010) 379–389.

[21] F. Glover, M. Laguna, Tabu Search, Kluwer Academic, Boston, 1997.

[22] F.F. Easton, N. Mansour, A distributed genetic algorithm for deterministic and stochastic labor scheduling problems, European Journal of Operational Research 118 (1999) 505–523.

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

[24] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, E. Burke, Setting the research agenda in automated timetabling: The second international timetabling competition, INFORMS Journal on Computing 22(1) (2010) 120–130.

[25] N. Mladenović, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (11) (1997) 1097–1100.

[26] C.W. Mueller, J.C. McCloskey, Nurses job satisfaction: A proposed measure, Nursing Research 39 (1990) 113–117.

[27] K. Nonobe, INRC2010: An approach using a general constraint optimization solver, PATAT 2010.

[28] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, E. Housos, A systematic two phase approach for the Nurse Rostering problem, PATAT 2010.

[29] C. Valouxis, E. Housos, Hybrid optimization techniques for the workshift and rest assignment of nursing personnel, Artificial Intelligence in Medicine 20 (2000) 155–175.

Table 1: Constant and variable notations used in the mathematical formulation

Symbols	Constr.	Description
$sc(d,h)$	H_1	the total number of required nurses for day $d \in \mathcal{D}$ and shift type $h \in \mathcal{H}$
$shift(s)^{+,-}$	$S_{1,2}$	the maximum/minimum number of shifts that can be assigned to nurse s
$work(s)^{+,-}$	$S_{3,4}$	the maximum/minimum number of consecutive working days of nurse s
$free(s)^{+,-}$	$S_{5,6}$	the maximum/minimum number of consecutive free days of nurse s
$Night$	S_7	the night shift type
$wkd(s)^{+,-}$	$S_{8,9}$	the maximum/minimum number of consecutive working weekends of nurse s
n_wkd	S_{10-12}	the total number of weekends
$n\_wkd(s)^{+}$	S_{10}	the maximum number of working weekends of nurse s
$nd(s)$	S_{11-12}	the total number of days for each weekend of nurse s, which could be 2 or 3
$day\_req(s,d)$	$S_{13,14}$	*on* (*off*) if nurse s requests (not) to work any shift at day d; *null* otherwise
$sh\_req(s,d,h)$	$S_{15,16}$	*on* (*off*) if nurse s requests (not) to work shift h at day d; *null* otherwise
$qual(s,h)$	S_{17}	*true* if nurse s has all the required skills of shift h; *false* otherwise
$unwantp(s)$	S_{18}	the set of the unwanted patterns of nurse s
Variables		
$n\_wksect(s)$	$S_{3,4}$	the number of working sections of nurse s, where a working section is a series of consecutive working days
$len\_wksect(s,i)$	$S_{3,4}$	the length of the ith working section of nurse s
$n\_frsect(s)$	$S_{5,6}$	the number of free sections of nurse s, where a free section is a series of consecutive free days
$len\_frsect(s,i)$	$S_{5,6}$	the length of the ith free section of nurse s
$n\_wkdsect(s)$	$S_{8,9}$	the number of weekend working sections of nurse s, where a weekend working section is a series of consecutive working weekends
$len\_wkdsect(s,i)$	$S_{8,9}$	the length of the ith weekend working section of nurse s
$h\_wkd(s,i,j)$	S_{12}	the shift type assignment at the jth day of the ith weekend for nurse s
$nwd(s,i)$	S_{10-12}	the number of working days of nurse s at the ith weekend, i.e., $nwd(s,i) = \sum_{j=1}^{nd(s)} \chi(h\_wkd(s,i,j) \neq -1)$
$nh(s,i,h)$	S_{12}	the number of shift type h of nurse s at the ith weekend, i.e., $nh(s,i,h) = \sum_{j=1}^{nd(s)} \chi(h\_wkd(s,i,j) = h)$
$n\_unwp(s,p)$	S_{18}	the total number of occurring patterns of type p for nurse s

Table 2: Settings of important parameters

Parameters	Section	Description	Values	
			This paper	Preferable
φ	3.3	neighborhood selection coefficient	0.4	[0.2, 0.8]
tl	3.4	tabu tenure constant	$0.8S$	$[0.6S, 0.85S]$
$\lvert\mathcal{S}^*\rvert$	3.4	move size of intermediate search	$0.5S$	$[0.45S, 0.7S]$
β_1	3.4	search strategy selection coefficient	0.3	[0.2, 0.4]
β_2	3.4	search strategy selection coefficient	0.7	[0.5, 0.8]
θ	3.5	threshold for adaptive adjustment	$\frac{SH}{10}$	$[\frac{SH}{12}, \frac{SH}{9}]$

Table 3: Computational results under the *INRC-2010* competition time limit

Instance	BKS	*ANS* Algorithm				
		f_{best}	f_{avr}	σ	$iter$	t_{best}
sprint01	**56**	**56**	56.050	0.219	1111	0.09
sprint02	**58**	**58**	58.058	0.234	2486	0.21
sprint03	**51**	**51**	51.269	0.604	3613	0.30
sprint04	**59**	**59**	59.695	0.683	6535	0.56
sprint05	**58**	**58**	58.034	0.180	688	0.06
sprint06	**54**	**54**	54.168	0.374	1307	0.11
sprint07	**56**	**56**	56.218	0.470	3591	0.30
sprint08	**56**	**56**	56.067	0.250	893	0.07
sprint09	**55**	**55**	55.412	0.571	3522	0.29
sprint10	**52**	**52**	52.235	0.480	2063	0.17
sprint_late01	**37**	**37**	40.309	1.305	18470	4.37
sprint_late02	**42**	**42**	43.796	0.947	8565	0.84
sprint_late03	**48**	**48**	50.464	1.145	7084	1.62
sprint_late04	75	***73***	84.846	5.110	33139	7.71
sprint_late05	**44**	**44**	46.035	0.921	12517	2.90
sprint_late06	**42**	**42**	42.289	0.463	1941	0.08
sprint_late07	**42**	**42**	44.194	1.550	37419	1.62
sprint_late08	**17**	**17**	17.000	0.000	126	0.00
sprint_late09	**17**	**17**	17.000	0.000	84	0.00
sprint_late10	**43**	**43**	46.347	1.993	9332	0.40
sprint_hidden01	33	***32***	34.703	1.652	14388	1.32
sprint_hidden02	**32**	**32**	33.769	1.400	3153	0.28
sprint_hidden03	**62**	**62**	65.163	2.172	6305	1.49
sprint_hidden04	67	***66***	68.771	1.536	9906	2.40
sprint_hidden05	**59**	**59**	62.851	1.950	7176	1.64
sprint_hidden06	134	***130***	146.614	10.992	95327	8.85
sprint_hidden07	**153**	**153**	173.042	16.978	9179	0.82
sprint_hidden08	209	***204***	232.278	13.576	21450	5.09
sprint_hidden09	**338**	**338**	358.350	10.620	11139	2.70
sprint_hidden10	**306**	**306**	339.885	17.242	6898	1.61
medium01	**240**	**240**	240.943	0.464	90731	22.08
medium02	**240**	**240**	240.606	0.521	175726	42.33
medium03	**236**	**236**	236.996	0.377	261635	62.83
medium04	**237**	**237**	237.976	0.154	277283	66.49
medium05	**303**	**303**	303.870	0.674	172359	41.77
medium_late01	**158**	164	174.245	3.743	1343797	648.48
medium_late02	**18**	20	24.968	1.894	1281136	616.91
medium_late03	**29**	30	33.804	1.502	2380850	593.10
medium_late04	**35**	36	40.388	1.841	2113881	996.13
medium_late05	**107**	117	133.791	5.994	1266691	678.04
medium_hidden01	130	***122***	139.965	6.958	1541511	1014.00
medium_hidden02	**221**	224	243.617	9.355	1254666	830.66
medium_hidden03	36	***35***	40.206	1.852	753233	502.75
medium_hidden04	**80**	**80**	85.617	1.921	1574698	1026.09
medium_hidden05	**122**	***120***	129.370	4.119	1393978	892.08
long01	**197**	**197**	197.933	0.573	22323924	16874.94
long02	**219**	222	224.650	1.415	28821326	22195.02
long03	**240**	**240**	240.000	0.000	49639	35.55
long04	**303**	**303**	303.267	0.442	2059272	1456.06
long05	**284**	**284**	284.267	0.442	1111975	785.86
long_late01	**235**	237	242.400	2.703	31966257	36327.86
long_late02	**229**	**229**	239.000	2.859	31467349	35777.50
long_late03	**220**	222	230.783	3.787	9767199	11006.73
long_late04	**221**	227	232.478	3.412	10082982	11366.68
long_late05	**83**	**83**	84.016	1.175	9758472	11136.71
long_hidden01	363	***346***	348.783	0.720	20298494	24485.10
long_hidden02	90	***89***	90.870	0.899	4385549	5059.59
long_hidden03	**38**	**38**	38.591	0.834	2995944	3423.21
long_hidden04	**22**	**22**	22.000	0.000	1957811	2232.43
long_hidden05	**41**	45	50.571	3.417	25735308	29859.56

Table 4: Comparison with other competition finalists under *INRC-2010* time limit

Instance	BKS	f_{best}					
		Ours	Winner [28]	Burke [12]	Nonobe [27]	Bilgin [9]	ILP [9]
sprint01	**56**	**56**	**56**	**56**	**56**	57	**56***
sprint02	**58**	**58**	**58**	**58**	**58**	59	**58***
sprint03	**51**	**51**	**51**	**51**	**51**	**51**	**51***
sprint04	**59**	**59**	**59**	**59**	**59**	60	**59***
sprint05	**58**	**58**	**58**	**58**	**58**	**58**	**58***
sprint06	**54**	**54**	**54**	**54**	**54**	**54**	**54***
sprint07	**56**	**56**	**56**	**56**	**56**	**56**	**56***
sprint08	**56**	**56**	**56**	**56**	**56**	**56**	**56***
sprint09	**55**	**55**	**55**	**55**	**55**	**55**	**55***
sprint10	**52**	**52**	**52**	**52**	**52**	**52**	**52***
sprint_late01	**37**	**37**	**37**	**37**	**37**	40	39
sprint_late02	**42**	**42**	**42**	**42**	**42**	44	43
sprint_late03	**48**	**48**	**48**	**48**	**48**	50	54
sprint_late04	75	***73***	76	75	76	81	99
sprint_late05	**44**	**44**	**44**	**44**	45	45	47
sprint_late06	**42**	**42**	**42**	**42**	**42**	**42**	**42***
sprint_late07	**42**	**42**	43	**42**	43	46	**42***
sprint_late08	**17**	**17**	**17**	**17**	**17**	**17**	21
sprint_late09	**17**	**17**	**17**	**17**	**17**	**17**	35
sprint_late10	**43**	**43**	44	**43**	44	46	**43***
sprint_hidden01	33	***32***	33	–	–	–	–
sprint_hidden02	**32**	**32**	33	–	–	–	–
sprint_hidden03	**62**	**62**	62	–	–	–	–
sprint_hidden04	67	***66***	67	–	–	–	–
sprint_hidden05	**59**	**59**	60	–	–	–	–
sprint_hidden06	134	***130***	139	–	–	–	–
sprint_hidden07	**153**	**153**	153	–	–	–	–
sprint_hidden08	209	***204***	220	–	–	–	–
sprint_hidden09	**338**	**338**	338	–	–	–	–
sprint_hidden10	**306**	**306**	306	–	–	–	–
medium01	**240**	**240**	**240**	**240**	241	242	**240***
medium02	**240**	**240**	**240**	**240**	**240**	241	**240***
medium03	**236**	**236**	**236**	**236**	**236**	238	**236***
medium04	**237**	**237**	**237**	**237**	238	238	**237***
medium05	**303**	**303**	**303**	**303**	304	304	**303***
medium_late01	158	164	159	***157***	176	163	219
medium_late02	**18**	20	20	**18**	19	21	41
medium_late03	**29**	30	30	**29**	30	32	37
medium_late04	**35**	36	36	**35**	37	38	42
medium_late05	**107**	117	113	**107**	125	122	153
medium_hidden01	130	***122***	131	–	–	–	–
medium_hidden02	**221**	224	221	–	–	–	–
medium_hidden03	36	***35***	38	–	–	–	–
medium_hidden04	**80**	**80**	81	–	–	–	–
medium_hidden05	**122**	***120***	122	–	–	–	–
long01	**197**	**197**	**197**	**197**	**197**	**197**	**197***
long02	**219**	222	**219**	**219**	224	220	**219***
long03	**240**	**240**	**240**	**240**	**240**	**240**	**240***
long04	**303**	**303**	**303**	**303**	**303**	**303**	**303***
long05	**284**	**284**	**284**	**284**	**284**	**284**	**284***
long_late01	**235**	237	239	**235**	267	241	241
long_late02	**229**	**229**	231	**229**	245	245	237
long_late03	**220**	222	222	**220**	254	233	229
long_late04	**221**	227	228	**221**	260	246	232
long_late05	**83**	**83**	**83**	**83**	93	87	90
long_hidden01	363	***346***	363	–	–	–	–
long_hidden02	90	***89***	106	–	–	–	–
long_hidden03	**38**	**38**	38	–	–	–	–
long_hidden04	**22**	**22**	22	–	–	–	–
long_hidden05	**41**	45	41	–	–	–	–

Table 5: Computational results for the *Sprint* and *Medium* tracks under relaxed time limit

Instance	BKS	*ANS* Algorithm				
		f_{best}	f_{avr}	σ	$iter$	t_{best}
sprint_late01	37	37	38.298	0.829	58454	13.64
sprint_late02	42	42	42.663	0.481	96489	9.35
sprint_late03	48	48	48.797	0.679	92411	21.27
sprint_late04	75	***73***	75.842	1.778	267513	60.94
sprint_late05	44	44	44.349	0.494	23195	5.40
sprint_late06	42	42	42.000	0.000	2839	0.12
sprint_late07	42	42	42.713	0.649	50803	2.16
sprint_late08	17	17	17.000	0.000	116	0.00
sprint_late09	17	17	17.000	0.000	76	0.00
sprint_late10	43	43	44.017	0.785	72709	3.05
sprint_hidden01	33	***32***	32.282	0.459	9441	0.86
sprint_hidden02	32	32	32.017	0.128	3361	0.29
sprint_hidden03	62	62	62.324	0.600	12897	2.98
sprint_hidden04	67	***66***	66.046	0.228	30894	7.35
sprint_hidden05	59	59	59.542	0.694	21307	4.91
sprint_hidden06	134	***130***	132.502	3.679	88936	8.06
sprint_hidden07	153	153	156.307	4.808	27007	2.39
sprint_hidden08	209	***204***	211.751	7.286	102836	24.00
sprint_hidden09	338	338	343.091	4.971	206956	50.14
sprint_hidden10	306	306	319.431	12.882	162436	36.89
medium_late01	158	164	172.084	3.810	6980981	3327.11
medium_late02	18	20	23.622	1.796	5568298	2634.84
medium_late03	29	30	32.958	1.362	16825097	4146.19
medium_late04	35	35	39.244	1.782	4925094	2291.98
medium_late05	107	112	127.126	5.832	2531760	1327.90
medium_hidden01	130	***117***	133.000	7.301	5038892	3364.43
medium_hidden02	221	***220***	232.235	7.754	1966573	1286.51
medium_hidden03	36	***35***	38.731	1.948	2218753	1456.58
medium_hidden04	80	***79***	84.017	1.923	2668541	1712.39
medium_hidden05	122	***119***	125.513	3.325	4367953	2763.44