European Journal of Operational Research 258 (2017) 411–423

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A hybrid Integer Programming and Variable Neighbourhood Search algorithm to solve Nurse Rostering Problems

Erfan Rahimian[a,*], Kerem Akartunalı[a], John Levine[b]

[a] Department of Management Science, University of Strathclyde, 130 Rottenrow, Glasgow G4 0GE, UK
[b] Computer and Information Sciences, University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH, UK

ARTICLE INFO

Article history:
Received 24 January 2016
Accepted 19 September 2016
Available online 22 September 2016

Keywords:
Timetabling
Nurse Rostering
Hybrid algorithm
Integer Programming
Variable Neighbourhood Search

ABSTRACT

The Nurse Rostering Problem (NRP) is defined as assigning a number of nurses to different shifts during a specified planning period, considering some regulations and preferences. This is often very difficult to solve in practice particularly by applying a sole approach. In this paper, we propose a novel hybrid algorithm combining the strengths of Integer Programming (IP) and Variable Neighbourhood Search (VNS) algorithms to design a hybrid method for solving the NRP. After generating the initial solution using a greedy heuristic, the solution is further improved by employing a Variable Neighbourhood Descent algorithm. Then IP, deeply embedded in the VNS algorithm, is employed within a ruin-and-recreate framework to assist the search process. Finally, IP is called again to further refine the solution during the remaining time. We utilise the strength of IP not only to diversify the search process, but also to intensify the search efforts. To identify the quality of the current solution, we use a new generic scoring scheme to mark the low-penalty parts of the solution. Based on the computational tests with 24 instances recently introduced in the literature, we obtain better results with our proposed algorithm, where the hybrid algorithm outperforms two state-of-the-art algorithms and Gurobi in most of the instances. Furthermore, we introduce 11 randomly generated instances to further evaluate the efficiency of the hybrid algorithm, and we make these computationally challenging instances publicly available to other researchers for benchmarking purposes.

1. Introduction

Nurse Rostering (also referred to as *Nurse Scheduling*) is the process of creating a schedule by assigning some nurses to different shift types, e.g. day, and night, during a predetermined planning horizon, where many limitations such as hospital regulations and employee contracts as well as management and individual preferences are taken into account. The output of this process is a roster of working shifts for all the involved nurses, which is expected to result in an increase of job satisfaction and staff utilisation while reducing stress and outsourcing cost (Burke, De Causmaecker, Berghe, & Van Landeghem, 2004; Ernst, Jiang, Krishnamoorthy, Owens, & Sier, 2004a; Ernst, Jiang, Krishnamoorthy, & Sier, 2004b). Real-world Nurse Rostering Problems are very difficult to solve and comprise many challenges for the people involved in the preparation process, e.g. personnel managers, and head nurses (Ernst et al., 2004b).

Many studies have been accomplished for the Nurse Rostering Problem (NRP) over the last few decades, with a variety of methods and algorithms applied to solve this problem in real-world settings. The proposed approaches are mainly based on meta-heuristic algorithms (Blum & Roli, 2003; Glover & Kochenberger, 2003; Talbi, 2009), which are straightforward and effective for many practical problems. These range from Variable Neighbourhood Search (Della Croce & Salassa, 2014; Stølevik, Nordlander, Riise, & Froyseth, 2011) and Tabu Search (Burke, Causmaecker, & Berghe, 1999) to Genetic Algorithms (Aickelin & Dowsland, 2004) and tailor-made heuristics (Lu & Hao, 2012; Valouxis, Gogos, Goulas, Alefragis, & Housos, 2012). However, meta-heuristic algorithms are not as efficient for problem instances where the structure of the problem is very complex, making it challenging to find a good-quality (or even a feasible) solution in a reasonable runtime. On the other hand, there is also some research employing exact approaches such as Integer Programming (IP) (Beaumont, 1997; Dowsland & Thompson, 2000) and Constraint Programming (CP) (Bourdais, Galinier, & Pesant, 2003; Cipriano, Gaspero, & Dovier, 2006), which are very powerful at dealing with complex structures. Nevertheless, they are not efficient enough for solving

* Corresponding author.
E-mail addresses: erfan.rahimian@strath.ac.uk (E. Rahimian), kerem.akartunali@strath.ac.uk (K. Akartunalı), john.levine@strath.ac.uk (J. Levine).

http://dx.doi.org/10.1016/j.ejor.2016.09.030
0377-2217/

many medium- to large-scale problem instances in practice, even though there are some very powerful and mature commercial solvers applying these methods such as Gurobi (Gurobi Optimization, 2015) and IBM CP Optimiser (IBM, 2015). Having said that, in recent years, some researchers have focused on combining these two approaches to utilise their complementary strengths in order to solve highly-constrained real-world NRPs efficiently (Burke, Li, & Qu, 2010; Qu & He, 2009; Rahimian, Akartunali, & Levine, 2015; Stølevik et al., 2011).

In this paper, we propose a novel hybrid Integer Programming and Variable Neighbourhood Search (VNS) algorithm to solve the Nurse Rostering Problem in modern hospital environments. We employ IP not only to diversify the search process, but also to improve the quality of the obtained solutions from the VNS algorithm in a creative way. First, a greedy heuristic is used to generate an initial solution, and then the generated solution is further improved using a VNS algorithm until a stopping criterion is met. To further enhance the efficiency of the VNS algorithm, IP is employed iteratively during the running of the algorithm as a neighbourhood structure to improve the quality of the incumbent solution using a *ruin-and-recreate* framework (Stølevik et al., 2011). In this framework, the high-penalty components of the solution are destroyed according to a generic scoring scheme, and then they are created again by an IP solver. Finally, IP is applied once more to the best-found solution to improve it globally as much as possible until the overall time limit is reached. The proposed algorithm is designed to perform efficiently when only short computational times are available, so that many practical problems can be tackled.

The novelty of our approach is to embed IP as a neighbourhood structure through a ruin-and-recreate framework in the VNS algorithm to improve the quality of the obtained solution and diversify the search process at the same time. Our method of hybridisation is entirely different from the similar algorithms reported in the literature (Burke et al., 2010; Qu & He, 2009; Stølevik et al., 2011). In fact, there are various hybridisation schemes in order to combine different approaches together (Raidl, Puchinger, & Blum, 2010). For example, Qu and He (2009) applied CP to generate an initial solution by decomposing the problem to various subproblems, and then applying VNS to improve the generated solution. Stølevik et al. (2011) applied an Iterated Local Search framework for generating an initial solution and employed VNS and CP in order to improve the solution and diversify the search process, respectively. Burke et al. (2010) employed IP to generate a solution satisfying all hard constraints, and then improve it using VNS to satisfy the remaining soft constraints. In most of the mentioned approaches, IP or CP is used to generate a solution satisfying some constraints of the problem (or parts of the problem), and then a meta-heuristic algorithm is applied to further improve the generated solution. However, in our approach, we employ VNS as the main local search framework and then embed IP as a neighbourhood structure to intensify and diversify the search process in an iterative manner considering all the constraints. Indeed, we use IP through a ruin-and-recreate strategy to escape from local optima and at the same time, improve the quality of the obtained solution. Having said that, incorporating IP in our hybrid algorithm, we also allow the search process to traverse the infeasible space by allowing all the constraints to be violated in order to find out the latent feasible solutions. Moreover, we hybridise IP through VNS in a lower level compared with the approaches reported in the literature (Talbi, 2009) and therefore we exploit the complementary strengths of both methods in a more sophisticated and effective way. In addition, we have applied a scoring scheme to evaluate the quality of the obtained solution according to the associated underlying elements such as nurses or days, which empower the hybrid algorithm to focus on parts of the solution having the most likelihood of gaining a better solution. The proposed algorithm also works with a pre-determined time limit in which the algorithm tries to generate the best solution.

The rest of this paper is organised as follows. We first describe the studied Nurse Rostering Problem and present the relevant IP formulation in Section 2. Next, we elaborate on the solution method and different components of the proposed hybrid algorithm in Section 3. Finally, in Sections 4 and 5, we present our computational results, and draw some conclusions and potential future research directions, respectively.

2. Problem description and IP formulation

In this section, we provide a brief description of the studied problem and the relevant constraints, and present a mathematical formulation. For further information regarding the problem, we refer interested readers to Curtois and Qu (2014), where the detailed description of the problem as well as some instances are presented.

The NRP is defined as assigning a number of nurses to different shifts (e.g. early, late) during a specified planning period, where some regulations (e.g. employee contracts) and preferences (e.g. individual requested days off) are taken into account. Most NRPs including the studied problem are $\mathcal{NP}$-hard (Chuin Lau, 1996; Osogami & Imai, 2000) and computationally challenging, and have a very complex structure even when the problem size is relatively small. Tackling this problem in real-world settings, the constraints of the problem are often classified as hard and soft constraints. Hard constraints are necessary to be satisfied under any circumstances, and therefore, make a problem feasible when they are met. Soft constraints, on the other hand, are those we would prefer to be met (but are not crucial), and define the quality of a generated roster according to the degree to which they are satisfied. Therefore, the objective is to reduce the number of violations associated with the soft constraints as much as possible, i.e. increase the quality of the roster. In the following, the hard and soft constraints of the problem (denoted by prefixes *HC* and *SC*, respectively) are explained:

- *HC1*: nurses cannot be assigned more than one shift on a day.
- *HC2 [Shift rotations]*: the shift assignment of nurses on two consecutive days must comply with the pre-defined set of shift patterns (rotations). The shift patterns prevent forbidden shift sequences.
- *HC3 [Maximum number of shifts]*: the maximum number of shift types that can be assigned to each nurse within the planning period.
- *HC4 [Maximum total minutes]*: the maximum amount of total time in minutes that can be assigned to each nurse within the planning period.
- *HC5 [Minimum total minutes]*: the minimum amount of total time in minutes that can be allocated to each nurse within the planning period.
- *HC6 [Maximum consecutive shifts]*: the maximum number of consecutive shifts, which are allowed to be worked within the planning period.
- *HC7 [Minimum consecutive shifts]*: the minimum number of consecutive shifts, which are allowed to be worked within the planning period.
- *HC8 [Minimum consecutive days off]*: the minimum number of consecutive days off, which are allowed to be assigned within the planning period.
- *HC9 [Maximum number of weekends]*: the maximum number of worked weekends (a weekend is defined as being worked if there is a shift on Saturday or Sunday) within the planning period.

- *HC10 [Requested days off]*: shifts must not be assigned to a specified nurse on some specified days.
- *SC1 [Shift on/off requests]*: a shift assignment to a specific nurse should comply with a pre-defined set of preferences. The penalty associated with this constraint is equal to the total number of all violated assignments multiplied by the specified relevant weight defined in the problem data.
- *SC2 [Coverage]*: the required number of nurses assigned to a specified day for a specified shift should be within a particular range. The penalty associated with this constraint is equal to the total amount of violated coverage multiplied by the specified relevant under- or over-weight defined in the problem data.

For constraints *HC2* and *HC6*, it is assumed that the last day of the previous planning period and the first day of the next planning horizon are days off. Furthermore, for constraint *HC7*, it is assumed that there are an infinite number of consecutive shifts assigned at the end of the previous planning period and at the start of the next planning period. For constraint *HC8*, a similar arrangement applies with days off.

Based on the problem definition, we will present the associated IP formulation, in a similar fashion to the formulation given in Curtois and Qu (2014), which will be crucial for the IP components of our proposed hybrid algorithm. This IP model helps us to facilitate the search process with an IP solver in order to have better exploration and exploitation. Next, we present our notations before presenting the formulation.

Sets and parameters:

D	set of days in the planning horizon
W	set of weekends in the planning horizon
I	set of nurses
T	set of shift types
R_t	set of shift types that cannot be assigned immediately after shift type $t \in T$
N_i	set of days that nurse $i \in I$ cannot be assigned a shift on
l_t	length of shift type $t \in T$ in minutes
m_{it}^{max}	maximum number of shifts of type $t \in T$ that can be assigned to nurse $i \in I$
b_i^{min}, b_i^{max}	minimum and maximum number of minutes that nurse $i \in I$ must be assigned
c_i^{min}, c_i^{max}	minimum and maximum number of consecutive shifts that nurse $i \in I$ must work. c is the index of possible number of consecutive shifts
o_i^{min}	minimum number of consecutive days off that nurse $i \in I$ can be assigned. b is the index of possible number of consecutive days off
a_i^{max}	maximum number of weekends that nurse $i \in I$ can work
q_{idt}	the incurred penalty if shift type $t \in T$ is not assigned to nurse $i \in I$ on day $d \in D$
p_{idt}	the incurred penalty if shift type $t \in T$ is assigned to nurse $i \in I$ on day $d \in D$
u_{dt}	preferred total number of nurses to whom is assigned shift type $t \in T$ on day $d \in D$
$w_{dt}^{min}, w_{dt}^{max}$	under-weight and over-weight relevant to the total coverage of shift type $t \in T$ on day $d \in D$

Decision variables:

x_{idt}	$= 1$ if nurse $i \in I$ is assigned to shift type $t \in T$ on day $d \in D$, $= 0$ otherwise
k_{iw}	$= 1$ if nurse $i \in I$ works on weekend $w \in W$, $= 0$ otherwise
y_{dt}	total number of nurses below the preferred coverage for shift type $t \in T$ on day $d \in D$
z_{dt}	total number of nurses above the preferred coverage for shift type $t \in T$ on day $d \in D$
v_{idt}	total incurred penalty relevant to shift on/off requests of nurse $i \in I$ for shift type $t \in T$ on day $d \in D$

Constraints:

$$\sum_{t\in T} x_{idt} \le 1, \quad \forall i \in I, d \in D \tag{HC1}$$

$$x_{idt} + x_{i(d+1)u} \le 1, \quad \forall i \in I, d \in \{1 \dots |D| - 1\}, t \in T, u \in R_t \tag{HC2}$$

$$\sum_{d\in D} x_{idt} \le m_{it}^{max}, \quad \forall i \in I, t \in T \tag{HC3}$$

$$b_i^{min} \le \sum_{d\in D}\sum_{t\in T} l_t x_{idt} \le b_i^{max}, \quad \forall i \in I \tag{HC4, HC5}$$

$$\sum_{j=d}^{d+c_i^{max}} \sum_{t\in T} x_{ijt} \le c_i^{max}, \quad \forall i \in I, d \in \{1 \dots |D| - c_i^{max}\} \tag{HC6}$$

$$\sum_{t\in T} x_{ijt} + c - 1 - \sum_{j=d+1}^{d+c}\sum_{t\in T} x_{ijt} + \sum_{t\in T} x_{i(d+c+1)t} \ge 0, \tag{HC7}$$
$$\forall i \in I, c \in \{1 \dots c_i^{min} - 1\}, d \in \{1 \dots |D| - (c+1)\}$$

error

$$1 - \sum_{t\in T} x_{ijt} + \sum_{j=d+1}^{d+b}\sum_{t\in T} x_{ijt} + \sum_{t\in T} x_{i(d+b+1)t} \ge 0, \tag{HC8}$$

error

$$\forall i \in I, b \in \{1 \dots o_i^{min} - 1\}, d \in \{1 \dots |D| - (b+1)\}$$

$$k_{iw} \le \sum_{t\in T} x_{i(7w-1)t} + \sum_{t\in T} x_{i(7w)t} \le 2k_{iw}, \quad \forall i \in I, w \in W, \tag{HC9}$$
$$\sum_{w\in W} k_{iw} \le a_i^{max}, \quad \forall i \in I$$

$$x_{int} = 0, \quad \forall i \in I, n \in N_i, t \in T \tag{HC10}$$

$$q_{idt}(1 - x_{idt}) + p_{idt}x_{idt} = v_{idt}, \quad \forall i \in I, d \in D, t \in T \tag{SC1}$$

$$\sum_{i\in I} x_{idt} - z_{dt} + y_{dt} = u_{dt}, \quad \forall d \in D, t \in T \tag{SC2}$$

$$x_{idt}, k_{iw} \in \{0, 1\}, y_{dt}, z_{dt}, v_{idt} \in \mathbb{Z}, \quad \forall i \in I, d \in D, t \in T, w \in W$$

Objective function:

$$\min \sum_{i\in I}\sum_{d\in D}\sum_{t\in T} v_{idt} + \sum_{d\in D}\sum_{t\in T} w_{dt}^{min} y_{dt} + \sum_{d\in D}\sum_{t\in T} w_{dt}^{max} z_{dt}$$

It is also assumed that all weeks start on Monday and the planning horizon consists of a whole number of weeks. We will discuss some statistics relevant to the studied instances based on the presented IP model in Section 4.

3. Hybrid approach

In this section, we describe a hybrid method combining Variable Neighbourhood Search and Integer Programming techniques (aka. a pseudo-exact or matheuristic Raidl & Puchinger, 2008) to solve modern Nurse Rostering Problems. The schematic overview of the proposed hybrid algorithm is demonstrated in Algorithm 1.

After generating an initial solution using a greedy heuristic (*GreedyHeuristic()*), a Variable Neighbourhood Descent (VND) algorithm (*VNDSearch()*) using a set of distinct neighbourhoods tries to improve the generated initial solution until no more improvements can be obtained by cycling through all the neighbourhoods. Then

Algorithm 1: The overall pseudo code of the hybrid algorithm.

```
x* ← x ← GreedyHeuristic();
repeat
    x ← VNDSearch(x);
    x ← IPRuinAndRecreate(x);
    if x < x* then
        x* ← x;
    end
until some stopping criteria are met;
x* ← IPImprove(x*);
return [x*]
```

the best solution obtained from the VND algorithm is employed by an IP solver by fixing low-penalty parts of the solution, where it tries to generate a better-quality (exploitation) and a different-structured (exploration) solution. In fact, in this step, the best solution obtained so far is partially destroyed and again recreated aiming to have a higher-quality, and at the same time, a different solution in terms of the underpinning structure (a ruin-and-recreate strategy). In other words, the IP solver is applied as a shaking neighbourhood within the VNS, aiming to change the structure and at the same time to improve the quality of the obtained solution. All this process is accomplished in the *IPRuinAndRecreate()* block. To ensure a sufficient diversification through the search process, and therefore, not being stuck in local optima, some low-penalty parts of the solution might also be destroyed and recreated randomly (e.g. within the *random* configuration, which will be explained later). The final obtained solution is again imported to the VND search algorithm and this process continues until some stopping criteria are met. Ultimately, the attained solution is further improved by applying IP to the whole problem instance to ensure a global search until the overall time limit is reached (*IPImprove()*).

Next, we elaborate on each of the main components of the hybrid algorithm, i.e. initial solution construction in *GreedyHeuristic()* block, VND search algorithm in *VNDSearch()* block, and IP ruin-and-recreate framework in *IPRuinAndRecreate()* block.

3.1. Initial solution construction

In this block, a greedy heuristic search is employed to generate an initial solution for the VND algorithm. Empirically, we have observed that having a high-quality initial solution reduces the efficiency of the VNS algorithm subsequently. For the same reason, a random initialisation often results in poor performance due to the very low-quality of the generated solution. Therefore, we decide to apply a simple greedy heuristic algorithm. The pseudo-code of this algorithm is depicted in Algorithm 2. After pre-processing of the problem data and creating required data structures, an empty solution (roster) is created. Starting from a randomly selected nurse, at the first step, we set all the pre-defined days off for the current nurse in *SetDaysOff()* block. In the next step, we randomly mark all the working days to which a specific shift needs to be assigned later (*AssignWorkDays()*), and then we assign a randomly selected shift to those days accordingly (*AssignShifts()*). Assigning shifts to the nurses within two different levels, i.e. first assigning working and non-working days, and then assigning shifts to the working days, helps us to only check the constraints related to each level independently, and hence, reducing the complexity of the constraint conflict resolution process. Having said that, in the first level, only the maximum number of working days, and the minimum and maximum number of consecutive shift types including day off shifts are checked. Therefore, in the next level, we only need to check the maximum number of shift types and avoid assigning shifts where a forbidden pattern is matched.

Algorithm 2: The pseudo code of the greedy heuristic algorithm to generate an initial solution.

```
Pre-process problem data;
Create an empty solution x = {x_i, i ∈ I};
foreach i ∈ I do
    while x_i is not feasible do
        x_i ← SetDaysOff(x_i);
        x_i ← AssignWorkDays(x_i);
        x_i ← AssignShifts(x_i);
        p_1 = EvaluateWorkload(x_i);
        p_2 = EvaluateWeekend(x_i);
        if p_1 + p_2 > 0 then
            x_i ← Destroy(x_i);
        end
    end
end
return [x]
```

Finally, to ensure satisfying the remaining constraints, i.e. the maximum number of worked weekends, and the minimum and maximum total times of assigned shifts, we calculate the associated incurred penalties (indicated as p_1 and p_2, computed within the *EvaluateWeekend()* and *EvaluateWorkload()* blocks, respectively). If there is any associated penalty, we destroy the current schedule for the current nurse by unassigning all the allocated shifts using the *Destroy()* block, and repeat the process until a feasible solution is obtained. This process is iterated for all the nurses until all the required shifts are assigned to all the days within a schedule, while satisfying all the hard constraints. According to our experiments, the greedy heuristic is able to produce a feasible solution for all the problem instances in up to 100 cycles per nurse. Finally, the generated feasible solution is returned to the VND algorithm for further improvement.

3.2. Variable neighbourhood descent

When there is an initial solution, either generated using the greedy heuristic algorithm in the *GreedyHeuristic()* block or passed from the previous cycle of the hybrid algorithm to the current one (in Algorithm 1), the Variable Neighbourhood Descent (VND) algorithm is applied to refine the solution locally according to a set of distinct neighbourhoods (Hansen & Mladenovic, 2001). In the VND search algorithm, a best-improvement descent local search algorithm is applied through cycling a set of neighbourhoods until no improvement can be found in all the neighbourhood structures, or when the total number of iterations is reached a certain maximum value. The reason to choose VND is that it is capable of exploring a variety of different-structured solutions throughout the search space due to applying a set of different neighbourhoods, which makes it a very suitable candidate for solving highly-constrained problems such as NRPs. Apart from the successful implementation of VND in the relevant literature (Burke et al., 2010; Stølevik et al., 2011), it is very easy to incorporate sophisticated neighbourhood structures such as our ruin-and-recreate framework, which is needed to successfully implement the hybrid algorithm.

VNS as a generalised VND approach is a relatively recent metaheuristic approach based on the simple idea of systematically changing neighbourhoods both to escape from the areas which contain local optima and within a local search to identify better local optima (Hansen & Mladenovic, 1999; 2001). It has been applied to many $\mathcal{NP}$-hard problems including NRPs (Burke et al., 2010; Stølevik et al., 2011). In a simple VNS scheme, a local search is applied to the incumbent solution using a neighbourhood structure until certain criteria such as the total number of iterations are met. Then the local search is restarted using a different neighbourhood structure, trying to improve the best solution obtained from the previous iterations. This process continues until there is no more improvement gained from any of the neighbourhood structures. The neighbourhood structures in a VNS are often selected to drive the search process towards different desired objectives, or to investigate different structures of the obtained solution in order to diversify the search process, and therefore, to avoid being stuck in local optima. In Algorithm 3 , the pseudo code of the applied VND algorithm is presented.

Algorithm 3: The pseudo code of the VND algorithm.

Define a set of neighbourhood structures N_k, $k = 1, \dots, k_{max}$;
Create an initial solution x;
Set $k \leftarrow 1$;
while $k < k_{max}$ **do**
  Explore the neighbourhood N_k of x;
  Find the best solution x' in N_k;
  if $x' < x$ **then**
    Set $x \leftarrow x'$;
    Set $k \leftarrow 1$;
  else
    Set $k \leftarrow k + 1$;
  end
end
return *[the best solution found]*

The following neighbourhoods are applied to the VND block of the hybrid algorithm (*VNDSearch()*):

1. *2-Exchange:* this neighbourhood consists of all moves, where two shifts are swapped between two different nurses on the same day.
2. *3-Exchange:* it includes all moves, where three (or more) shifts are exchanged between three (or more) different nurses on the same day.
3. *Double-Exchange:* it includes all moves that swap two shifts between two different nurses on two different days. In fact, this neighbourhood is made from two different *2-Exchange* neighbourhoods applying on two consecutive days.
4. *Multi-Exchange:* this neighbourhood is very similar to *Double-Exchange* but three (or more) shifts are swapped between two different nurses on three (or more) different days. Indeed, this neighbourhood is made from three (or more) different *2-Exchange* neighbourhoods, which are not necessarily applied on consecutive days.
5. *Block-Exchange:* this neighbourhood includes all moves where a specific number of consecutive shifts is swapped between two different nurses within the planning period.

Apart from *Multi-Exchange* neighbourhood, which is our new neighbourhood structure, the rest of the defined neighbourhoods are used in many local search algorithms in the literature (Burke et al., 2010; Stølevik et al., 2011). The *Multi-Exchange* neighbourhood is defined to overcome the complex structure of the problem, and therefore, to overcome the potential complicated local optima by applying some simple *2-Exchange* moves simultaneously. In fact, on the one hand, this neighbourhood helps to break complicated structures for a number of constraints such as HC7 and HC8. On the other hand, it is helpful to move good shift patterns from one nurse to another. Experimentally speaking, this neighbourhood structure gives better performance rather than a simple *2-Exchange* or *Double-Exchange*. The defined neighbourhoods are illustrated on a weekly roster for five nurses in Fig. 1, where *E*, *L*, and *N* indicate early, late, and night shifts, respectively, and all blank shifts are days off. In this figure, the swaps of shifts between nurses 1 and 3 are examples of *2-Exchange* (Tuesday), *Multi-Exchange* (Tuesday, Saturday, and Sunday), and *Double-Exchange* (Saturday and Sunday) neighbourhoods. As an example of *3-Exchange* neighbourhood, the three shifts between nurses 1, 4, and 5 on Wednesday can be sequentially swapped. Moreover, swapping the blocks of shifts from Friday to Sunday between nurses 4 and 5 can be an instance of a *Block-Exchange* neighbourhood.

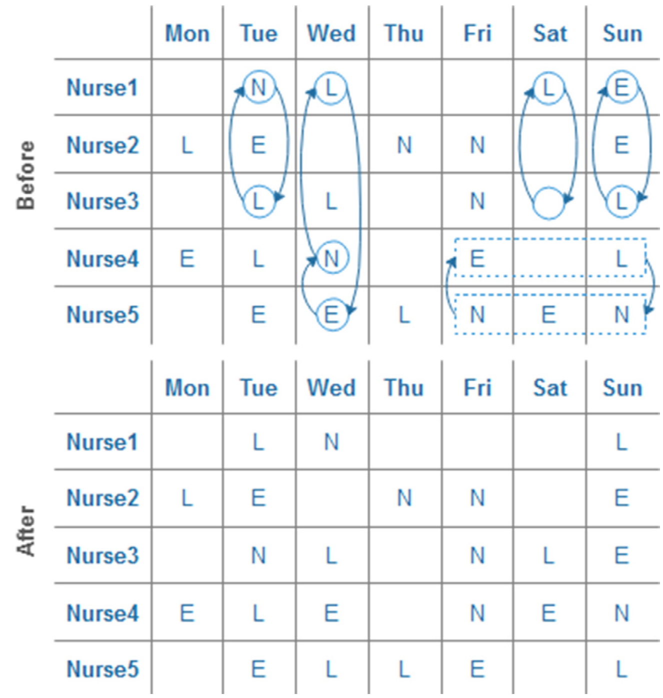

Fig. 1. Examples of 3-Exchange (Nurses 1, 4, and 5 on Wednesday), Block-Exchange (Nurses 4 and 5 from Friday to Sunday), Multi-Exchange (Nurses 1 and 3 on Tuesday, Saturday, and Sunday), 2-Exchange (Nurses 1 and 3 on Tuesday), and Double-Exchange (Nurses 1 and 3 on Saturday and Sunday) neighbourhoods applied in the VND algorithm.

3.3. IP ruin-and-recreate framework

If the VND search algorithm could not find any better solutions by cycling through the set of neighbourhoods or reaches a maximum number of iterations, the incumbent solution is passed to an Integer Programming solver as a perturbation neighbourhood structure within the VNS. The IP solver searches for a better alternative solution based on the IP model of the problem introduced in Section 2, by fixing the low-penalty parts of the solution and exploring all the remaining possibilities to find a higher-quality solution in an iterative manner (*ruin-and-recreate* framework). Throughout this process, two possible outcomes may happen: (1) the IP solver can find a better solution, which can be a different solution in terms of the underlying structure in comparison with the last one. In this case, the IP solver helps the VND algorithm both in terms of intensification and diversification. (2) the IP solver cannot produce a better-quality solution (and in some cases, even produce a worse-quality solution) due to the timeout criterion or due to the non-existence of a better alternative. In this case, the IP solver may produce a solution with a different structure, and hence, helps the search process only in terms of diversification. In either case, the role of the embedded IP component in the hybrid algorithm is essential, where solving the problem only using a pure heuristic approach often results in poor performance. It is noteworthy to mention that by destroying parts of the solution and repairing it again, there is no guarantee of the quality and structure of the new solution. In terms of diversification, generating a solution with a different structure is crucial. The structure of the solution is different, if it cannot be obtained by searching through the defined neighbourhoods of the current solution and some nearby solutions. Indeed, a solution is different from the other solutions in terms of the underlying structure, if we cannot generate it by iteratively applying the defined neighbourhoods for a sufficient number of iterations. In the literature, a ruin-and-recreate strategy either by IP or CP is mostly employed in order to diversify the search process and to perturb the obtained solution. For example, Stølevik et al. (2011) have applied this strategy by destroying parts of the solution and then using CP to rebuild it. Li, Aickelin, and Burke (2009) have also used this strategy by the evolutionary elimination of parts of the solution and subsequently repairing it by using a greedy heuristic. A more advanced ruin-and-recreate based algorithm is also reported in Li, Bai, Shen, and Qu (2015), where the authors applied a stochastic modelling and Markov chain analysis. Nonetheless, in the proposed hybrid algorithm, we apply this strategy not only for the diversification purpose but also for improving the quality of the obtained solution, i.e. intensification. This is the main reason that we select IP for ruin-and-recreate framework compared with CP and other heuristics, which is able to improve the current solution, and at the same time, investigate many areas within the search space. Moreover, we have empirically observed that destroying parts of the solution is more effective than using sophisticated neighbourhoods or similar techniques.

Another novel aspect of our ruin-and-recreate framework is due to applying a flexible generic scoring scheme to evaluate different parts of the solution, which allows us to adaptively focus on those areas of the solution, which has a higher probability to generate a better solution, if they are changed. In order to fix some parts of the solution, we apply a scoring scheme by assigning a value to each cell within the roster, where each cell is an intersection of one particular day and one particular nurse. In fact, using the scoring scheme, the total penalty associated with the current solution can be broken down to the fundamental elements of the problem. It means a shift can be assigned to each cell and if so, there is an associated penalty according to the objective function and the constraints that are involved. In other words, each cell can be designated by a value, which is the proportion of (an assignment to) that cell of the total number of violations respecting to the current solution. We call this estimated value as *cell penalty*. Cell penalties can be easily aggregated to different dimensions, therefore providing an insightful tool to analyse and discriminate different parts of the solution. For example, in *Random* configuration which we explain later in this section, this value is used as a weight in a simple linear weighted random function, where a random cell is selected in order to be destroyed later.

Next, we demonstrate how to calculate a cell penalty, which is calculated based on the total incurred violations of the constraints involved defined in Section 2. It should be noted that although this calculation is not accurate in general, it is sufficient for our purpose. Here, we consider all the constraints either hard or soft, which might be violated throughout the search process using the hybrid algorithm, and hence, we also define the violations associated with hard constraints. To determine the relevant weights of hard constraints, a significant value (here, 1000 for constraints *HC4* and *HC5* and 10,000 for the remaining hard constraints) is selected to ensure that the final solution is feasible by directing the algorithm to feasible regions. Table 1 shows for each constraint the assigned weight (w_c), the relevant violation of the constraint, cell share, i.e. the relevant proportion of the constraint violation for all the affected cells (s_c), and the affected cells associated with the violation, respectively. In this table, $|D|$, $|V|$, $|W|$ and $|I|$ denote the total number of days within the planning period, the total number of violations relevant to a constraint, the total number of weekends, and the total number of nurses, respectively. All the other parameters are already defined in Section 2.

To calculate the cell penalty (p_{cell}) for a particular cell, we need to multiply the amount of cell share (s_c) with the relevant weight (w_c) for each constraint ($c \in C$) for a particular day and nurse. Hence, the total penalty allocated to each cell can be calculated using the following equation:

$$p_{cell} = \sum_{c \in C} s_c w_c$$

For example, for constraint *HC2*, assuming that we have only one violated rotation for a specific nurse occurred on Tuesday and Wednesday in the first week of a roster, the calculated penalty for each of the two cells involved in the violated rotation is equal to 5000. It should be noted that for constraint *SC2*, because the minimum and maximum number of shifts per day are given, we need to sum up the total number of violations for all shifts ($t \in T$) in order to calculate the cell penalty. It is noteworthy to mention that the penalty evaluation is quite fast since it is done using a delta function (i.e., we only calculate the difference of total penalties between two solutions according to the violated constraints) and highly optimised data structures.

Calculating and estimating cell penalties in a schedule, we are also able to accumulate them over different dimensions such as nurses and days in the schedule. In fact, by accumulating the cell penalties, we can elicit more information, which gives us more insights into the destroying and recreating processes. Having said that, we use the following *aggregation settings* to configure the IP solver in order to fix different parts of the solution during the search process. In Section 4, we will test the hybrid algorithm by combining the following settings together within different configurations in order to identify the best efficient one.

1. *Nurses:* by accumulating cell penalties within the planning horizon for each nurse, we are able to identify the contribution of each nurse in the total penalty respecting to the current solution. Therefore, we can recognise nurses who have the most contributed penalties among the others.
2. *Days:* in this setting, cell penalties are accumulated for all the nurses within each day. Therefore, similar to *Nurses* setting, we can identify the days with the most contributed penalties.
3. *Weeks:* analogous to the other settings, here cell penalties are accumulated for all nurses and all days within each week.
4. *Random:* in this setting, there is no accumulation indeed. Instead, cells are selected randomly according to their relevant contributed penalty. For this purpose, a simple linear weighted random function is used, where cells with a higher penalty have more chance to be selected.

For illustration purposes, Fig. 2 shows the first week of a roster, in which the cell penalties are calculated for all the involved constraints. It can be seen that for some cells, there are not any associated penalties (blank cells), which means they are not contributed to the total incurred penalty of the current solution. For the cell at the intersection of *Monday* and *Nurse5*, the calculated cell penalty is 150, which is the highest value among all the cells. Therefore, in the Random aggregation setting, this cell is very likely to be selected and unassigned afterwards. If we aggregate the cell penalties for all the nurses and days as calculated in the last column and row of the weekly roster, it is realised that *Monday* and *Nurse2* (shown as underlined style) have the greatest contributions to the total penalty associated with the roster. Therefore, using the Nurses and Days aggregation settings, *Monday* and *Nurse2* are selected for being destroyed. Consequently, it can be seen that much

Table 1
The required information to calculate cell penalties for all the constraints.

Ct.	Weight (w_c)	Violation	Cell share (s_c)	Affected cells
HC2	10,000	Number of forbidden shift rotations	$1/2\|V\|$	All the involved cells
HC3	10,000	Number of shifts more than the maximum value	$1/\|D\|$	All cells
HC4	1000	Number of minutes more than the maximum value	$1/\|D\|$	All cells
HC5	1000	Number of minutes less than the minimum value	$1/\|D\|$	All cells
HC6	10,000	Number of shifts more than the maximum value in an isolated sequence of shifts	$\|V\|/c_i^{max} + \|V\|$	All the involved cells
HC7	10,000	Number of shifts less than the minimum value in an isolated sequence of shifts	$\|V\|/c_i^{min} + \|V\|$	All the cells on two sides of the isolated sequence of shifts
HC8	10,000	Number of days off less than the minimum value in an isolated sequence of shifts	$\|V\|/o_i^{min} + \|V\|$	All the cells on two sides of the isolated sequence of days off
HC9	10,000	Number of weekends more than the maximum value	$\|V\|/2\|W\|$	All the involved cells
HC10	10,000	One assigned shift	1	The involved cell
SC1	1	One shift (un-)assignment	1	The involved cell
SC2	$w_{dt}^{min}, w_{dt}^{max}$	Number of deviations from the minimum and maximum values	$\Sigma_{t \in T}\|V\|/\|I\|$	All the involved cells (for all nurses)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun	SUM
Nurse1	100							100
Nurse2	140	50	50		50	50		340
Nurse3	100		10	10	10	10	10	150
Nurse4	100		70	70				240
Nurse5	150	40				15	15	220
SUM	590	90	130	80	60	75	25	1050

Fig. 2. The associated cell penalties calculated for the first week of a roster.

useful information can be extracted from a solution after calculating the relevant cell penalties.

When the candidate cells required to be fixed (or to be destroyed) are identified, the IP solver solves the problem using the incumbent solution, where the integer variables associated with the fixed cells (x_{idt}) are set before starting the search process. Next, considering all constraints as soft except *HC1*, the IP solver produces another solution which can be different from the current solution in terms of quality and the underlying structure. In either case, the generated solution is passed to the next iteration of the VND algorithm as an initial solution. It is noteworthy to mention that because a significant number of variables in the IP model is fixed, particularly those which are involved in more constraints, the IP solver can easily solve the problem even when the scale of the problem instance is relatively large. Therefore, according to our experiments, in most cases, the IP solver can produce a solution even in a very short timeout condition.

Ultimately, after running the VND search algorithm and the IP ruin-and-recreate blocks in order to find a better solution, there is still a chance of being stuck in a local optimum. In fact, due to not having any global picture throughout the search space, there can be another solution close to the current local optimum, but not detectable due to the complex structure of the problem. To resolve this issue, another IP solver improves the best-found solution at the end of the hybrid algorithm to solve the problem in the remaining time, and then the final solution is reported to the user. This IP solver employs the same IP model introduced in Section 2 including all the constraints, but it starts from the best-found solution thus far. To configure the IP solver to start from the current best solution rather than a randomly generated roster, the appropriate parameter (e.g. *MIPFocus* parameter in Gurobi) is set before starting the search process. It should be noted that the final IP solver is also useful to provide some insights to the optimality of the current solution, which makes the hybrid algorithm a pseudo-exact method.

4. Computational results

We tested the proposed hybrid algorithm on 24 instances that have been recently introduced by Curtois and Qu (2014), and then on 12 randomly public generated instances introduced in this paper (Rahimian, 2015), which will be explained later. Table 2 summarises the characteristics of the first set of benchmark instances. Despite other extensively studied instances in the literature (Burke, Curtois, Qu, & Berghe, 2008), which are based on models and assumptions that are different than ours, and are mostly easy to solve, these instances are emphasised to be challenging for the state-of-the-art algorithms. Moreover, these instances are varied in terms of complexity and size, which makes them an appropriate benchmark for the proposed algorithm.

The variety of the benchmark instances with different characteristics and structures allows us to test the hybrid algorithm thoroughly. Generally speaking, moving from *Instance01* to *Instance24*, the computational difficulty of the problem instances increases, which often requires spending more time and computer memory. In particular, the last five instances are computationally challenging due to their huge sizes. Having said that, the number of days off and shift on/off requests (columns 5 and 6 in Table 2) can be appropriate indicators for the difficulty of the problem instances. In general, according to our experiments, the more the number of requests, the more difficult to deal with a problem instance both in IP and heuristic algorithms.

We conducted our tests on a PC (Intel Core-i5 3.4 gigahertz with 4 gigabytes RAM) running Windows 7. We implemented the hybrid algorithm in Java 1.7 (De Beukelaer, Davenport, De Meyer, & Fack, 2015) and employed Gurobi (Gurobi Optimization, 2015) 5.6 as the IP solver. We also made a concerted effort to optimise

Table 2
The characteristics of the first set of benchmark instances.

Instance	Days	Nurses	Shift types	Day off requests	Shift on/off requests
Instance01	14	8	1	8	26
Instance02	14	14	2	14	62
Instance03	14	20	3	20	64
Instance04	28	10	2	20	71
Instance05	28	16	2	32	106
Instance06	28	18	3	36	135
Instance07	28	20	3	40	168
Instance08	28	30	4	60	225
Instance09	28	36	4	72	232
Instance10	28	40	5	80	284
Instance11	28	50	6	100	336
Instance12	28	60	10	120	422
Instance13	28	120	18	240	841
Instance14	42	32	4	128	359
Instance15	42	45	6	180	490
Instance16	56	20	3	120	280
Instance17	56	32	4	160	480
Instance18	84	22	3	176	414
Instance19	84	40	5	320	834
Instance20	182	50	6	900	2318
Instance21	182	100	8	1800	4702
Instance22	364	50	10	1800	4638
Instance23	364	100	16	3600	9410
Instance24	364	150	32	5400	13,809

the implemented code using the latest software technologies and code optimisation practices. For example, we used efficient hash algorithms and appropriate data structures similar to the ones available in Boost C++ libraries, and generic programming to minimise performance overheads. In all the experiments, the algorithm was run three times and the best-obtained solution is reported. Moreover, we run all our experiments on one CPU core to have a fairer and more accurate comparison. After extensive testing of the algorithm using different settings, the following parameters were set. We dedicate 70% of the total allowed runtime to the VNS algorithm (VND and IP ruin-and-recreate components), and the rest, to the final IP block. For VND stopping criteria, we set the maximum number of iteration to 50,000 and the maximum number of non-improvement iterations to 5. We also employed the neighbourhoods 2-Exchange, Double-Exchange, Multi-Exchange with the length of 3, Block-Exchange with the length of 4, and 3-Exchange with the length of 3 in order.

Three experiments were carried out to test the proposed algorithm: first, we investigate different aggregation settings through the ruin-and-recreate framework to understand how they affect the performance of the algorithm, and then we analyse the performance of different components of the hybrid algorithm. Second, we compare the hybrid algorithm with two state-of-the-art algorithms, which currently generate the best results for the studied instances and most existing instances in the literature, and the Gurobi IP solver. Third, due to the unavailability of a benchmark dataset in the literature which would have allowed us to compare the hybrid algorithm with another similar method (Burke et al., 2010), we create 11 randomly generated instances to further benchmark the hybrid algorithm against.

In the first experiment, in order to examine the best combination of aggregation settings defined in Section 3 for the IP ruin-and-recreate component, we ran the algorithm with a variety of combinations. For each combination, we define a *selection range*, i.e. the total percentage of available candidates in each aggregation setting which is selected in decreasing order to be destroyed and recreated. According to our non-exhaustive preliminary tests on a variety of combinations using the benchmark instances in Table 2, in the following, we present the best three identified *configurations* and the relevant selection ranges:

Table 3
Results of the hybrid algorithm by applying different configurations of aggregation settings.

Instance	Config1		Config2		Config3	
	Obj.	Gap (%)	Obj.	Gap (%)	Obj.	Gap (%)
Instance08	1958	33.76	1695	23.48	1364	4.91
Instance09	439	7.52	439	7.52	439	7.52
Instance10	4631	0.00	4631	0.00	4631	0.00
Instance11	3443	0.00	3443	0.00	3443	0.00
Instance12	4045	0.12	4045	0.12	4042	0.05
Instance13	3109	56.71	3109	56.71	3109	56.71
Instance14	1361	6.17	1342	4.84	1281	0.31
Instance15	4463	14.72	4588	17.04	4144	8.16
Instance16	3384	4.73	3306	2.48	3306	2.48
Instance17	5956	3.86	6043	5.25	5760	0.59
Instance18	5158	15.65	5158	15.65	5049	13.82
Instance19	4365	32.53	4145	28.95	3974	25.89
Instance20	5451	12.99	5603	15.35	5242	9.52
Instance21	27,281	23.51	28,356	26.41	24,977	16.45
Instance22	176,652	86.38	173,371	86.12	130,107	81.50
Instance23	57,210	95.17	97,893	97.18	40,543	93.18
Instance24	3,173,810	97.74	3,160,760	97.73	2,829,680	97.46
Average		28.91		28.51		24.62

- *[Config1]*: Use only Nurses aggregation setting with the selection range of at least 20%.
- *[Config2]*: Apply Nurses, Days, and Weeks aggregation settings in order, with the selection ranges of at least 20%, 50%, and 60%, respectively.
- *[Config3]*: Apply Nurses, Days, Weeks, and Random aggregation settings in order, within the selection ranges of [10%, 30%], [10%, 40%], [10%, 50%], and [30%, 50%], respectively. For each setting, the minimum value in the relevant range is increased by 10% (the increment rate) after each VNS iteration (e.g. [10%, 30%] is changed to [20%, 30%]).

Table 3 shows the results of the benchmarked configurations for instances 8–24, where the algorithm is run for 10 minutes. We do not report the results for the first seven instances, since they are not very complicated for the proposed hybrid algorithm, and hence, it returns the same results for all the mentioned configurations. In this table, the objective function value and its difference to the best-known lower bound in percentage (denoted as Gap (%)) according to Curtois and Qu (2014) are shown for each configuration and instance. One can see that running the algorithm with the third configuration generally results in better solutions in average. The reason for the superiority of the third configuration is due to the comprehensive investigation of the solution space by using different ruin-and-recreate strategies, and as a result, facilitating the hybrid algorithm to escape from a variety of local optima. In fact, in this configuration, we re-evaluate the current solution through four different dimensions after being stuck in each local optimum. Moreover, changing the selection ranges incrementally, equips the hybrid algorithm to behave adaptively during the search progress. It means, the more the hybrid algorithm advances within the search process, the more parts of the solution are selected to be changed, i.e. the diversification rate is being increased.

Using the third configuration (i.e. Config3), we run the algorithm for all the benchmark instances for 10 minutes computational time. The detailed results of this test are shown in Table 4, where the initial solution generated by the greedy heuristic algorithm, the improved solution by the VNS, and the final solution further improved by the IP solver are reported in average, respectively. In this table, Δ_{iv}%, and Δ_{vo}% denote the percentage of improvement achieved using the VNS component and the final IP block, respectively. Furthermore, the number of cycles and the average improvement obtained throughout each cycle (denoted as

Table 4
Detailed results of the hybrid algorithm by applying the third configuration for 10 minutes.

Instance	Initial	Δ_{iv}%	VNS	Δ_{vo}%	Obj.	Cycle	$\frac{\Delta}{C}$%
Instance01	11,525	94.73	607	0.00	607	18	5.26
Instance02	14,827	94.42	828	0.00	828	30	3.15
Instance03	22,480	95.55	1001	0.00	1001	16	5.97
Instance04	20,775	91.74	1716	0.00	1716	56	1.64
Instance05	31,947	96.41	1147	0.35	1143	48	2.01
Instance06	30,944	93.40	2041	4.46	1950	52	1.80
Instance07	42,625	97.49	1070	1.31	1056	52	1.88
Instance08	63,153	95.98	2538	46.26	1364	81	1.21
Instance09	55,320	99.21	439	0.00	439	42	2.36
Instance10	102,073	94.97	5133	9.78	4631	19	5.02
Instance11	120,287	97.13	3450	0.20	3443	25	3.89
Instance12	146,970	96.05	5801	30.32	4042	77	1.26
Instance13	289,121	98.88	3231	3.78	3109	49	2.02
Instance14	117,166	98.19	2116	39.46	1281	79	1.25
Instance15	144,631	96.37	5245	20.99	4144	60	1.62
Instance16	105,714	95.40	4861	31.99	3306	81	1.20
Instance17	174,308	95.99	6986	17.55	5760	67	1.44
Instance18	181,068	96.79	5815	13.17	5049	50	1.94
Instance19	322,730	98.59	4564	12.93	3974	47	2.10
Instance20	910,083	99.42	5242	0.00	5242	36	2.76
Instance21	197,130,000	99.99	26,989	0.04	26,977	44	2.27
Instance22	168,433,000	99.92	130,107	0.00	130,107	42	2.38
Instance23	15,542,000	99.74	40,543	0.00	40,543	45	2.22
Instance24	201,119,000	98.55	2,925,411	3.27	2,829,680	17	5.80
Average		96.87		9.83			2.60

$\frac{\Delta}{C}$%) are reported in the last two columns of Table 4. As it can be seen, the VNS algorithm is able to improve the generated initial solution by the greedy heuristic by 97%, which then is further optimised by the final IP block by 10% in average. Moreover, it is observed that the final IP solver is not able to improve the generated solution for a number of instances. For example, for *Instance02*, the IP solver is only employed to prove the optimality of the obtained solution by the VNS algorithm. However, for some instances such as *Instance23*, the IP solver does not manage to produce any better solutions due to the limited computational time. Nevertheless, the role of the final IP solver as the last component of the hybrid algorithm in order to improve the output of the VNS algorithm is crucial, where the attained improvement can be even reached more than 30% for some instances.

In the second experiment, in order to benchmark the efficiency of the proposed algorithm with the current state-of-the-art algorithms, we compared it with the results published in Curtois and Qu (2014), where the authors report the results of two algorithms from Burke and Curtois (2014), i.e. a branch-and-price and an ejection chain heuristic. All the published benchmark results were run on Intel Core2 Duo 3.16 gigahertz with 8 gigabytes RAM. Unfortunately, we could not find any other benchmark results, and to the best of our knowledge, at this time the two benchmark algorithms produce the best results (Burke & Curtois, 2014) for the studied problem instances and most existing instances in the relevant literature. To have a fair comparison, the algorithm is run only on one core of CPU and employs the same version of Gurobi solver. All our experiments are given a computational time of 10 minutes, since the hybrid algorithm is particularly designed to perform well in short computational times, and also it is common to use short times, as seen in the relevant literature and the nurse rostering competition (Haspeslagh, De Causmaecker, Schaerf, & Stølevik, 2014). However, to have a comprehensive comparison with the available results and the benchmark algorithms, we also run the proposed algorithm for a longer time, i.e. 60 minutes. Table 5 presents the best results from the ejection chain method, Gurobi IP solver with default settings, and our hybrid algorithm using the third configuration (Config3 in Table 3) running for the limited computational time of 10 and 60 minutes, respectively. The results for the branch-and-price (B&P) algorithm without any time limits are also presented. In this table, "–" indicates that the algorithm does not generate any feasible solutions within the allocated time limit.

As we can see in Table 5, within the 10 minutes computational time, from the total of 24 instances, the hybrid algorithm outperforms the ejection chain method for 23 instances, and produces the same results for *Instance01*. In comparison with the Gurobi IP solver, the algorithm performs better in 14 instances and generates the same results for the remaining 10 instances, where 9 of them are optimal solutions (shown as underlined style). In overall, the proposed algorithm outperforms the ejection chain method and Gurobi IP solver within 10 minutes computational time for 14 instances (shown as bold style), and produces the same or better results for the rest of the instances. It should be noted that the ejection chain method and Gurobi IP solver could not solve the last 3 and 5 instances, respectively. Having said that, obtaining the reported solutions for these instances, which are very hard to solve and huge in size, make the hybrid algorithm an appropriate candidate to tackle such instances even in a very short runtime.

Running our benchmarks for the longer runtime of 60 minutes, the hybrid algorithm outperforms the ejection chain method for 22 instances and does not generate a better result only for *Instance24*. The reason of obtaining a poor-quality solution for *Instance24* might be the inherent nature of the hybrid algorithm as a pseudo-exact method (matheuristic). Since this instance is huge in size, it is a challenge for the algorithm to solve it in comparison with a meta-heuristic approach like ejection chain method. Moreover, it might have a particular structure which cannot be exploited using the current setting of the proposed algorithm, but easy to be identified by the ejection chain method. Similarly, in comparison with Gurobi IP solver, the hybrid algorithm is able to generate better results for 13 instances and obtains the same results for the remaining 10 instances. For *Instance08*, the IP solver outperforms the hybrid algorithm for only a slight difference. In overall, the hybrid algorithm attains better solutions for half of the instances (shown as bold style), which makes it a successful candidate even for longer computational times.

Table 5
The benchmark results for the hybrid algorithm in comparison with the best current algorithms including branch-and-price and ejection chain heuristic (Burke & Curtois, 2014), and Gurobi IP solver (Gurobi Optimization, 2015) running for 10 and 60 minutes.

Instance	10 minutes			60 minutes			
	Hybrid algorithm	Ejection chain	Gurobi	Hybrid algorithm	Ejection chain	Gurobi	B&P
Instance01	607	607	607	607	607	607	607
Instance02	828	923	828	828	837	828	828
Instance03	1001	1003	1001	1001	1003	1001	1001
Instance04	1716	1719	1716	1716	1718	1716	1716
Instance05	1143	1439	1143	1143	1358	1143	1160
Instance06	1950	2344	1950	1950	2258	1950	1952
Instance07	1056	1284	1056	1056	1269	1056	1058
Instance08	**1364**	2529	8995	1344	2260	**1323**	1308
Instance09	439	474	439	439	463	439	439
Instance10	4631	4999	4631	4631	4797	4631	4631
Instance11	3443	3967	3443	3443	3661	3443	3443
Instance12	**4042**	5611	4045	4040	5211	4040	4046
Instance13	**3109**	8707	500,410	**1905**	3037	3109	–
Instance14	**1281**	2542	1482	**1279**	1847	1280	–
Instance15	**4144**	6049	78,144	**3928**	5935	4964	–
Instance16	**3306**	4343	3521	**3225**	4048	3233	3323
Instance17	**5760**	7835	6149	**5750**	7835	5851	–
Instance18	**5049**	6404	7950	**4662**	6404	4760	–
Instance19	**3974**	6522	29,968	**3224**	5531	5420	–
Instance20	**5242**	23,531	–	**4913**	9750	–	–
Instance21	**26,977**	38,294	–	**23,191**	36,688	–	–
Instance22	**130,107**	–	–	**32,126**	516,686	–	–
Instance23	**40,543**	–	–	**3794**	54,384	–	–
Instance24	**2,829,680**	–	–	2,281,440	**156,858**	–	–

Comparing with branch-and-price (B&P) algorithm without any time limit, the hybrid algorithm is outperformed only for *Instance08*, where B&P takes more than 197 minutes to generate the solution. Apart from the 11 instances, for which B&P cannot produce any results, the hybrid algorithm beats B&P for 5 instances and achieves the same results for the rest of the instances.

In the third experiment, we try to compare the hybrid algorithm with a similar approach reported in Burke et al. (2010), in which IP and VNS are hybridised in a pipeline fashion, i.e. running sequentially. The authors also developed a decomposition technique for handling constraints, and evaluated their hybrid VNS using the studied decomposition. Unfortunately, after making inquiries from the relevant authors, it is found out that the benchmarked dataset including 12 instances is lost except one of them, i.e. the first instance for January. Being unsuccessful in obtaining the benchmarked instances and willing to further evaluate the efficiency of the hybrid algorithm, we randomly generated 11 instances (instead of the 11 extinct ones) attempting to make them similar to the sole existing instance we already have. These instances are made publicly available (Rahimian, 2015) to facilitate other researchers to benchmark their algorithms. The problem description regarding the remaining available instance called *ORTEC01* is accessible in Burke et al. (2008), and the associated IP formulation is reported in Rahimian et al. (2015). As we tried to generate instances resembling the only available instance as closely as possible, we made the following assumptions:

1. Since the lost instances belong to a yearly dataset extracted from a real hospital over 12 months, we assumed that all the staff contracts are not changed and fixed during the planning year.
2. Considering the yearly nature of the dataset, we assumed that there is not any change in the hospital regulations, number of shifts, and number of nurses (i.e. no hiring or firing occurs).
3. Considering the yearly nature of the dataset, we assumed that there should not be any major changes in the coverage and shift on/off requests.
4. We assumed that the coverage data for the weekend days follow a similar pattern to the coverage data of the available instance.

Based on these assumptions, we only generate random instances by changing the coverage and shift on/off requests constraints. For generating coverage data, for all the weekdays and for all the shifts except night, we use a weighted uniform random function within the range of [2, 4], by considering the associated weights of 0.25, 0.5, 0.25 for the included numbers within the range. For night shifts, we use a uniform random function to generate the coverage data within the range of [1, 2]. To have similar coverage data compared with the available data, we also try to keep the difference between the total sum of all the coverages during the planning horizon less than 40 for generated instances and the available instance. Thus, we ensure that the generated coverage data are similar to the available instance with only very slight perturbation.

For generating shift on/off requests, first, we use a uniform random function to generate some request data including the involved employees, the requested shifts consist of days off, the requested days, and the associated weights, while considering ranges of [0, total number of employees], [0, total number of shifts + 1], [0, total number of days], and the set of {100, 1000, 10,000}, respectively. Then we use a uniform random function again to generate the required number of shift on/off requests within the range of [0, 5] independently. Finally, knowing the total number of shift on/off requests, first we pick the number of shift on requests and then the number of shift off requests from the generated request data, if any.

We use an identical random seed for the whole generation of the instances, and we repeat the process until we obtain a feasible problem instance. It is noteworthy to mention that although we try to generate the instances very similar to the one in hand, due to the complexity of the constraints and the importance of the coverage and shift on/off requests in the structure of the problem, the generated instances are different in terms of computational complexity and even more challenging than the available one as we see

Table 6
The characteristics of the generated random instances (second set of benchmark instances) and results of standard Gurobi IP solver for 10 minutes.

Instance	IP statistics			Gurobi		
	Constraints	Variables	RR iterations	Obj.	LB	Gap (%)
ORTEC01	*20,611*	*21,954*	*7280*	*1410*	*145*	*89.72*
ORTEC02	20,581	21,924	7426	15,500	570	96.32
ORTEC03	20,580	21,923	10,732	31,741	200	99.37
ORTEC04	20,586	21,929	9354	15,510	122	99.21
ORTEC05	20,582	21,925	9904	25,495	1300	94.90
ORTEC06	20,582	21,925	10,598	14,855	211	98.58
ORTEC07	20,581	21,924	9804	2911	138	95.26
ORTEC08	20,583	21,926	8358	5660	141	97.51
ORTEC09	20,581	21,924	7526	385	201	47.79
ORTEC10	20,577	21,920	8715	14,940	1130	92.44
ORTEC11	20,583	21,926	10,105	22,863	310	98.64
ORTEC12	20,580	21,923	9265	37,698	110	99.71

Table 7
Results of the hybrid algorithm by applying different configurations of aggregation settings for the second set of benchmark instances.

Instance	Config1		Config2		Config3	
	Obj.	Δ%	Obj.	Δ%	Obj.	Δ%
ORTEC01	*465*	*58.06*	*380*	*71.05*	*270*	*100.00*
ORTEC02	7770	98.97	8800	87.39	7690	100.00
ORTEC03	9900	99.00	9880	99.20	9801	100.00
ORTEC04	6510	97.39	7370	86.02	6340	100.00
ORTEC05	4090	98.29	4870	82.55	4020	100.00
ORTEC06	9507	99.40	9499	99.48	9450	100.00
ORTEC07	2396	99.33	2491	95.54	2380	100.00
ORTEC08	4390	76.99	4470	75.62	3380	100.00
ORTEC09	282	90.78	295	86.78	256	100.00
ORTEC10	4660	76.39	3670	97.00	3560	100.00
ORTEC11	8850	99.66	9030	97.67	8820	100.00
ORTEC12	3481	99.11	3451	99.97	3450	100.00
Average		91.12		89.86		100.00

Table 8
Detailed results of the hybrid algorithm by applying the third configuration for 10 minutes for the second set of benchmark instances.

Instance	Initial	Δ_{iv}%	VNS	Δ_{vo}%	Obj.	Cycle	$\frac{\Delta}{C}$%
ORTEC01	*74,611*	*98.27*	*1291*	*79.09*	*270*	*55*	*1.81*
ORTEC02	70,863	89.12	7711	0.27	7690	50	1.78
ORTEC03	76,169	87.02	9886	0.86	9801	50	1.74
ORTEC04	127,870	95.01	6380	0.63	6340	41	2.32
ORTEC05	87,018	95.35	4050	0.74	4020	53	1.80
ORTEC06	89,815	89.35	9561	1.16	9450	53	1.69
ORTEC07	67,417	96.44	2401	0.87	2380	54	1.79
ORTEC08	91,082	96.23	3431	1.49	3380	58	1.66
ORTEC09	70,700	98.07	1368	81.29	256	33	3.02
ORTEC10	82,975	95.63	3625	1.79	3560	63	1.52
ORTEC11	111,571	92.08	8840	0.23	8820	51	1.81
ORTEC12	88,373	96.04	3501	1.46	3450	57	1.69
Average		94.05		14.16			1.89

Table 9
The benchmark results for the hybrid algorithm in comparison with Gurobi IP solver (Gurobi Optimization, 2015) and the hybrid VNS algorithm (IPVNS) reported in Burke et al. (2010) for the second set of benchmark instances.

Instance	Hybrid algorithm		Gurobi		IPVNS
	10 minutes	60 minutes	10 minutes	60 minutes	60 minutes
ORTEC01	*<u>270</u>, 315*	*270*	*1410*	*405*	*460*
ORTEC02	7690	7620	15,500	11,162	–
ORTEC03	9801	9638	31,741	12,850	–
ORTEC04	6340	5230	15,510	8553	–
ORTEC05	4020	3700	25,495	13,385	–
ORTEC06	9450	9400	14,855	14,855	–
ORTEC07	2380	2320	2911	2521	–
ORTEC08	3380	3220	5660	5301	–
ORTEC09	256	230	385	241	–
ORTEC10	3560	3360	14,940	5880	–
ORTEC11	8820	8530	22,863	22,551	–
ORTEC12	3450	3290	37,698	5828	–

next. Table 6 summarises the characteristics of the generated random instances and the obtained results from the standard Gurobi IP solver for 10 minutes runtime. In the first part of this table, some statistics regarding the IP formulation of the generated instances including the number of constraints, variables, and simplex iterations to solve the root relaxation of the problem instances are presented, respectively. In the second part, the objective function value, lower bound, and the optimality gap resulting from running the IP solver for 10 minutes timeout condition are shown, respectively. The optimality gap is defined as the discrepancy between the value of the current feasible solution (for the primal problem) and the value of the lower bound (feasible for the dual problem). When the optimality gap is zero, the current feasible solution is an optimal solution. It should be noted that in Table 6, the instance *ORTEC01* is the instance we had available (shown in italic style), and the rest indicates the instances we generated.

As one can see in Table 6, by observing the results obtained from the IP solver for 10 minutes runtime, all the generated instances are solved to a gap greater than 90% except instance *ORTEC09* with the gap of 48%. Therefore, it can be seen that apart from the instance *ORTEC09*, all the other instances are even more difficult to solve rather than *ORTEC01*. As a result, we can argue that the randomly generated instances are difficult to solve and they can be a suitable benchmark dataset for evaluating the performance of the hybrid algorithm.

We run the hybrid algorithm for 10 minutes using the three configurations introduced in the first experiment. The results are reported in Table 7, where the absolute objective value and the associated normalised percentage (denoted as Δ%) are shown, respectively. Similar to the first experiment, we observe that running the algorithm with the third configuration results in a better solution for all the instances. In order to analyse the performance of the main components of the hybrid algorithm, we run it using the generated new benchmark instances for 10 minutes computational time. The detailed average results are reported in Table 8, where the VNS algorithm improves the initial solution by 94%, which is further enhanced by the final IP block by 14% in average. It should be noted that for generating the initial solution, a similar IP solver is run for 20 seconds, since employing a greedy heuristic similar to the one explained in Section 3.1 often results in infeasible solutions. To compare the performance of the hybrid algorithm with Gurobi IP solver and the hybrid VNS algorithm reported in Burke et al. (2010), we run our hybrid algorithm using the third configuration and report the results in Table 9, where the hybrid algorithm and IP solver are run for 10 and 60 minutes, and the hybrid VNS (shown as IPVNS) is run for 60 minutes computational time.

To have a fairer and more accurate comparison, we simulate the computational environment of IPVNS algorithm (Pentium 2.0 gigahertz PC) by running the hybrid algorithm on a different PC with an Intel Core-i7 1.6 gigahertz CPU but only using one core of the CPU. Having said that, the first reported value for instance *ORTEC01* is the one similar to the other instances by running on our regular benchmark PC, and the second one is relevant to the less-powerful PC used only for comparing with IPVNS algorithm. As we can see in Table 9, compared with the results obtained by the IP solver, the hybrid algorithm finds better solutions for all the instances. In particular, for instance *ORTEC01*, when we compare the results with IPVNS, the algorithm reaches the objective value of 315, which is 31% better than the one obtained by IPVNS, i.e. 460 on a similar computational environment. Running the algorithm on our regular benchmark PC, the objective value is slightly improved and reaches the value of 270 known as the optimal solution, which might be due to using a more powerful PC.

Considering instance *ORTEC01*, we also benchmark the algorithm against the winner of Personnel Scheduling track of CHeSC hyper-heuristic competition (Hsiao, Chiang & Fu, 2011; Hyde & Ochoa, 2011), where the authors developed a VNS-based hyper-heuristic (VNS-TW) consisting of two steps, i.e. shaking and local search, which is able to dynamically adjust to various problems using different techniques. Running the hybrid algorithm within the standardised time limit using the benchmark tool provided by the competition organisers, it obtains the objective value of 270 in comparison with the result of 320 obtained by VNS-TW.

Comparing our hybrid algorithm and Gurobi IP solver for a longer runtime of 60 minutes, the hybrid algorithm obtains better results, though it is not designed to be run for such a relatively long computational time. It is worth noting that the results generated by the hybrid algorithm for 10 minutes even outperform the solutions produced by the Gurobi IP solver for 60 minutes except for instance *ORTEC09*, where there is only a slight difference.

5. Conclusion

We have presented a hybrid algorithm employing a Variable Neighbourhood Search algorithm and Integer Programming to make the search process more efficient. At the first step, after generating an initial solution using a greedy heuristic, the solution is improved using a Variable Neighbourhood Descent algorithm. To increase the exploitation and exploration in the VNS, Integer Programming within a ruin-and-recreate framework is employed, where parts of the solution are kept fixed by applying a new scoring scheme. In order to ensure the investigation of the search space globally, IP again is applied to improve the obtained solution in the remaining time.

We evaluated the proposed algorithm using 24 instances introduced in the recent literature, and 12 randomly generated instances presented in this paper. The benchmark results showed better performance for most of the instances in comparison with two state-of-the-art algorithms in the literature and a standard Gurobi IP solver. The algorithmic concepts of the proposed algorithm are general enough to be applied to other similar Timetabling and Resource Allocation problems. Moreover, incorporating an IP approach into a meta-heuristic algorithm confirms the applicability of exact methods for practical instances in a hybrid setting. We also proposed a general scoring scheme to break down the total penalty associated with a solution into the fundamental elements of the problem, which is able to guide the search process adaptively towards high-potential parts of the solution.

Future research involves investigating the structure of the studied problem to accommodate other heuristic algorithms such as population-based meta-heuristics and Constraint Programming techniques to the current developed hybrid framework. We also aim to apply other Integer Programming techniques such as column generation in order to enhance the efficiency of the IP component. Another interesting research direction is to investigate more sophisticated neighbourhood structures in order to improve the efficiency of the VNS algorithm. Moreover, it would be interesting to employ a parameter tuning tool (e.g. Hutter, Hoos, Leyton-Brown, & Stützle, 2009) to precisely select best configurations for the proposed ruin-and-recreate framework.

References

Aickelin, U., & Dowsland, K. A. (2004). An indirect genetic algorithm for a nurse-scheduling problem. *Computers and Operations Research, 31*(5), 761–778. doi:10.1016/S0305-0548(03)00034-0.

Beaumont, N. (1997). Scheduling staff using mixed integer programming. *European Journal of Operational Research, 98*(3), 473–484. doi:10.1016/S0377-2217(97)00055-6.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308. doi:10.1145/937503.937505.

Bourdais, S., Galinier, P., & Pesant, G. (2003). HIBISCUS: A constraint programming application to staff scheduling in health care. In F. Rossi (Ed.), *Principles and practice of constraint programming (CP): Vol. 2833* (pp. 153–167). Springer Berlin Heidelberg. doi:10.1007/978-3-540-45193-8_11.

Burke, E., Causmaecker, P. D., & Berghe, G. V. (1999). A hybrid tabu search algorithm for the nurse rostering problem. In B. McKay, X. Yao, C. Newton, J.-H. Kim, & T. Furuhashi (Eds.), *Simulated evolution and learning: Vol. 1585*, chapter 25 (pp. 187–194). Springer Berlin Heidelberg. doi:10.1007/3-540-48873-1_25.

Burke, E. K., & Curtois, T. (2014). New approaches to nurse rostering benchmark instances. *European Journal of Operational Research, 237*(1), 71–81. doi:10.1016/j.ejor.2014.01.039.

Burke, E. K., Curtois, T., Qu, R., & Berghe, G. V. (2008). Problem model for nurse rostering benchmark instances. *Technical report*. Jubilee Campus, Nottingham, UK: ASAP, School of Computer Science, University of Nottingham. http://www.cs.nott.ac.uk/~tec/NRP/papers/ANROM.pdf.

Burke, E. K., De Causmaecker, P., Berghe, G. V., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of Scheduling, 7*(6), 441–449. doi:10.1023/B:JOSH.0000046076.75950.0b.

Burke, E. K., Li, J., & Qu, R. (2010). 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. doi:10.1016/j.ejor.2009.07.036.

Chuin Lau, H. (1996). On the complexity of manpower shift scheduling. *Computers & Operations Research, 23*(1), 93–102. doi:10.1016/0305-0548(94)00094-O.

Cipriano, R., Gaspero, L. D., & Dovier, A. (2006). Hybrid approaches for rostering: A case study in the integration of constraint programming and local search. In F. Almeida, M. Blesa Aguilera, C. Blum, J. Moreno Vega, M. Pérez Pérez, A. Roli, & M. Sampels (Eds.), *Hybrid metaheuristics: Vol. 4030*, chapter 9 (pp. 110–123). Springer Berlin Heidelberg. doi:10.1007/11890584_9.

Curtois, T., & Qu, R. (2014). Computational results on new staff scheduling benchmark instances. *Technical report 06-Oct-2014*. ASAP Research Group, School of Computer Science, University of Nottingham.

De Beukelaer, H., Davenport, G. F., De Meyer, G., & Fack, V. (2015). JAMES: A modern object-oriented Java framework for discrete optimization using local search metaheuristics. In *4th International symposium and 26th National conference on Operational Research. Hellenic Operational Research Society* (pp. 134–138).

Della Croce, F., & Salassa, F. (2014). A variable neighborhood search based matheuristic for nurse rostering problems. *Annals of Operations Research, 218*(1), 185–199. doi:10.1007/s10479-012-1235-x.

Dowsland, K. A., & Thompson, J. M. (2000). Solving a nurse scheduling problem with knapsacks, networks and tabu search. *Journal of the Operational Research Society, 51*(7), 825–833. doi:10.1057/palgrave.jors.2600970.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research, 127*(1–4), 21–144. doi:10.1023/B:ANOR.0000019087.46656.e2.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004b). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research, 153*(1), 3–27. doi:10.1016/S0377-2217(03)00095-X.

Glover, F., & Kochenberger, G. A. (2003). *Handbook of metaheuristics*. Kluwer Academic Publishers. doi:10.1007/b101874.

Gurobi Optimization, I. (2015). *Gurobi*. http://www.gurobi.com.

Hansen, P., & Mladenovic, N. (1999). *An introduction to variable neighborhood search*. Springer.

Hansen, P., & Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449–467. http://dx.doi.org/10.1016/S0377-2217(00)00100-4.

Haspeslagh, S., De Causmaecker, P., Schaerf, A., & Stølevik, M. (2014). The first international nurse rostering competition 2010. *Annals of Operational Research, 218*(1), 221–236. doi:10.1007/s10479-012-1062-0.

Hsiao, P.-C., Chiang, T.-C., & Fu, L.-C. (2011). *CHeSC 2011: A variable neighborhood search-based hyperheuristic for cross-domain optimization problems*. Tech. rep., CHeSC 2011. URL http://www.asap.cs.nott.ac.uk/external/chesc2011/

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research, 36*, 267–306. doi:10.1613/jair.2808.

Hyde, M., & Ochoa, G. (2011). *CHeSC 2011 – The First Cross-domain Heuristic Search Challenge*. http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html.

IBM (2015). IBM ILOG CPLEX CP Optimizer. http://www.ibm.com/software/integration/optimization/cplex-cp-optimizer/.

Li, J., Aickelin, U., & Burke, E. K. (2009). A component-based heuristic search method with evolutionary eliminations for hospital personnel scheduling. *INFORMS Journal on Computing, 21*(3), 468–479. doi:10.1287/ijoc.1080.0298.

Li, J., Bai, R., Shen, Y., & Qu, R. (2015). Search with evolutionary ruin and stochastic rebuild: A theoretic framework and a case study on exam timetabling. *European Journal of Operational Research, 242*(3), 798–806. doi:10.1016/j.ejor.2014.11.002.

Lu, Z., & Hao, J. K. (2012). Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research, 218*(3), 865–876. doi:10.1016/j.ejor.2011.12.016.

Osogami, T., & Imai, H. (2000). Classification of various neighborhood operations for the nurse scheduling problem. In Lecture notes in computer science*: Vol. 1969* (pp. 72–83). Springer. doi:10.1007/3-540-40996-3_7.

Qu, R., & He, F. (2009). A hybrid constraint programming approach for nurse rostering problems. In T. Allen, R. Ellis, & M. Petridis (Eds.), *Proceedings of the AI 2008, the twenty-eighth SGAI international conference on innovative techniques and applications of artificial intelligence on Applications and innovations in intelligent systems XVI,* chapter 16 (pp. 211–224). Springer London. doi:10.1007/978-1-84882-215-3-16.

Rahimian, E. (2015). *Nurse Rostering Dataset*. URL http://dx.doi.org/10.15129/9664f00a-e2fd-4dbb-afef-f3c076e2c4f7.

Rahimian, E., Akartunali, K., & Levine, J. (2015). A hybrid constraint integer programming approach to sSolve nurse scheduling problems. In *Proceedings of the multidisciplinary international conference on scheduling: Theory and applications, MISTA 2015* (pp. 429–442).

Raidl, G., & Puchinger, J. (2008). Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. Aguilera, A. Roli, & M. Sampels (Eds.), *Studies in computational intelligence (SCI): Vol. 62,* chapter 2 (pp. 31–62). Springer Berlin Heidelberg. doi:10.1007/978-3-540-78295-7_2.

Raidl, G. R., Puchinger, J., & Blum, C. (2010). Metaheuristic hybrids. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics: Vol. 146,* chapter 16 (pp. 469–496). Springer US. doi:10.1007/978-1-4419-1665-5_16.

Stølevik, M., Nordlander, T. E., Riise, A., & Frøyseth, H. (2011). A hybrid approach for solving real-world nurse rostering problems. In *International Conference on Principles and Practice of Constraint Programming* (pp. 85–99). Springer Berlin Heidelberg.

Talbi, E. G. (2009). *Metaheuristics: From design to implementation*. Wiley.

Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., & Housos, E. (2012). A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research, 219*(2), 425–433. doi:10.1016/j.ejor.2011.12.042.