MISTA 2015

A Hybrid Constraint Integer Programming Approach to Solve Nurse Scheduling Problems

Erfan Rahimian • Kerem Akartunali • John Levine

Abstract The Nurse Scheduling Problem can be simply defined as assigning a series of shift sequences (schedules) to several nurses over a planning horizon according to some constraints and preferences. The inherent benefits of having higher-quality and more flexible schedules are a reduction in outsourcing costs and an increase of job satisfaction in health organizations. In this paper, we present a novel systematic hybrid algorithm, which combines Integer Programming (IP) and Constraint Programming (CP) to efficiently solve highly-constrained Nurse Scheduling Problems. Our focus is to exploit the problem-specific information to improve the performance of the algorithm, and therefore obtain high-quality solutions as well as strong lower bounds. We test our algorithm based on some real-world benchmark instances. Very competitive results are reported compared to the state-of-the-art algorithms from the recent literature, showing that the proposed algorithm is able to solve a wide variety of real-world instances with different complex structures.

1 Introduction

In order to ensure the right staff on the right duty at the right time, Nurse Scheduling (NS) has drawn significant attention during the last few decades, helping many health organizations to increase their efficiency and productivity. Creating a high-quality nurse schedule raises the recruitment and retention levels of nursing personnel, and maintains a reasonable overtime budget for nursing staff. In terms of financial issues, it can reduce outsourcing and planning costs due to hiring fewer bank nurses to compensate gaps in rosters, and having flexible schedules [1, 2]. In terms of human resource issues, it can increase the job satisfaction and diminish the fatigue and stress, and hence result in improving caring services provided to patients [3, 4].

Nurse Scheduling Problem (NSP) aims to generate schedules for several nurses over a planning horizon. A schedule consists of a sequence of different types of shifts (e.g. early, late,

Erfan Rahimian, Kerem Akartunali
Dept. of Management Science, University of Strathclyde, Glasgow, G1 1QE, UK
E-mail: {erfan.rahimian, kerem.akartunali}@strath.ac.uk

John Levine
Computer And Information Sciences, University of Strathclyde, Glasgow, G1 1XH, UK
E-mail: john.levine@strath.ac.uk

vacations) spanning over the whole planning period. The pattern of shifts is generated according to a set of requirements such as hospital regulations, and a number of preferences such as fair distribution of shifts between nurses. Due to their complex and highly-constrained structure, most NSPs in real-world situations are computationally challenging and they can be also classified as NP-hard [5, 6]. The inherent nature of the problem usually leads to divide all constraints to two categories in practice: hard and soft constraints. Hard constraints must be satisfied to have a feasible roster, whereas soft constraints may be violated. To evaluate the quality of a roster, one can minimize the sum of all penalties incurred due to soft constraint violations. For more information regarding NSPs and generally staff scheduling problems, we refer interested readers to [3, 28].

The focus of this paper is on integrating Integer Programming (IP) and Constraint Programming (CP) to solve NSPs, where we exploit the problem-specific information in order to improve both IP and CP performance. In the literature, there are two areas of general methods used to solve these problems: exact and heuristic methods. Exact methods involve IP [1, 7, 8] and CP [9, 10], which are capable of finding the optimal solution, albeit often resulting in unacceptable computational times. However, recent research in Operations Research and Artificial Intelligence communities, combined with powerful solvers such as IBM Ilog Cplex and Gurobi, focused on using these methods in hybrid settings [14-16]. On the other hand, in order to address the computational limitations of exact methods, many heuristic methods have been proposed in the literature. However, these methods sacrifice the guarantee of an optimal solution (or even any information about the solution quality) in order to generate good solutions in acceptable computational times. We note [11-13] as some recent examples of using heuristic methods in the NSP literature.

In recent years, some researchers experimented with hybridizations of different methods, e.g. CP and heuristics [14], IP and heuristics [15], and less well-investigated combination of IP and CP [16], in order to utilize the strengths of all methods together. In this paper, we propose a new systematic hybrid algorithm using IP and CP approaches, which is capable of finding the optimal solution. Due to the exact nature of the proposed algorithm, it can generate a good solution as well as a good lower bound in contrast to heuristic methods. The hybrid algorithm exploits the problem-specific information to reduce the search space, to fine tune the search parameters, and to improve the efficiency of the search process in a novel way. In other words, using an IP approach as the main solution method, we employ a CP approach and some other algorithmic aids to improve the efficiency of the algorithm. Moreover, the proposed algorithm is designed to obtain the best result in a pre-defined limited computational time. We model the problem according to a general comprehensive model reported in the literature [17] and evaluate it using some test instances published therein.

The rest of this paper is organized as follows: problem definition and assumptions are explained in Section 2. The mathematical and CP formulations is presented in Section 3 and 4. In Section 5, we describe the proposed hybrid algorithm and its components. Computational results are reported in Section 6, and some conclusions are drawn in Section 7.

2 Problem Definition

NSP is the process of assigning a number of nurses to a number of work shifts during a planning horizon according to a set of requirements and constraints. These constraints are usually categorized to hard and soft constraints. In the following, we define decision variables and constraints according to the conceptual model described in [17], which will be used to construct an IP model.

We define our decision variables for each nurse, for each day, and for each shift type. This way of modeling allows us to better utilize the problem-specific structure in order to reduce the search space, although it is less flexible and contains more symmetry compared to the pattern-based modelling (e.g. [12]), which generates all possible weekly shift sequences (patterns) and hence considers all constraints except coverage constraints. We assume the current roster is modelled over a planning horizon in an isolated way, i.e. no information (history) from the

previous roster is used to construct the current one. We also consider a day-off as a shift type for modelling purposes. For the sake of simplicity, we assume all nurses belong to the same skill category. In addition, we assume all rosters start from Monday and are made from a complete week (includes seven days with a two-day weekend). The constraints of the model are:

1. Maximum one assignment per shift type per day,
2. Coverage constraints: the number of shift types for each day must be fulfilled,
3. The minimum and maximum number of:
   (a) shift assignments within the scheduling period,
   (b) consecutive working days over the planning horizon,
   (c) working hours within the scheduling period (and/or during a week),
   (d) shift assignments within a week,
   (e) shift assignments at the weekend,
   (f) consecutive shift types over the planning period,
4. Minimum number of days-off after a night shift or a series of night shifts,
5. Complete weekends: over the weekends, there should be either an assignment to all days of weekends or no assignments at all,
6. No night shift before free weekends, where there is no assignment at all,
7. Maximum number of consecutive worked weekends, where there is at least one assignment,
8. Requested shifts (days) on or off,
9. Forbidden shift type patterns (e.g. the "ND" pattern, where the shift type "D" is not allowed to be assigned right after the shift type "N").

In the next two sections, we formulate this model using Integer Programming (IP) and Constraint Programming (CP). We also note that the above constraints can be considered hard or soft according to different settings. For the sake of simplicity, we only provide here a formulation assuming that all constraints are hard. In case any soft constraints exist in the model, our objective function can be defined as the weighted sum of all slack variables in IP or reified variables in CP for each soft constraint.

3 Mathematical Formulation

Here we present our mathematical formulation using Integer Programming based on the definitions and assumptions from the previous section. The variables, parameters, and constraints of the model are defined as follows:

Decision Variables:

x_{ead}	Binary variable indicating whether shift type a on day d is assigned to nurse e or not.
p_{ed}	Binary variable indicating whether nurse e works on day d or not.
k_{ew}	Binary variable indicating whether nurse e is assigned to weekend w or not.
y_{ea}	Total number of times that shift type a assigned to nurse e over the planning period.
z_{ewa}	Total number of shift type a assigned to nurse e during week w.

Parameters:

N	Set of nurses.
D	Set of days.
A	Set of shift types.
W	Set of weeks.

H_a	Set of shift types that cannot be assigned immediately after shift type *a*.
PR_{ad}	Set of pre-assigned nurses to shift type *a* on day *d*.
ML_e, MU_e	Minimum and maximum number of shifts that can be assigned to nurse *e* within the planning period.
WL_w, WU_w	Minimum and maximum number of shifts that can be assigned to a nurse within week *w*.
VL_d, VU_d	Minimum and maximum number of shifts that can be assigned to nurses on day *d*.
AL, AU	Minimum and maximum number of hours that can be assigned to each nurse during the planning period.
EL_w, EU_w	Minimum and maximum number of hours that can be assigned to each nurse during week *w*.
NL, NU	Minimum and maximum number of consecutive working days over the planning period.
HL_a, HU_a	Minimum and maximum number of consecutive shift type *a* over the planning period.
KL, KU	Minimum and maximum number of worked weekends over the planning horizon.
CU	Maximum number of consecutive worked weekends over the planning period.
UT_a	Total workloads (hours) of shift type *a* within the planning period.
UT_{aw}	Total workloads (hours) of shift type *a* during week *w*.

Constraints:

Next, we present our IP formulation, where the order of the constraints is preserved the same as the order of the constraints presented in Section 2:

$$\sum_{a \in A} x_{ead} = 1, \forall e \in N, d \in D \quad (1)$$

$$p_{ed} = \sum_{a \in A - \{r\}} x_{ead}, \forall e \in N, d \in D \quad (2)$$

$$VL_d \leq \sum_{e \in N} p_{ed} \leq VU_d, \forall d \in D$$

$$y_{ea} = \sum_{d \in D} x_{ead}, \forall e \in N, a \in A \quad (3.a)$$

$$ML_e \leq \sum_{a \in A} y_{ea} \leq MU_e, \forall e \in N$$

$$\sum_{g=d}^{NU+d} p_{eg} \leq NU, \forall e \in N, d \in \{1 \dots |D| - NU\} \quad (3.b)$$

$$\sum_{i=1}^{NL-1} p_{ed+i} \leq p_{ed} + p_{ed+NL} + NL - 2, \forall e \in N, d \in D$$

$$AL \leq \sum_{a \in A} y_{ea} UT_a \leq AU, \forall e \in N \quad (3.c)$$

$$z_{ewa} = \sum_{d=7(w-1)+1}^{7w} x_{ead}, \forall e \in N, a \in A, w \in W$$

$$EL_w \le \sum_{a \in A} z_{ewa} UT_{aw} \le EU_w, \forall e \in N, w \in W$$

$$WL_w \le \sum_{a \in A} z_{ewa} \le WU_w, \forall e \in N, w \in W \quad (3.d)$$

$$k_{ew} \le p_{ed} + p_{ed+1} \le 2k_{ew}, \forall e \in N, w \in W, d = 7w - 1 \quad (3.e)$$

$$KL \le \sum_{w \in W} k_{ew} \le KU, \forall e \in N$$

$$\sum_{g=d}^{HU_a+d} x_{eag} \le HU_a, \forall e \in N, a \in A, d \in \{1 \dots |D| - HU_a\} \quad (3.f)$$

$$\sum_{i=1}^{HL_a-1} x_{ead+i} \le x_{ead} + x_{ead+HL_a} + HL_a - 2, \forall e \in N, a \in A, d \in D$$

$$x_{end} \le x_{end+1} + 1 - p_{ed+1}, \forall e \in N, d \in \{1 \dots |D| - 1\} \quad (4)$$

$$x_{end} - p_{ed+1} \le 1 - p_{ed+2}, \forall e \in N, d \in \{1 \dots |D| - 2\}$$

$$x_{erd} = x_{erd+1}, \forall e \in N, d \in \{6,13, \dots, |D| - 1\} \quad (5)$$

$$x_{end} \le p_{ed+1} + p_{ed+2}, \forall e \in N, d \in \{5,12, \dots, |D| - 2\} \quad (6)$$

$$\sum_{i=0}^{CU} k_{ew+i} \le CU, \forall e \in N, w \in \{1 \dots |W| - CU\} \quad (7)$$

$$x_{ead} = 1, \forall e \in PR_{ad}, a \in A, d \in D \quad (8)$$

$$x_{ead} + x_{ehd+1} \le 1, \forall e \in N, a \in A, h \in H_a, d \in \{1 \dots |D| - 1\} \quad (9)$$

In constraint (4), we assume that there should be two days-off after a night shift or a series of night shift types. Furthermore, in constraints (2), (4), (5), and (6), "*n*" and "*r*" indicate a night shift type and a day-off, respectively.

4 Constraint Programming Formulation

Here we present our CP formulation based on Constraint Satisfaction Problem (CSP) model according to the definitions and assumptions provided in Section 2. The presented model is detailed enough for the needs of this paper, however, we would add other redundant constraints or variables to increase the efficiency of the CP solver. In this section, first, we concisely explain the two types of global constraints which we use in the CP model: *Cardinality* and *Stretch*. For more information about global constraints in CP, we refer to [24-25]. Next, we define the variables, parameters, and constraints of the model. We use the same parameters as defined in IP formulation (Section 3), therefore we define only new ones here.

Cardinality constraints (aka. *GCC* or *generalized cardinality*) bounds the number of times that variables take a certain set of domain values. It is written as $cardinality(x, v, l, u)$ where x is a set of variables $(x_1, \dots, x_n)$; v is a m-tuple of domain values of the variables x; l and u are m-tuples of non-negative integers defining the lower and upper bounds of the times value v being taken by variable x, respectively. The constraint defines that, for $j = 1, \dots, m$, at least l_j and at most u_j of the variables x take value v_j.

Stretch constraints bounds the sequence of consecutive variables that take the same value (stretch), i.e. $x_{j-1} \neq 1, x_j, \dots, x_k = v, x_{k+1} \neq v$. It is written as $stretch(x, v, l, u, P)$ where x is a set of variables $(x_1, \dots, x_n)$; v is a m-tuple of possible domain values of x; l and u are m-tuples of lower and upper bounds for x, respectively. P is a set of patterns, i.e. pairs of values (v_j, v_k),

requiring that when a stretch of value v_j immediately precedes a stretch of value v_k, the pair (v_j, v_k) must be in P.

Decision Variable:

s_{ed}	Integer variable indicating the shift type assigned to nurse *e* on day *d*.

Parameters:

$\widetilde{H}_a$	Set of shift types that can be assigned immediately after shift type a.
UT	The vector of total workloads (hours) of the shift types within the planning period.
UT_w	The vector of total workloads (hours) of the shift types during week *w*.

Constraints:

Next, we present our CP formulation based on the defined global constraints, where the order of the constraints is preserved the same as the order of the constraints presented in Section 2:

$$cardinality\left(\bigcup_{e\in N} s_{ed}, A, VL_d, VU_d\right), \forall d \in D \quad (2)$$

$$cardinality\left(\bigcup_{d\in D} s_{ed}, A, ML_e, MU_e\right), \forall e \in N \quad (3.a)$$

$$stretch(s_{ed}, A, NL, NU, P), \forall e \in N, d \in D, P = \{(a, r) | a \in A\} \quad (3.b)$$

$$AL \leq prod(s_{ed}, UT) \leq AU, \forall e \in N, d \in D \quad (3.c)$$

$$EL_w \leq prod(s_{ed}, UT_w) \leq EU_w, \forall e \in N, w \in W, d = 7(w-1)+1$$

$$cardinality\left(\bigcup_{d=7(w-1)+1}^{7w} s_{ed}, A, WL_w, WU_w\right), \forall e \in N, w \in W \quad (3.d)$$

$$cardinality(s_{ed}, r, KL, KU), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1\}\} \quad (3.e)$$

$$stretch(s_{ed}, a, HL_a, HU_a, P), \forall e \in N, d \in D, a \in A, P = \{\} \quad (3.f)$$

$$stretch(s_{ed}, n, 2,3, P), \forall e \in N, d \in D, P = \{(n, r)\} \quad (4)$$

$$s_{ed} = s_{ed+1}, \forall e \in N, d \in \{6,13, \dots, |D| - 1\} \quad (5)$$

$$stretch(s_{ed}, n, 2,3, P), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1,2\}\}, P = \{(n, r)\} \quad (6)$$

$$stretch(s_{ed}, r, 2(|W| - CU), 2|W|, P), \forall e \in N, d \in \{7w - i | w \in W, i \in \{0,1\}\}, P = \{(r, r)\} \quad (7)$$

$$s_{ed} = a, \forall e \in PR_{ad}, a \in A, d \in D \quad (8)$$

$$stretch(s_{ed}, a, 0,2, P), \forall e \in N, d \in D, P = \{(a, \widetilde{H}_a) | a \in A\} \quad (9)$$

In constraint (4), we assume that there should be two days-off after a night shift or a series of night shift types. Furthermore, in the mentioned constraints, "*n*" and "*r*" indicate a night shift type and a day-off, respectively. It should be noted that constraint (1) is already satisfied due to the inherent structure of the CP model.

5 Integration of CP and IP

For small- to medium-sized problems, IP solvers are often efficient to find the optimal solution and to generate strong lower bounds. Similarly, CP solvers are capable of finding feasible solutions. However, using these approaches on their own for solving large-scale problems, or even small-scale problems with a highly-constrained structure often leads to a very poor performance. For example, solving NSPs using the model presented in Section 3 with a pure IP approach, we were not able to obtain an optimal solution (and in some cases even a good-quality solution) in a reasonable amount of time, where some instances took more than 24 hours to solve. Similarly, a pure CP approach results in poor performance as well, since it often takes a long time to achieve a feasible or optimal solution. Therefore, it is intuitive to hybridize them in order to utilize their strengths for efficiently solving NSPs. In this paper, we integrate IP and CP approaches in a pipeline fashion to solve the problem. To improve the efficiency of the hybrid algorithm, we exploit the problem structure to provide valuable information about search space, hence improve the performance of the proposed algorithm. Indeed, we use a CP approach and some other algorithmic procedures to help the IP approach as our main solution method.

The algorithm presented in this paper is tested on nine different instances published in [17]. The diversity in the structure and complexity of these instances allows us to test our algorithm thoroughly. Table 1 provides more information about these instances, where the reported number of variables and constraints are based on the described model presented in Section 3. It is noteworthy to mention that although we tried to solve a few instances based on real-world cases, we developed our solution method without any fine tuning. Therefore, we believe that our approach can be easily generalized to solve different instances based on the presented IP model.

Table 1. Benchmark instances

Instance	Nurses	Shift types	Days	Shift permutations	Variables	Constraints
GPOST	8	3	28	3136	5680	5504
GPOSTB	8	3	28	3136	5680	5496
ORTEC01	16	5	33	7821	19096	19170
ORTEC02	16	5	33	7821	19101	19175
Valouxis-1	16	4	28	5824	9776	9968
SINTEF	24	6	21	6867	8118	6927
WHPP	30	4	14	5880	6000	5842
MILLAR-1	8	3	14	784	1956	1820
LLR	27	4	7	1323	1139	979

In the following, we provide a brief description of the performance of the hybrid algorithm, and later we will elaborate each associated component. After a quick pre-processing in order to create appropriate data structures for the algorithm, at first step, we employ an *IP pre-solver* in order to identify any valuable information. If any valuable information is identified, we continue to use the *IP solver* (rather than a *CP solver*) for the next steps, since it has more potential to be successful in solving the problem, as we experienced in our experiments. In the next step, we employ a *CP solver* to solve the problem considering only those constraints which will not make the problem difficult to solve. Identifying difficult constraints is achieved with solving a hierarchy of different CSPs iteratively. Next, using the information provided by the CP solver operated on a modified problem and generated CSPs, we solve the problem by an *IP solver* (or the *CP solver* based on the obtained information from the IP pre-solver) during the remaining time. We also add three other components to reinforce the search process using the exploited problem-specific information: i) *Symmetry breaker*, which tries to remove (or mitigate) the symmetric structures; ii) *Weight balancer*, which tries to modify each constraint's weight based on a pre-defined threshold in order to tighten the problem formulation; and iii) *Decomposer*, which provides a lower bound for the IP solver.

It should be noted that the proposed hybrid algorithm runs in a pre-defined time to solve the problem. In fact, the user determines the running time of each component by setting the relevant computational time parameter.

The schematic diagram of the proposed algorithm is depicted in Figure 1:

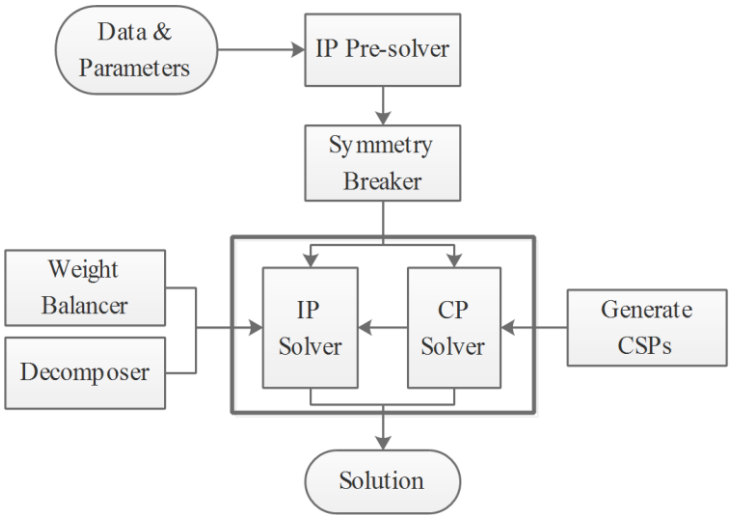

Fig. 1. Schematic diagram of the proposed hybrid algorithm

Next, we explain each component individually in more details:

IP Pre-solver: In fact, this component is the first step in most of the commercial solvers to analyze and simplify the problem structure, and also identify any specific structures such as network flow or assignment problems. If the IP solver can identify any particular structures, it often leads to a better performance during the search process. Here, we only call the pre-solve step of an IP solver from the hybrid algorithm as a black-box. We use the information obtained from this step to predict if there are any specific structures, and therefore improving the performance of the IP solver. Particularly, we use the obtained lower bound and relaxed objective function value to understand the existence of any specific structures in this black-box indirectly. According to our experiments, if the IP pre-solver component provides a stronger (greater) lower bound compared to the relaxed objective function value (the initial identified lower bound for an IP problem), the employed IP solver is a better choice to solve the problem, otherwise we will use the *CP solver* instead. We also switch on the relevant parameter for the pre-solve step of the IP solver to the highest degree (aggressive mode) in this component (e.g. setting the *Presolve* parameter in Gurobi). Moreover, using the reported number of constraints and variables in this step, if they are more than a user-defined threshold (*psThr*), we will change the search strategy of the *IP Solver* accordingly. We will explain this setting in the IP Solver component in more details.

CP Solver: During the search process, the hybrid algorithm may call the CP solver in two cases: first, as the main solver if the IP pre-solver does not provide valuable information about the problem due to its complex and highly-constrained structure; second, as an aid for the IP solver to provide a good-quality initial solution. This solver solves the problem based on the Constraint Satisfaction Problem (CSP) model presented in Section 4. In our experiments on the benchmark instances, CP approach did not provide very good-quality solutions in a limited time. To address this issue, we implement the following procedure: First, we generate a CSP model considering all constraints that have a weight higher than a user-defined threshold (*cspThr*). If the problem

is infeasible, we will increase the threshold by one unit. Otherwise, we will generate a number of solutions based on the modified model according to a user-defined parameter (*numSols*). Therefore, on each threshold level, there might be several feasible solutions. This process continues until the number of constraints in the new generated model is equal to the number of constraints in the original model. Finally, we report the best-quality solution in terms of objective function value. Next if the IP solver is a candidate for solving the problem, the reported solution will be imported to the IP solver. Otherwise, we continue solving the problem using the CP solver in the remaining time. We will explain this setting in the IP solver component in more details. The pseudo code of this procedure is presented below, where *p*, *p'*, *cspThr*, and *numSols* indicate the original problem, the new generated problem in each threshold level, the user-defined threshold level, and the user-defined number of solutions needs to be generated in each threshold level, respectively.

```
Solutions = empty
p' = p
While (true)
   p' = generateCSP(p, cspThr)
   If p' is feasible then
      For i = 1 to numSols
            Solutions.add(solve(p'))
      Next
   Else
      cspThr++
   End If
   If numConstraint(p') >= numConstraint(p) then break
End While
Return bestObj(Solutions)
```

Using the information provided in this procedure by solving a variety of CSP problems, we can also find out an estimate for the difficulty of each constraint. If the solution time for adding a constraint to a problem in order to generate a new modified problem is significant, we will count it as a "difficult constraint". We only record the solution time for the last occurrence when a specific constraint is added to a problem during the process of generating CSPs. To our experiments, 15 seconds is sufficient for most of the benchmark instances. This simple inference helps us later in the *Weight Balancer* component to make the formulation of the problem tighter.

IP Solver: In this component, we use a state-of-the-art IP solver to solve the problem during the remaining time. The only difference between this component and running a pure IP solver is the initial solution and parameter settings provided to the solver from other relevant components. We use the solution obtained from the CP solver as a warm start for the IP solver. Moreover, we change some parameters of the IP solver based on the information provided by the IP Pre-solver. Indeed, if the IP Pre-solver provides a good lower bound (elaborated in the IP Pre-solver component), we switch off the pre-solve step in this component (e.g. setting the *Presolve* parameter in Gurobi). We also change the search strategy based on the number of constraints and variables provided by the IP Pre-solver, and a user-specified threshold, i.e. *psThr*. If the number of constraints and variables of the problem are more than *psThr*, we set the search strategy to spend more efforts on obtaining a feasible solution rather than proving optimality. We do not change the default search strategy in case a problem is not difficult to solve. In most of the modern solvers, the user can change the search strategy by a specific parameter defined therein. For example, in Gurobi solver, the user can tailor the search strategy by setting the *MIPFocus* parameter. Furthermore, using the lower bound provided by the *Decomposer* component, we enforce it on the IP solver by setting the relevant parameter accordingly (e.g. setting the *Start* parameter in Gurobi).

Symmetry Breaker: As we mentioned in Section 2, modeling the problem using indexed variables can create symmetry issues. To resolve these issues, we add lexicographic ordering constraints [25] to both CSP and IP models applied to the main variables (i.e. x_{ead} and s_{ed}, respectively). We then use the new model for both the IP and CP Solver components. In Section 6, we will mention that breaking a symmetric structure in the model is often beneficial for the solver.

Weight Balancer: In order to improve the efficiency of the IP solver during the search process, we modify the weights in the objective function due to the difficulty degree of constraints, which we elaborated in the CP Solver component. Based on this degree, if a constraint is not difficult, we impose it to the IP solver as a hard constraint. Theoretically, this process may lead to an infeasible problem. In this case, we undo the relevant change and continue the process for the rest of the constraints. Finally, we solve the new modified problem using the IP solver. This technique helps to reduce the search space, which results in a better efficiency during the search process.

Decomposer: One of the design aspect of the proposed hybrid algorithm is to generate a good lower bound for most of the benchmark instances. In this component, we decompose the problem to weekly rosters, and then we evaluate all possible shift patterns according to "forbidden shift pattern" and "request on or off" constraints (constraints 8 and 9 in Section 2). In this process, we try to find out whether there is an inevitable conflict in the model, which can be discovered before solving the problem. When there is an inherent conflict in the model according to the current data, we can calculate the associated penalty based on the objective function and consider it as a new lower bound. We do this particular evaluations for all decomposed weekly rosters in a problem. This process is elaborated in [7], where the authors try to infer a lower bound for two specific instances. However, here we use the same technique but for all decomposed weekly rosters, and not only for particular instances. Apart from this process, we also solve all decomposed weekly rosters by an IP solver to discover any further potential lower bounds. Finally, the best lower bound calculated in this component will be imported to the IP solver by setting the relevant parameter (e.g. setting the *Start* parameter in Gurobi).

6 Computational Results

To evaluate the proposed hybrid algorithm, we implemented our algorithm in Java 1.7, and used the IBM ILOG CP solver 1.7 for solving all CSPs and Gurobi 5.6 to solve all IPs. The reason to use the aforementioned solvers is that we found them easier to implement in terms of modeling, and also they suit our hybrid framework better than other software packages. In addition, we note that the benchmarks reported in [27] show that Gurobi and IBM Ilog Cplex produce very similar results for most of the instances. We run our experiments on a PC with Intel 3.4 GHz processor and 4 GB of RAM, and we used the benchmark instances introduced in Section 5. The variety in benchmark instances helps us to test and analyze our algorithm in different circumstances. To the best of our knowledge, we are the first researchers experimenting with all these instances.

For evaluation purposes, we run the hybrid algorithm for 10 minutes, and distribute 10%, 30%, and 50% of the time to IP Pre-solver, CP Solver, and IP Solver components, respectively. The rest of the time is distributed equally to other components as they require very short times in comparison. The reasons for benchmarking the proposed algorithm in 10 minutes are two-fold: i) we primarily designed the hybrid algorithm to run in a short time; ii) the selected time is in line with the testing times used by most of the algorithms reported in the literature, including the time used in the first International Nurse Rostering Competition (INRC-I) [26], and hence provides a platform for a fair comparison. Furthermore, we set the threshold parameters for the IP Pre-solver and CP Solver components, i.e. *psThr* and *cspThr*, to 10000 and 10 respectively. We also set the *numSols* parameter for the CP Solver component to 500. The design of the

algorithm is primarily deterministic, however to address the minor random behavior due to the intrinsic nature of the employed solvers, we run it three times per instance for each experiment and report average values.

We conducted two experiments to test the proposed algorithm: first, we investigate the benefit and efficiency of the Symmetry Breaker and Weight Balancer components, and how they affect the performance of the algorithm. Then, we compare the hybrid algorithm against five most recent best algorithms in the literature.

The first experiment is designed to investigate the effects of breaking symmetry and modifying weights on overall performance of the hybrid algorithm. For each test, the best objective function value, lower bound, and duality gap were recorded. The results are shown in Table 2. It should be noted that the algorithm solved instances SINTEF, MILLAR-1, and LLR in less than 3 seconds, therefore, we only report the results for the rest of the instances (six instances) in this experiment.

The results of running the hybrid algorithm using all the components are indicated as "default setting" in the first part of Table 2. For the next two parts, we remove the Symmetry Breaker and Weight Balancer components, respectively. As it can be seen, having symmetry structures in the problem worsens the duality gap for three of instances, i.e. GPOST, ORTEC01, ORTEC02, whereas it does not change the duality gaps for instances GPOSTB, Valouxis-1, and WHPP. The reason to obtain the same results is because of the limited complexity in the structure of these instances. As a result, the hybrid algorithm solved them easily compared with the other instances, although they have symmetry issues. Therefore, Symmetry Breaker component seems to improve the efficiency of the hybrid algorithm, in particular for problems with a very complex structure.

In the third part, we removed only the Weight Balancer component. The results show similar duality gaps to the second part for all the instances except a reduction for instance ORTEC02, and an increase for instance ORTEC01, which are not significant. Indeed, these instances have a particular structure that only modifying weights could not improve the performance of the algorithm. However, one can see the effect of this component when it is accompanied by the Symmetry Breaker component (Table 3). Consequently, we decided to include this component in the default setting for two reasons: first, our aim is to develop a hybrid algorithm, which is able to solve a variety of instances (in particular hard ones) successfully. Since the third instance is one of the difficult instances in our benchmark, and adding the Weight Balancer component results in a better solution, it is reasonable to keep this component in the hybrid algorithm. Second, according to our other experiments on some modified version of the current instances, and also some new generated instances, we found that generally including the Weight Balancer component leads to better-quality solutions.

Table 2. The hybrid algorithm results in different settings

Instance	Default setting			No Symmetry Breaker			No Weight Balancer		
	UB	LB	G(%)	UB	LB	G(%)	UB	LB	G(%)
GPOST	5	5	0	8	5	37.5	5	5	0
GPOSTB	5	0	100	3	0	100	5	0	100
ORTEC01	380	150	60.52	530	140	73.58	680	140	79.41
ORTEC02	370	150	59.45	570	140	75.44	340	140	58.82
Valouxis-1	20	0	100	20	0	100	20	0	100
WHPP	5	0	100	5	0	100	5	0	100

To compare the performance of the current algorithm against the stat-of-the-art algorithms reported in the literature, Table 3 shows the best-published results from: a hybrid Variable Neighborhood Search [18], a Memetic Algorithm [19], a Variable Depth Search [20], a Harmony Search Algorithm [21], a Scatter Search [22], and another hybrid Variable Neighborhood Search [23]. Unfortunately, to the best of our knowledge, we are not aware of any exact approaches and

hence we did not include any in our benchmarking. Moreover, as we mentioned in Section 5, we do not report the results of pure IP and CP solvers due to their poor performance on most of the benchmark instances. In Table 3, the column "Opt." shows the known optimal solution for the benchmark instances according to [17], where often obtained using column generation and relaxation techniques with an IP solver for a long runtime. We report the best results and their computational times (in seconds) in columns "Best" and "T", respectively. Although we run all experiments only for 10 minutes, we report the computational times for any instances the hybrid algorithm could find an optimal solution sooner. Columns "LB" and "G(%)" indicate the obtained lower bounds and duality gaps for the benchmark instances, respectively.

As it can be seen, our proposed hybrid algorithm is able to outperform other algorithms for six instances, and obtained promising results for instances ORTEC01 and ORTEC02. For instance WHPP, we could not find out any reported result in the literature other than the optimal solution mentioned in [17]. Furthermore, for instances GPOST, SINTEF, MILLAR-1, and LLR, the hybrid algorithm obtained the optimal solution in a very short time compared to other algorithms. Comparing the results of our algorithm with the Scatter Search, for instances GPOSTB, MILAAR-1, and LLR, we obtained the same results, but in a shorter time. For instances GPOST, Valouxis-1, and SINTEF, the hybrid algorithm found the best solutions, which are significantly better than the others.

It is worth noting that the proposed algorithm found the solutions reported in Table 3, while our aim of designing the hybrid algorithm was not only to find a good feasible solution, but also to achieve a better duality gap for ensuring solution quality.

Table 3. Benchmark results for our algorithm versus other algorithms reported in the literature

Instance	Opt.	The hybrid algorithm				[18]		[19]		[20]		[21]		[22]		[23]	
		Best	LB	G(%)	T(s)	Best	T(h)	Best	T(s)	Best	T(s)	Best	T(s)	Best	T(s)	Best	T(s)
GPOST	5	5	5	0	323			915	121					9	861	8	234
GPOSTB	3	5	0	100				789	95					5	791		
ORTEC01	270	380	150	60.52		541	12	535	1516	360	300	310	412	365	680		
ORTEC02	270	370	150	59.45								330	446				
Valouxis-1	20	20	0	100				560	593					100	800	160	3780
SINTEF	0	0	0	0	6			8	175					4	821		
MILLAR-1	0	0	0	0	1			100	8					0	182	0	1
WHPP	5	5	0	100													
LLR	301	301	301	0	8			305	38					301	423	314	79

7 Summary and Conclusion

This paper proposed a new systematic hybrid algorithm combining IP and CP to solve real-world Nurse Scheduling Problems. The algorithm utilized the strengths of CP to aid the IP solver to achieve better solutions. Concentrated on the problem structure, we developed some components to provide valuable problem-specific information for both IP and CP solvers so that better performance can be achieved in solving highly-constrained instances. In contrast to heuristic methods reported in the literature, we attempted to design a hybrid method to generate a good optimality gap. Moreover, we provided both CP and IP models of the problem.

We tested our algorithm on a diverse test bed of nine real-world instances from the literature. We conducted two experiments to evaluate the effectiveness of different components of the proposed algorithm, and its performance compared to some state-of-the-art algorithms. The results show that proposed algorithm is capable of obtaining competitive results.

Our future work will investigate different models for the Nurse Scheduling Problems compared with the classical model consists of indexed variables. We will also try to add a heuristic component to the proposed hybrid algorithm to improve its performance. Exploiting the problem-specific information, we will attempt to design a more sophisticated framework

doing lots of heuristics, automation, etc. to accommodate different characteristic of the problem. Finally, we are going to extend our algorithm to other scheduling problems.

References

1. M'Hallah, R. and A. Alkhabbaz, Scheduling of nurses: A case study of a Kuwaiti health care unit. Operations Research for Health Care, 2013. 2(1–2): p. 1-19.
2. Kazahaya, G., Harnessing technology to redesign labor cost management reports. Healthcare financial management: journal of the Healthcare Financial Management Association, 2005. 59(4): p. 94-100.
3. Burke, E., et al., The State of the Art of Nurse Rostering. Journal of Scheduling, 2004. 7(6): p. 441-499.
4. Ozcan, Y.A., Quantitative methods in health care management: techniques and applications. Vol. 4. 2005: John Wiley & Sons.
5. Brucker, P., R. Qu, and E. Burke, Personnel scheduling: Models and complexity. European Journal of Operational Research, 2011. 210(3): p. 467-473.
6. Karp, R.M., Reducibility among combinatorial problems. 1972: Springer.
7. Glass, C.A. and R.A. Knight, The nurse rostering problem: A critical appraisal of the problem structure. European Journal of Operational Research, 2010. 202(2): p. 379-389.
8. Maenhout, B. and M. Vanhoucke, Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. Journal of Scheduling, 2009. 13(1): p. 77-93.
9. Soto, R., et al., Modeling NRPs with Soft and Reified Constraints. AASRI Procedia, 2013. 4(0): p. 202-205.
10. Gîrbea, A., C. Suciu, and F. Şişak, Design and implementation of a fully automated planner-scheduler constraint satisfaction problem. 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), 2011: p. 477-482.
11. Lü, Z. and J.-K. Hao, Adaptive neighborhood search for nurse rostering. European Journal of Operational Research, 2012. 218(3): p. 865-876.
12. Burke, E.K., J. Li, and R. Qu, A Pareto-based search methodology for multi-objective nurse scheduling. Annals of Operations Research, 2012. 196(1): p. 91-109.
13. Brucker, P., et al., A shift sequence based approach for nurse scheduling and a new benchmark dataset. Journal of Heuristics, 2010. 16(4): p. 559-573.
14. Stølevik, M., et al., A Hybrid Approach for Solving Real-World Nurse Rostering Problems, in Principles and Practice of Constraint Programming – CP 2011, J. Lee, Editor. 2011, Springer Berlin Heidelberg. p. 85-99.
15. Valouxis, C., et al., A systematic two phase approach for the nurse rostering problem. European Journal of Operational Research, 2012. 219(2): p. 425-433.
16. Spencer, K.L., L. Ho-fung, and J.H.M. Lee. Guided complete search for nurse rostering problems in Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on. 2005.
17. Burke, E.K., et al., Problem model for nurse rostering benchmark instances. 2009: http://www.cs.nott.ac.uk/~tec/NRP/papers/ANROM.pdf [last accessed on: 2nd July 2014].
18. Burke, E.K., Curtois, T., Post, G., Qu, R., Veltman, B.: A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operation-al Research 188, 330-341 (2008)
19. Burke, E., Cowling, P., De Causmaecker, P., Berghe, G.V.: A memetic approach to the nurse rostering problem. Applied intelligence 15, 199-214 (2001)

20. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A Time Pre-defined Variable Depth Search for Nurse Rostering. (2007)
21. Hadwan, M., Ayob, M., Sabar, N.R., Qu, R.: A harmony search algorithm for nurse rostering problems. Information Sciences 233, 126-140 (2013)
22. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A scatter search methodology for the nurse rostering problem. Journal of the Operational Research Society 61, 1667-1679 (2009)
23. Métivier, J.-P., Boizumault, P., Loudni, S.: Solving Nurse Rostering Problems Using Soft Global Constraints. In: Gent, I. (ed.) Principles and Practice of Constraint Programming - CP 2009, vol. 5732, pp. 73-87. Springer Berlin Heidelberg (2009)
24. Laburthe, F., Jussien., N.: CHOCO solver documentation. (2012)
25. Beldiceanu, N., Carlsson, M., Rampon, J.-X.: Global Constraint Catalog. (2014)
26. Haspeslagh, S., Causmaecker, P.D., Stolevik, M., Schaerf, A.: INRC-First International Nurse Rostering Competition 2010. (2010)
27. Mittelmann, H.D.: Decision Tree for Optimization Software. (2014)
28. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153, 3-27 (2004)