A hybrid approach for solving real-world nurse rostering problems

Martin Stølevik, Tomas Eric Nordlander, Atle Riise, and Helle Frøyseth*

SINTEF ICT, Department of Applied Mathematics,
P.O. Box 124 Blindern, NO-0314 Oslo, Norway
{Martin.Stolevik,Tomas.Nordlander,Atle.Riise}@sintef.no
*hellef@gmail.com
http://www.SINTEF.no

Abstract. Nurse rostering is the process of creating a plan for nurse working hours over a given time horizon. This problem, most variants of which are NP-hard, has been studied extensively for many years. Still, practical nurse rostering is mostly done manually, often by highly qualified health care personnel. This underlines the need to address the challenges of realistic, applied nurse rostering, and the implementation of advanced rostering methods in commercial software.
In this paper, we present an industrial case study of a nurse rostering software currently used in several hospitals and other health care institutions in Norway and Sweden. The presented problem model has a rich set of hard and soft constraints, as required by Norwegian hospitals. Our solution approach is a hybrid: An Iterated Local Search framework that uses Constraint Programming for initial solution construction and diversification, and a Variable Neighborhood Descent for iterative improvement. The search method shows good results in terms of solution quality and computation time on a set of real world instances. We make these test instances available on-line.

Keywords: Nurse rostering, Iterated Local Search, Constraint Programming, Variable Neighborhood Descent, Hybrid optimization, Real-world test cases, Industrial case study.

1 Introduction

Nurse rostering is the process of creating a work schedule for hospital nurses by matching employees to shifts over a given planning horizon, while considering skills, competence, fairness, and laws and regulations. The output is a roster of the working hours for the nurses that also provides an overview of staff utilization and associated costs.

Producing a nurse roster is a complex task: The hospital typically has a continuous demand for personnel, and this demand varies over time. In addition, the roster must follow labor laws, union regulations, as well as hospital policy. Also, the problem involves multiple stakeholders (employer, employees, and patients), whose preferences must be taken into consideration. The main focus of the employer might be efficient resource utilization at minimum cost, while an employee would like to have a fair distribution of work load and the option to influence the planning of his or her days off. For the patients, short waiting time and treatment by personnel with the right competence could be the main concern.

Currently, nurse rostering is usually conducted manually, often by highly qualified health care personnel. There are important advantages to be gained by automating the construction and maintenance of rosters. Not only can the added computation power lead to better rosters, it also drastically reduces the time used in this task. Thus, time is freed up for the involved health care personnel — time that can be better spent on clinical tasks and care.

The presented work was done in a R&D project for Gatsoft AS, a developer of personnel management software that currently serves 80 % of the Norwegian Hospital market. The solution approach showed very good results and was implemented in their personnel management system already in 2005. Today, the module is used by several hospitals and other health care institutions in Norway and Sweden, and has also attracted interest from food production and transportation companies.

In this paper we present the developed problem model and solution approach. These were developed to meet performance requirements and functional specifications from Norwegian hospitals. The solution approach is a hybrid between Constraint Programming, Iterated Local Search [20] and Variable Neighborhood Descent [13], and will be presented in the following. The test results (section 5) show that this search method can solve large, realistic instances within reasonable time. All test instances are from real world rostering applications, and are available online [25].

The paper is organized as follows: Section 2 provides background information on related research. In section 3, we describe the problem, while the solution method is presented in section 4. Experimental results are given in section 5. We conclude and highlight possible directions for future research in section 6.

2 Background

Nurse rostering problems (NRPs) are combinatorial optimization problems that in most cases are NP-hard [17]. There is a large literature on different solution methods applied on NRPs. However, according to [8] and [18], only a few methods have been tested on real world instances and even fewer have been implemented and used in hospitals: For a more comprehensive overview of the nurse rostering literature, see [8].

2.1 Related Work

Both complete and incomplete solution methods have been applied to the NRP. Examples of complete methods are: Mathematical Programming [3], Goal Programming [11], and Constraint Programming [14, 29]. Examples of incomplete methods includes: Variable Neighborhood Search [7], Simulated Annealing [5], Genetic Algorithm [23], Tabu Search [6, 12], and Multi-Objective Optimization with Variable Neighborhood Search [9].

Our approach is a hybrid, combining Constraint Programming (CP) with Iterated Local Search (ILS) and Variable Neighborhood Descent (VND). Similar hybrid approaches include [19], who combine CP with Tabu Search. However, while they formulate a weighted Constraint Satisfaction Problem, we differentiate between hard and soft constraints in a manner similar to that of CP's constraint hierarchy [4]. There is also a difference in CP heuristics: [19] involve only some of the nurses in their CP model, while our CP search constructs complete solutions for all nurses. Also, while they use Tabu Search for improving their partial solution, we use ILS to improve on our complete solution. [28] present an early and interesting hybrid approach: They use a simplified Integer Linear Programming (ILP) formulation of the complete model to produce an initial solution. Tabu Search is used in a second stage to repair and improve on the solution found by ILP. Another relevant approach is [15], where Large Neighborhood Search is used as the algorithmic framework. Here, fragments of low quality in the solution are destroyed, and CP assists by rebuilding the solution. [7] has the approach most similar to ours. They use an iterative process of Variable Neighborhood Search (VNS), then destroy (un-assign) shifts for the most penalized nurses. However, their re-construction is done by a heuristic construction method rather than a CP search, and the solutions so constructed are allowed to be infeasible. These infeasibilities are subsequently removed by the next application of VNS.

There are few systems implemented and used in hospitals. For example, Gymnaste [22], Interdip [1], and Orbis:Dienstplan [2], which all use CP as their solution method. In addition, we have ORTEC's rostering software Harmony [23], which use Genetic Algorithm as its solution method. The only hybrid we have found is Plane [6], which combine simple Tabu Search with problem solving heuristics (diversification and greedy shuffling).

3 Problem description

Based on extensive dialog with Norwegian hospitals, we believe that the proposed model includes all important constraints for Norwegian nurse rostering. In the following, E is the set of all employees, while D is the set of all days in the roster. Table 1 lists the hard and soft constraints of the model.

Each shift is a member of one and only one *shift category*, which is a "collection" of shifts that are concurrent (day shifts, evening shifts and night shift). A *manpower plan* (cover requirement) is a table summarizing how many employees that are needed for each shift on the different weekdays. The inclusion and parametrization of individual constraints will vary between problem instances. A feasible solution must satisfy all included hard constraints. The objective function of the problem, f, is a weighted sum of the penalties derived from violating the soft constraints. For an exact mathematical definition of the problem, please see [26].

Hard constraints	**Soft constraints**
HC1: Maximum one shift is assigned on each day to each employee.	**SC1:** Avoid too many consecutive working days with the same shift category.
HC2: The manpower plan must be covered exactly on each day ('cover requirement').	**SC2:** Avoid too many consecutive working days.
HC3: The sum of working hours for each employee must not deviate too much with respect to the employee's contracted hours (typical for Norway). Note that within the hard limits of HC3, the deviation is minimized by soft constraint SC6.	**SC3:** Avoid too few consecutive working days with the same shift category.
HC4: Employees can only work shifts for which they have the required competence.	**SC4:** Avoid too few consecutive working days.
HC5: There must be a minimum time between shifts on consecutive working days.	**SC5:** Minimize deviation from minimum and maximum number of shifts in each category.
HC6: Every week must have a minimum continuous free period.	**SC6:** Minimize deviation from the employee's contracted hours. While HC3 provide a minimum and maximum limit for the sum working hours, this constraint tries to minimize the distance to the contracted number of working hours within those limits.
HC7: The maximum weekly working time must not be violated.	**SC7:** Cluster days off as much as possible.
	SC8: Maximize wanted shift patterns.
	SC9: Minimize unwanted shift patterns.

Table 1. Hard and soft constraints.

4 Solution method

We employ a hybrid solution approach combining Constraint Programming (CP) and Variable Neighborhood Descent (VND) in an Iterated Local Search (ILS) framework. All parts of this hybrid algorithm were implemented using SINTEF's in-house optimization library, SCOOP.

keywordstyle

```
IteratedLocalSearch
    $\mathbf{x}^* \leftarrow \mathbf{x} \leftarrow$ CPBuild($\mathbf{x}_0$)
    repeat
        $\mathbf{x}$   $\leftarrow$ VariableNeighborhoodDescent($\mathbf{x}$)
        $\mathbf{x}^*$  $\leftarrow$ Accept($\mathbf{x}$, $\mathbf{x}^*$)
        repeat // Diversification
            $\mathbf{x}'$  $\leftarrow$ DestroyPartsOfSolution($\mathbf{x}$)
            $\mathbf{x}'$  $\leftarrow$ CPBuild($\mathbf{x}'$)
        until $\mathbf{x}'$ is a legal solution
        $\mathbf{x} \leftarrow \mathbf{x}'$
    until some termination condition is met
    return $\mathbf{x}^*$
```

Listing 1.1. The iterated local search scheme.

Listing 1.1 shows a high level pseudo-code of the algorithm. The search is initiated in line 2 where CP is used to create an initial feasible solution. This becomes the starting point of the improvement phase in the ILS algorithm. More details on the procedure *CPBuild* can be found in Section 4.1. In line 4, VND is run to improve the current solution $\mathbf{x}$. For more details about the VND algorithm, see Section 4.2. The VND can get stuck in local optima and the purpose of the diversification step in lines 7 and 8 is to escape these. We employ a ruin (*DestroyPartsOfSolution*) and recreate (*CPBuild*) methodology for this diversification; see section 4.3 for a detailed description. The procedure *Accept* (line 5) simply accepts $\mathbf{x}$ as the new best solution if it improves the objective value. The search terminates when a pre-set time limit is reached, when a solution with zero penalty is found, or upon manual interruption by the user. Throughout the search we only store the best found solution ($\mathbf{x}^*$) at any time. Inferior solutions found during the search are discarded.

4.1 Initial solution construction

The procedure *CPBuild* applies CP search on a Constraint Satisfaction Problem (CSP) involving only the hard constraints. In line 2 in Listing 1.1, *CP-Build* constructs a complete feasible initial solution from scratch. We will later (in section 4.3) describe how *CPBuild* is used in the diversification step of the algorithm, to complete partial solutions where only some of the variables are instantiated.

The CSP model has a set of variables $X = \{X_{ed}\}$, where $e \in E$ and $d \in D$ indicates the corresponding employee and day, respectively. Each variable has a finite domain of discrete shift code values. A set of hard constraints restrict the domain values that any subset of variables can take simultaneously. A feasible solution contains an assignment of a shift code value to every variable X_{ed} in such a way that all the hard constraints are satisfied, and the procedure terminates as soon as as such a solution is found. If the procedure cannot find a feasible solution within an allocated time limit it will return a solution satisfying as many of the hard constraints as possible. This is ensured by first finding a solution that satisfy the two hard constraints, HC2 and HC3 (and HC1 implicitly by modelling). For our test cases, finding such "basic solution" is always possible and in general fast, if the rostering problem is set up with the right amount of personnel (which it typically true since this is used in hospitals where the cover / manpower plan is adjusted to the amount of available personnel). Next we try to find a solution where all the hard constraints in the problem are satisfied, if that works, this solution becomes the initial solution. If not; different combinations of the constraints are tried, in a pre-defined order. When the time limit is reached, the solution involving the most important / highest ranked constraint combination is returned.

The hard constraints HC1 and HC2 in Table 1 are always required to be satisfied and to ensure this, the time limit can be exceeded. Those hard constraints that are not satisfied are relaxed and added to the set of soft constraints that form the basis for the objective function used in the subsequent local search. Note that this happens very rarely, and did not happen in any of our tests.

Therefore, in the following discussion we assume that *CPBuild* always returns a feasible solution, satisfying all hard constraints.

We use a MAC CP search algorithm (from our SCOOP library) that establish arc consistency before and search and maintain it during search. The algorithm uses a depth first search with dynamic variable and value orderings. A standard CP search algorithm usually first selects a variable to instantiate, and then a value (or vice versa). In our CP search algorithm the variable and value selection is interconnected: The algorithm first partially orders variables by day, then orders shift values by criticality, and finally orders the variables by employees. The details of this selection procedure are as follows:

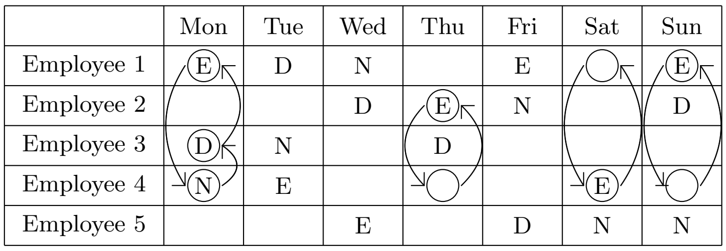
1. **Select days, most critical first.** The first variables to consider are those concerning weekend days. This is because the model requires that "weekends off" for the employees are specified as problem input. Thereafter the weekday variables are ordered in sequence of the last Friday to the first Monday of the plan.
2. **Select shift, most critical shift category first.** The shift categories are first ordered by descending value of the ratio p_{dc}/n_{dc} where p_{dc} is the required number of shifts of category c left to assign on day d and n_{dc} is the number of employees with shifts of category c in its domain on day d in the current solution.
   For example, assume that there are five night shifts and two day shifts left to assign on day d. Eight employees have night shifts (c=1) and four employees have day shifts (c=2) in their domain on day d. By looking at the ratios $p_{d1}/n_{d1} = 5/8 > p_{d2}/n_{d2} = 2/4$, we see that the next shift to assign will be a shift from the night shift category. We use a lexicographical ordering of shift codes within each shift category.
3. **Select employee for the shift according to employee's 'need'.** The selected shift is assigned to the variable on the selected day that correspond to the employee that 'needs' this shift the most. We define m_{ec} as the desired number of shifts of category c for employee e. Initially, m_{ec} is computed by adding all shifts required from the manpower plan for all days and all employees over the planning horizon and computing employee e's share according to his/her contract. This number is updated during the search, as shifts are assigned to the employee. Following the previous partial ordering described above, the variables are now ordered by descending m_{ec} value for the corresponding employees. If we continue with the previous example, then for day d, the next shift to assign is from the night shift category (c=2). We assign it to the employee e with the highest m_{e2}.

4.2 Variable Neighborhood Descent

When the construction algorithm *CPBuild* has created a feasible solution, either as an initial solution (line 2) or as part of the diversification step (line 8), VND is applied to locally refine the solution. The basic VND algorithm is described in [13]. Our implementation cycles through a set of basic neighborhoods, performing a first-improvement Descent local search for each of them. The search terminates when no improving neighbors can be found in any basic neighborhood (i.e. in a local optimum), or when the total improvement over a certain number of iterations is less than a set fraction of the best found objective value ('flattening'). We use the following three basic neighborhoods:

1. **2-Exchange**: This neighborhood consists of all moves where two shifts are swapped on the same day between two different employees, as illustrated in Thursday's column in Figure 1.
2. **3-Exchange**: All moves where three shifts are swapped on the same day between three different employees, as illustrated in Monday's column in Figure 1.
3. **Double 2-Exchange**: All moves that swap shifts between two employees on two days. Such moves are made up of two 2-Exchange moves on different days for the same two employees. Note that the two days are not necessarily consecutive. The two shifts that are moved must belong to the same shift category. This move is illustrated in Saturday's and Sunday's columns in Figure 1.

Before we added the Double 2-Exchange neighborhood, we experienced that the hard constraint concerning working hours for employees (HC3) often prevented the removing or adding of a free-shift to a roster using 2-Exchange. The Double 2-Exchange operator makes it easier to preserve the working hours when moving a free-shifts because exchanging a working shift with a free shift on one day, will be coupled with a free shift being exchanged with a working shift on the other day in the move. This preserves the number of shifts of different categories for both employees, thus improving the likelihood of an improving move.



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Employee 1	E	D	N		E		E
Employee 2			D	E	N		D
Employee 3	D	N		D			
Employee 4	N	E				E	
Employee 5			E		D	N	N

Fig. 1. Example of 2-Exchange (Thursday) and 3-Exchange (Monday), and Double 2-Exchange (Saturday+Sunday) neighborhoods. All blank cells are free shifts.

The above neighborhoods are used in many local search applications for the nurse rostering problem. The most basic move operator in nurse rostering is the replace move [21] which corresponds to (in our model) moving a shift from one employee to another. The 2-Exchange move can be viewed as a combination of two opposite replace moves.

Note that soft constraint SC8 is challenging for local search methods: The obvious objective function to use is the number of complete wanted patterns in the solution. However, such a function is lacking because its value is only reduced when a new complete pattern is found. This motivates keeping patterns that are already found, but there is no explicit mechanism that actually drives the local search in the direction of completing patterns. To introduce such a driving mechanism in our method, we use a modified penalty function [26] which decreases the penalty as we get closer to completing a wanted pattern. For instance, let D,D,D,E be a wanted pattern: When evaluating part of a roster with D,D,F,E (3 out of 4 shifts correct) against one with N,D,F,E (2 out of 4 shifts correct) the first combination (D,D,F,E) receives the lowest penalty. We use a fixed number from which we subtract the squared (normalized) Hamming distance. This is then a function with the property that a pattern with two wrong shifts are penalized harder than two patterns each with one wrong shift.

4.3 Neighborhood reduction

The neighborhoods, described in Section 4.2, are rather large. For example, the 2-Exchange neighborhood has a size of order $|E|^2|D|$. We use *focal points* to significantly reduce the size of the neighborhoods. A focal point identifies features in the solution which is expected to be critical to further improvement of the objective value. We create one focal point for each variable that is involved in a violation of one or more of the soft constraints. During the VND search, each neighborhood is reduced to those moves that somehow involve one or more focal points. When a move is performed, the list of focal points is updated. This is comparable to the "Cost-based Neighborhood" idea presented by [10] in which the authors focus the search effort on the part of the problem which has the greatest effect on the objective. In that work, however, more effort was done in evaluating several candidate improvements, in contrast to our first-improvement strategy. Analogous ideas exists in the project scheduling literature, where local search neighborhoods can be focused on critical paths in the project graph. Also, the Fast Local Search algorithm of [27] is similar.

4.4 Diversification

The purpose of the diversification step (line 7 and 8 in Listing 1.1) is to escape local optima and areas of the search space where little improvement is found ('flattening'). We do this by making a major change to the current solution by ruin-and-recreate [24]. The "ruin" mechanism removes the shift assignments for a subset (E') of the employees. A partial CSP solution is created, based on the partially ruined solution ($\mathbf{x}'$), where some of the variables are instantiated while those variables involved in the above "ruin" process get their full, initial domains. This partial solution becomes the input to *CPBuild* which then constructs a new feasible solution.

The number of rosters to ruin is picked randomly between 2 % and 30 % of the total number of rosters. Half of the rosters to ruin are randomly selected, while the other half are those that produce the highest penalties. We focus partly on the parts of the solution that have the largest potential for improvement — similar to the focal points of the VND, but also include some randomly chosen rosters to avoid recreating the same, high penalty solution, and intensify the diversification.

If a feasible solution is not found by *CPBuild* within a given time limit, the algorithm retries *DestroyPartsOfSolution* with a different selection of randomly chosen rosters. The time limit is equal to the time used when creating the initial solution. Our experience is that this happens very rarely.

5 Computational experiments

5.1 Test cases

The presented problem model and solution method were designed for integration in the leading nurse rostering system in the Norwegian market. The aim in this context was to develop a robust solution method that could solve a wide range of realistic problem instances as they occur in the hospitals. The model has a very similar structure to those found in common academic benchmark problems such as those at http://www.cs.nott.ac.uk/~tec/NRP/. There are, however, some differences. For example, most benchmark problems involve time limited "horizontal" constraints (constraints for one employee) while our horizontal constraints spans over the complete planning horizon (except for HC7). Furthermore, we assume the assignment of free weekends to be part of the problem input, which is not common in benchmark problems. Also, our functional requirement demanded that the manpower plan (cover requirement) must be covered exactly — it is a hard constraint that we implicitly handle by algorithm design. In the literature, some flexibility in the coverage is normally allowed, often modeled as a soft constraint.

Table 2 describe our seven test cases (OpTur1 – OpTur7). The first four rows provide general information for each case (e.g. number of employees to schedule, number of hours for the complete scheduling period, etc.). The following rows provide the parameterization for the constraints (see Table 1 in section 4 for more information about the individual constraints). In Table 2, the shift categories "Day", "Evening", and "Night" are represented by "D", "E", and "N". For example: For the case OpTur2, the hard constraint "HC5: N $\rightarrow$ E" forces the minimum time between consecutive shifts in the night category (N) and the evening category (E) to be at least 8 hours. An empty cell signifies that the corresponding constraint is not used in the test case. The notation '10/8' in for instance OpTur6's three HC5 constraints, sets this minimum time to 10 and 8 for weekends and weekdays, respectively. The symbol '-' means that the constraint parameter is not set (e.g. SC1 in OpTur2 has no specific maximum limit for day shift (D) while it only allows 4 consecutive evening shifts (E) and night shifts (N).

Each of our seven test cases contain a mix of employee contracts. E.g. most of the 51 employees in OpTur1 are on standard Norwegian nursing contracts with an average 35.5 hours working time is s. The same test case also involves contracts with an average of 26.6, 17.7, 8.9 and even 4.4 hours/week. Other important aspects of each test case concerns skills, nurse contracts, shifts, manpower plans patterns, etc. A complete description of all test cases can be found in [25].

	OpTur1	OpTur2	OpTur3	OpTur4	OpTur5	OpTur6	OpTur7
General information:							
# of employees	51	83	29	30	20	54	15
# of hours to schedule	19170	12819	4159.5	17352	2280	18978	2209.5
# days to schedule	84	42	42	168	28	84	42
# of shifts in use	9	9	8	8	9	5	6
Hard constraints:							
HC1	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HC2	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HC3	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HC4	Yes						Yes
HC5: (N $\rightarrow$ E)	8	8	11	11	11	10/8	11
HC5: (E $\rightarrow$ D)	8	9	11	9	8	10/8	11
HC5: (D $\rightarrow$ N)	8	8	11	11	11	10/8	11
HC6:	32	32	32	32	32	32	32
HC7:	54	50	48	48	48	54	48
Soft constraints:							
SC1: (D,E,N)	6,3,4	-,4,4	5,2,4	4,3,4	3,2,3	-,-,3	-,-,4
SC2:	6	6	12	7	5		6
SC3: (D,E,N)		-,-,2	2,2,2	-,-,2	2,-,2		-,-,2
SC4:	(Not used in any of the test cases)						
SC5: D			0/10				
SC5: E			0/10		0/5		
SC5: N			3/10		2/6		
SC6:	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SC7:		Yes	Yes	Yes	Yes	Yes	Yes
SC8:	1 p.		1 p.				3 ps.
SC9:	(Not used in any of the test cases)						

Table 2. Overview of the seven test cases; characteristics and parameterization of the constraints used.

5.2 Experimental results

We have conducted the experiments described in this paper on a PC (Intel Core2 Duo CPU at 2.53 GHz with 4 GB RAM) running 64-bits Windows 7. Through extensive testing we have determined the values for the different parameters of the algorithm. The most important of these are: first-improvement (rather than best-improvement) in the VND, the number of rosters to destroy in the diversification step (random number between 2 % and 30 %), and the "flat" criterion in VND (less than 5 % change in the objective value over the last n iterations, where $n = 50\sqrt{|D||E|}$).

The algorithm contains random elements in the neighborhood generation for VND and in the ruin part of diversification (see Listing 1.1, Section 4).

Therefore, to statistically verify our results, we have run every test case 66 times (the exact number was determined by our experimental setup), terminating each run after 1200 seconds. Table 3 presents some statistics of the results, taken over all runs. Line 2 shows the number of backtracks in the CP search used to compute an initial solution. The following rows present time spent (Mean, standard deviation, and coefficient of variation) on each of the following parts of the iterated local search: Generate Initial Solution (line 2), VND (line 4) and Diversification (line 7 + 8). The row "Initial Obj val" shows the objective value after the initial solution generation. The five last rows show statistics for the objective values found after 1200 seconds.

	OpTur1	OpTur2	OpTur3	OpTur4	OpTur5	OpTur6	OpTur7
# backtracks	199106	3994	160100	52133	294	1274	0
Initial solution							
Mean (sec)	59.0	1.87	33.8	17.2	0.131	0.818	0.0694
StDev (sec)	6.25	0.0162	0.184	0.164	0.00594	0.0124	0.00713
CV (%)	10.6	0.866	0.546	0.954	4.54	1.52	10.3
VND							
Mean (sec)	1141	1188	1144	1181	1104	1199	883
StDev (sec)	6.88	6.86	8.21	1.08	9.59	2.16	27.8
CV (%)	0.603	0.578	0.717	0.0914	0.869	0.180	3.15
Diversification							
Mean (sec)	0.472	10.3	22.4	2.00	96.3	1.19	317
StDev (sec)	1.79	6.88	8.27	1.02	9.53	2.15	27.8
CV (%)	379	66.6	36.9	50.9	9.90	181	8.76
Initial Obj val	428	162	253	203	253	48.0	378
Final Obj val							
Mean	17.0	3.09	81.7	2.48	8.34	1.83	156
StDev	6.95	0.190	1.79	0.503	1.99	0.0759	0.303
CV (%)	40.8	6.16	2.20	20.2	23.8	4.16	0.194
Min	13.1	2.82	78.1	1.56	4.48	1.66	156
Max	42.3	3.53	86.1	3.78	12.9	1.97	157

Table 3. Results for the seven test cases: Number of backtracks. Time spent on the initial solution generation, variable neighborhood search, and diversify parts of the algorithm, Last rows provide information concerning the objective value. All cases were run 66 times and for 1200 seconds.

Finding a solution with an objective value of zero on these over-constrained instances is very unlikely, as is the case in most real-world problems. Several of the soft constraints are easily parameterized in such way that violation of some soft constraints cannot be avoided. The results show that there are differences in how difficult it is to compute the initial solution. The number of backtracks (and thus time spent) during initial solution construction varies from almost nothing for OpTur5 and OpTur7, to several hundred thousand for OpTur1 and OpTur4. This is partly connected to the problem size and how constrained the problem is. But also because the diversification step is run many times, and the number of times differ between the cases. (The number depends on how often the variable neighborhood descent will end up in a local optimum or on a "flat" plateau.) If the diversification step is run a large number of times, the time spent on diversification will increase compared to the initial solution construction which is a one-time action. The most important observation to make, however, is that the time used to generate a feasible initial solution is very fast (less than a minute on average for all cases) compared to manual rostering, in which one typically use days to set up a feasible roster. In this sense, the performance of the construction algorithm is more than adequate.

After 1200 seconds, the objective value of the initial solution was drastically improved by the ILS algorithm, in most cases by more than 96 %. This is not surprising, since the initial solution was constructed without considering soft constraints. The exceptions are the cases OpTur7 ($378 \rightarrow 156$) and OpTur3 ($253 \rightarrow 81.7$), for which the improvements were 58.7 % and 67.7 %, respectively. Both OpTur 3 and OpTur7 contains wanted patterns (SC8) for the weekends.

To assess how the algorithm improves the objective value over time, one can consider the run-time distributions for each test case [chap. 4.2] [16]. Figure 2 shows such distributions for the case OpTur6, each curve representing the observed cumulative fraction of all runs reaching the corresponding objective value threshold as a function of computation time. Note that since we do not know the optimal value, solution quality thresholds are given in terms of percentual deviation from the best found value in any run. Observe that all runs achieved an objective value of 24 % deviation from the best known value, or better, in the first 162 seconds. After 1200 seconds, 10.6 % of the runs had passed the 3 % deviation threshold, while 25.8 % of the runs resulted in an objective value of less than 6 % above the best known objective value.

For some cases there is substantial variation in the objective value across runs, especially for OpTur1 ($\mathrm{cv} = 40.8$ %), OpTur4 ($\mathrm{cv} = 20.2$ %), and OpTur6 ($\mathrm{cv} = 23.8$ %). For OpTur1, this seems to happen because the best found value for each run ends up in one of three objective value ranges; 13.1 – 14.1, 18.2 – 19.8 or above 36.3. This may indicate that there are some distinct valleys or plateaus in the objective function surface of this problem instance. For the two other test cases, however, the observed objective values at the time of run termination are more or less evenly distributed between the minimum and the maximum values.

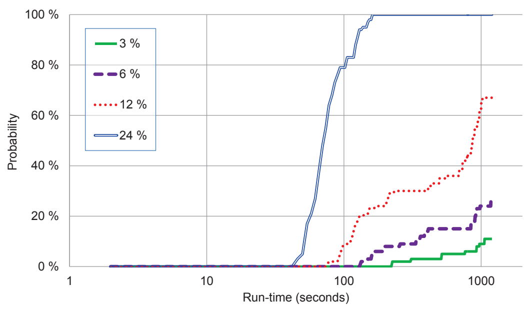


Fig. 2. Qualified Run-Time Distribution of OpTur6 for solution quality levels of 3 %, 6 %, 12 % and 24 % of the best found objective value, across all runs.

6 Conclusions and Future work

The case study presented here concerns a nurse rostering module that was developed for the software company Gatsoft AS and implemented in their personnel management system in 2005. Today, the module is used by several hospitals and other health care institutions in Norway and Sweden. The module applies a hybrid solution approach: An Iterated Local Search framework that uses Constraint Programming for initial solution construction and diversification, and a Variable Neighborhood Descent in the iterative improvement phase. The requirements and specifications for the model and the algorithm evolved from regular meetings and workshops with health care personnel. We believe it includes all the important constraints that are applied in Norwegian hospitals. In this paper, we show that the module can solve large real-world instances within reasonable time.

Further research involves adapting the model to handle a rigorous testing on the standard academic benchmarks. To further improve performance, we aim to develop massively parallel algorithms that exploit the computation power of emerging heterogeneous hardware platforms, where graphical processing units and multiple CPUs can be used together to produce high performance search methods.

It is also important to extend the research context to encompass related and practical extensions of the problem. For example, by introducing flexibility in the model's shifts by allowing their length, start, and end time to be dynamically adjusted: A small changes in start or end times may allow for plans that violate fewer soft constraints without affecting patient quality or increasing personnel costs.

Also, nurse rosters are typically static, while the daily situations at a hospital is very dynamic — employees get sick, take days off on a short notice, or the staff demand temporarily increases. In Norway, the resulting personnel shortages are normally filled through temporary work agencies, which is not only very expensive but quite inefficient. This opens up interesting research directions, such as robust nurse scheduling to minimize the impact due to dynamic events. A related problem is the dynamic re-scheduling of nurses across departments to minimize the impact of unexpected events while maximizing the competence build-up to obtain a more robust future personnel structure.

Acknowledgment This work is supported by the Research Council of Norway (grant no. 182610/I40) and Gatsoft AS. We wish to thank Morten Smedsrud for assisting us with coding and reading the manuscript.

References

1. Slim Abdennadher and Hans Schlenker. Nurse scheduling using constraint logic programming. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 838–843, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.
2. Harald Meyer auf'm Hofe. Conplan/siedaplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *Proceedings on the 3rd International Conference on Practical Applications of Constraint Technologies, pages 257-272, Practical Application Company Ltd*, 1997.
3. Nicholas Beaumont. Scheduling staff using mixed integer programming. *European Journal of Operational Research*, 98(3):473 – 484, 1997.
4. Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *LISP and Symbolic Computation*, 5(3):223–270, 1992. 10.1007/BF01807506.
5. Michael J. Brusco and Larry W. Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organizations. *European Journal of Operational Research*, 86(2):249 – 261, 1995.
6. Edmund K. Burke, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *SEAL'98: Selected papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning*, pages 187–194, London, UK, 1999. Springer-Verlag.
7. Edmund K. Burke, Timothy Curtois, Gerhard Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.
8. Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
9. Edmund K. Burke, Jingpeng Li, and Rong Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484 – 493, 2010.
10. Tom Carchrae and J. Christopher Beck. Cost-based large neighborhood search. In *Workshop on Combination of metaheuristic and local search with Constraint Programming techniques*, pages 28–29, 2005.
11. J.-G. Chen and T. Yeung. Hybrid expert system approach to nurse scheduling. *Computers in Nursing*, pages 183–192, 1993.
12. Kathryn A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106(2-3):393 – 407, 1998.

13. Pierre Hansen and Nenad Mladenovic. Variable neighborhood search. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, pages 211–238. Springer, 2005.
14. H. Hattori, T. Ito, T. Ozono, and T. Shintani. A nurse scheduling system based on dynamic constraint satisfaction problem. In *Innovations in Applied Artificial Intelligence*, volume 3533 of *Lecture Notes in Artificial Intelligence*, pages 799–808. Springer-Verlag Berlin, Berlin, 2005.
15. Fang He and Rong Qu. A constraint-directed local search approach to nurse rostering problems. In Yves Deville and Christine Solnon, editors, *Proceedings 6th International Workshop on Local Search Techniques in Constraint Satisfaction*, pages 69–80, 2009.
16. Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
17. Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
18. Deborah L. Kellogg and Steven Walczak. Nurse Scheduling: From Academia to Implementation or Not? *INTERFACES*, 37(4):355–369, 2007.
19. Haibing Li, Andrew Lim, and Brian Rodrigues. A hybrid ai approach for nurse rostering problem. In *Proceedings of the 2003 ACM symposium on Applied computing*, SAC '03, pages 730–735, New York, NY, USA, 2003. ACM.
20. Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. *Iterated Local Search*, pages 321–353. Kluwer Academic Publishers, 2003.
21. Amnon Meisels and Andrea Schaerf. Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence*, 39(1):41–59, 2003.
22. K. Heus P. Chan and G. Weil. Nurse scheduling with global constraints in chip: Gymnaste. In *In Practical Applications of Constraint Technology (PACT)*, 1998.
23. Gerhard Post and Bart Veltman. Harmonious personnel scheduling. In *PATAT 2004 - Proceedings of the 5th international conference on the Practice And Theory of Automated Timetabling*, pages 557–559, 2004.
24. Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139 – 171, 2000.
25. Martin Stølevik, Tomas Eric Nordlander, and Atle Riise. SINTEF ICT: Nurse rostering data. http://www.comihc.org/index.php/Test-Beds/sintef-ict-nurse-rostering-data.html, 2010. Accessed 2010-10-13.
26. Martin Stølevik, Tomas Eric Nordlander, and Atle Riise. A mathematical model for the nurse rostering problem. SINTEF Technical Report A19133. http://www.comihc.org/index.php/Models/sintef-ict-nurse-rostering-model.html., 2011. Accessed 2011-04-08.
27. Edward Tsang and Chris Voudouris. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters*, 20(3):119 – 127, 1997.
28. C Valouxis and E Housos. Hybrid optimization techniques for the workshift and rest assignment of nursing personnel. *Artificial Intelligence in Medicine*, 20(2):155 – 175, 2000. Planning and Scheduling in the Hospital.
29. Gary Yat Chung Wong and Hon Wai Chun. Nurse rostering using constraint programming and meta-level reasoning. In *IEA/AIE'2003: Proceedings of the 16th international conference on Developments in applied artificial intelligence*, pages 712–721. Springer Verlag Inc, 2003.