An Efficient Translation Scheme for Representing Nurse Rostering Problems as Satisfiability Problems

Stefaan Haspeslagh\textsuperscript{1,}, Tommy Messelis\textsuperscript{1,}, Greet Vanden Berghe\textsuperscript{2} and Patrick De Causmaecker\textsuperscript{1,2}
\textsuperscript{1}CODeS (member of ITEC-IBBT-KU Leuven), Department of Computer Science, KU Leuven KULAK, E. Sabellaan 53, 8500 Kortrijk, Belgium
\textsuperscript{2}CODeS, Vakgroep IT, KAHO, Gebr. De Smetstraat 1, 9000 Gent, Belgium

Keywords: Nurse Rostering, Satisfiability Problems, Automated Translation, Counting Constraints.

Abstract: In this paper we present efficient translation schemes for converting nurse rostering problem instances into satisfiability problems (SAT). We define eight generic constraints types allowing the representation of a large number of nurse rostering constraints commonly found in literature. For each of the generic constraint types, we present efficient translation schemes to SAT. Special attention is paid to the representation of counting constraints. We developed a two way translation scheme for counting constraints using $O(n\log n)$ variables and $O(n^2)$ clauses. We translated the instances of the First international nurse rostering competition 2010 to SAT and proved the infeasibility of the instances. The SAT translation was used for a hardness study of nurse rostering problem instances based on SAT features.

1 INTRODUCTION

We present schemes for automatically translating nurse rostering problem instances into satisfiability problem instances (SAT).

The SAT-translation scheme was originally designed for a hardness study of nurse rostering problem instances (Bilgin et al., 2009). Some algorithms perform well on some specific instances while those same algorithms perform worse on others. (Leyton-Brown et al., 2006) present an experimental approach for predicting the runtime of algorithms designed to solve the winner determination problem for combinatorial auctions. The previous strategy was applied to the more abstract problem of propositional satisfiability (Nudelman et al., 2004) introducing a set of 91 features for the runtime prediction of several algorithms that prove whether a certain problem instance is satisfiable or not. This work has led to the construction of SATzilla, a portfolio solver for SAT problems (Xu et al., 2008). This portfolio was very successful, winning several tracks of different SAT competitions\textsuperscript{1}. The aforementioned SAT-features set is used to predict the runtime of a certain algorithm and the value of the objective function obtained by the algorithm.

\textsuperscript{1}More information about these competitions can be found at http://www.satcompetition.org

A second potential application of the SAT translation scheme is applying SAT (or MaxSAT) solvers to nurse rostering problem instances. As a first attempt, (Acharyya, 2008) translates a simple nurse rostering problem with a small set of constraints into SAT and applies GSAT to solve it. The approach is limited to small instances as the number of clauses increases rapidly with the size of the instances. It is thus important to develop efficient encodings of the constraints.

Modeling optimisation problems involving constraints on sequences of decision variables to mixed integer programs (and SAT) can be very complex (Côté et al., 2011). Special attention is given to the translation of counting constraints into SAT clauses. We present an efficient translation procedure generating $O(n^2)$ clauses and $O(n\log n)$ variables. (Bailleux and Boufkhad, 2003) developed a translation scheme with similar complexity. (Sinz, 2005) provides a $O(nk)$ and (Asín et al., 2009) a $O(n\log^2 k)$ scheme to translate the constraint $x_1 + \ldots + x_n \leq k$. All contributions however, only provide a one way translation, preserving arc consistency. As for the hardness studies, we do not solely focus on solving nurse rostering problems using SAT, a two way translation was required.

(Cadoli and Scharf, 2005) present the automated translation of problem specifications expressed in NP-SEC into SAT. NP-SEC in a logic based language ca-
pable of expressing all problems belonging to complexity class NP. The authors tested the system on a few classical problems such as graph colouring and job-shop scheduling. NP-complete problems with a rather simple problem definition. The authors highlight the benefits of an automated approach. Although the performance of SAT solvers on the generated instances is often inferior to manual encodings, the authors stem that the system is a valuable tool for developing fast prototypes for new problems, or variations of known ones for which no specific solver is available.

In Section 2, an abstract representation of nurse rostering problems using numberings is described. Section 3 elaborates on the formal definition of generic constraints types. In Section 4, each generic constraint type is translated to SAT. Section 5 demonstrates the applicability of the SAT translation schemes for a hardness study of nurse rostering problem instances. Section 6 concludes and gives directions for further research.

2 ABSTRACT REPRESENTATION OF NURSE ROSTERING PROBLEMS USING NUMBERINGS

The nurse rostering problem under study is the problem presented in the First international nurse rostering competition (Haspeslagh et al., 2012). The authors used numberings, originally designed for the efficient evaluation of constraint violations (Burke et al., 2001) as an abstract representation of constraints. After we elaborate on the definition of numberings, we give an example numbering for some sample constraints. For a full description of the competition’s constraint, we refer to the original paper.

2.1 Numberings

We start with some elementary notation.

Definition 1. A time unit is an elementary interval of time in which a nurse can be assigned a shift.

In our case, shift types determine those intervals. So, each shift type has a corresponding time unit. In the case of the nurse rostering problems considered within this paper, the number of time units equals the number of shift types times the number of days in the planning period. Thus, supposing we have a planning horizon of D days and for each day there are Sh shift types, we have a set T of D * Sh time units.

A solution to a nurse rostering problem is then an assignment of nurses to shifts on specific time units.

Numberings on the time units are defined as follows:

Definition 2. A numbering N_i is a mapping of the set of time units onto a set of numbers extended with the symbol U i.e. N_i : T → {−M, −M + 1, ..., 0, ..., M − 1, M, U} where i = 1, ..., I and I is the total number of numberings. M is a positive integer and U (undefined) is a symbol introduced to represent the time units that are not mapped onto a number.

The mapping does not need to be into or onto, nor does it need to preserve sequence. An event is a time unit for which a nurse has a shift type assigned. Denote by N_p an event is a time unit e for which S_p(e) = working.

Denote by T_S_p the set of all time units for which S_p maps to working. T_S_p is thus the set of time units induced by S_p or in other words: the set of time units for which nurse p is assigned to work. Denote by T_N_i the set of time units for which the numbering N_i is defined.

Definition 5. The event set T_{S_p, N_i} := T_{S_p} ∩ T_{N_i} is the set of time units induced by the schedule S_p, for which the numbering N_i is defined (≠ U).

Definition 6. Two events e_j and e_{j+1} are consecutive with respect to numbering N_i if N_i(e_{j+1}) − N_i(e_j) = 1

We can now express the following four groups of constraints:

• total number of assignments (total): for a numbering N_i, this constraint type limits the number of events e for which N_i(e) ≠ U.
• total number of assignments of a certain type (per-Type): for a numbering N_i, this constraint type limits the number of events corresponding to a specific number j (N_i(e) = j).
• consecutive assignments (consecutiveness): for a numbering N_i, this constraint limits the length a a sequence of consecutive events in T_{S_p, N_i}.
• gaps between consecutive events (between): for a numbering N_i, this constraint type limits the maximum gap (e.g. free time) between two non-consecutive events e_j and e_{j+1} (the gap equals N_i(e_{j+1}) − N_i(e_j)).
Table 1: Example numbering.

<table>
<thead>
<tr>
<th>Day</th>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>cons</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>last</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

For each constraint type, a maximum and minimum bound can be set: max_total and min_total, max_consecutive and min_consecutive, max_between and min_between and max_pert and min_pert.

Four counting variables are introduced. total represents the total number of events for the numbering. consecutive represents the number of consecutive events. If an interruption in a sequence is found, the counter is reset to 0. pert keeps track of the number of events per value in the numbering and last keeps track of the number of the last evaluated event.

With each numbering N, a set of four of the above counters is associated. During the evaluation, by comparing the counter with one of the above-mentioned constraint types, constraint violations are detected.

An example for a simplified nurse rostering problem with two shift types (ST) is given in Table 1. We want to evaluate two constraints: the employee should work at most two consecutive days (consecutive = 2) and should have at least two consecutive days off (min_between = 2). E shows the assignments to one employee. At time unit 5, a first violation is detected as the value of the counter cons exceeds max_consecutive. At time unit 8, a second violation is detected as (current_number – last > min_between).

3 FORMAL DEFINITIONS OF GENERIC CONSTRAINTS

In this study, we only consider monotonically ascending (Definition 7) numberings.

Definition 7. A numbering N is monotonically ascending if, for every two time units j_a and j_b for which N is defined (≠ U), j_a < j_b ⇒ N(j_a) ≤ N(j_b).

The constraints on the numberings can be expressed using event sets. An important concept in the expression of constraints is the notion of event sequences, in particular sequences that increase at a steady pace.

Definition 8. The event set E of a numbering N and a personal schedule S_p is the set S_p ∪ N.

With every event e ∈ E, a unique number (∈ N) is associated. We refer to Section 2.1 for detailed information on the definition of S_p and T_p ∪ N.

Table 2: Formal definition of eight generic constraints.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mon_consecutive</td>
<td>min_e</td>
<td>There is no continuously ascending event sequence of length min_e.</td>
</tr>
<tr>
<td>max_consecutive</td>
<td>max_e</td>
<td>There is a continuously ascending event sequence of length max_e.</td>
</tr>
<tr>
<td>min_between</td>
<td>min_b</td>
<td>There are at least min_b consecutive events.</td>
</tr>
<tr>
<td>max_between</td>
<td>max_b</td>
<td>There are at most max_b consecutive events.</td>
</tr>
<tr>
<td>min_total</td>
<td>min_t</td>
<td>The event set E contains at least min_t events.</td>
</tr>
<tr>
<td>max_total</td>
<td>max_t</td>
<td>The event set E contains at most max_t events.</td>
</tr>
<tr>
<td>min_pert</td>
<td>min_p</td>
<td>The event set E contains at least min_p pert events.</td>
</tr>
<tr>
<td>max_pert</td>
<td>max_p</td>
<td>The event set E contains at most max_p pert events.</td>
</tr>
</tbody>
</table>

4 TRANSLATION OF GENERIC CONSTRAINTS TO SAT

We present a scheme to translate each of the eight generic constraints from Section 3 to Conjunctive Normal Form SAT clauses (Cook, 1971). A preprocessing step is performed before the translation of the constraint types.

4.1 Preprocessing

We introduce boolean decision variables v_p,j indicating whether nurse p is working time unit j. Since the expression of the sequence constraints is basically the same for all employees, we denote v_p,j as v_j, in order to simplify the notation.

For some numberings, consecutive time units are assigned the same number, e.g. the example numbering N in Table 1 does not distinguish between different shifts on the same day. Hence it is natural to introduce a variable t_i indicating for each sequence of consecutive time units with equal numbers whether an employee is working on a time unit within that sequence. Generally, for such a sequence of time units starting at k and ending at l:
In general, both constraints for a monotonically ascending event sequence of length \( l \) and \( (\alpha \leq \beta < \gamma) \) are present in Table 2. Without the implicit variables, some event sequences would lack. Suppose we want a gap of at most 1 between two consecutive events and an event occurs for \( t_2 \) (\( v_3 = \text{True} \)) or \( t_5 \) (\( v_5 = \text{False} \)). Without the implicit variables, some event sequences would lack. We suppose a gap of at most 1 between two consecutive events and an event occurs for \( t_2 \) (\( v_3 = \text{True} \)) or \( t_5 \) (\( v_5 = \text{False} \)). Between the two events, there is a gap of 2. By omitting the implicit variables, the two event sequences detecting the violation would be missing. The preprocessing step results in at most \( O(n) \) extra variables and as many extra clauses as variables added.

**4.2 Translation of 'Consecutive' and 'between' Constraints**

**4.2.1 Maximum Number of Consecutive Events and Free Time Units between Two Events**

As stated in Table 2 there should not be a continguously ascending event sequence of length \( \text{max} + 1 \). For every continguously ascending sequence \( \text{cas} \) (\( \text{cas}_i \) is the index in the original sequence of the \( i^{th} \) variable within \( \text{cas} \)) we have the following clauses:

\[
\neg \left( \bigwedge_{j=0}^{\text{max}_c} \neg t_{\text{cas}_j} \right), \quad \text{in cnf: } \bigvee_{i=0}^{\text{max}_c} (\neg t_{\text{cas}_i})
\]

Analogously, for a maximum gap between two events, we obtain:

\[
\neg \left( \bigwedge_{j=0}^{\text{max}_c} t_{\text{cas}_j} \right), \quad \text{in cnf: } \bigvee_{i=0}^{\text{max}_c} (t_{\text{cas}_i})
\]

In general, both constraints for a monotonically ascending numbering consisting of \( n \) numbers generates at most \( (n - \text{max}_c) \), respectively \( (n - \text{max}_b) \) clauses and no extra variables.

**4.2.2 Minimum Number of Consecutive Events and Free Time Units between Two Events**

Following Table 2, there should not be a continguously ascending event sequence of length \( l \) (\( \text{min}_c > l > 1 \)) which is not part of another continguously ascending event sequence of length at least \( \text{min}_c \). This implies that in any consecutive sequence of variables \( t_i \) of length \( l \) (\( \text{min}_c + 1 \leq l \geq 3 \)) the middle variables cannot be true without one of the border variables. For every continguously ascending sequence \( \text{cas} \) of length \( l \) (\( 3 \leq l \leq \text{min}_c + 1 \)) this results in the following clauses:

\[
\neg \left( t_{\text{cas}_0} \lor \neg t_{\text{cas}_{l-1}} \land \left( \bigwedge_{j=1}^{l-2} t_{\text{cas}_j} \right) \right)
\]

in cnf: \( t_{\text{cas}_0} \lor t_{\text{cas}_{l-1}} \lor \bigvee_{j=1}^{l-2} t_{\text{cas}_j} \)

Analogously, for a minimum gap between two events, we obtain:

\[
\neg \left( t_{\text{cas}_0} \land \neg t_{\text{cas}_{l-1}} \lor \left( \bigwedge_{j=1}^{l-2} \neg t_{\text{cas}_j} \right) \right)
\]

in cnf: \( \neg t_{\text{cas}_0} \lor \neg t_{\text{cas}_{l-1}} \lor \bigvee_{j=1}^{l-2} \neg t_{\text{cas}_j} \)

In general, both constraints for a monotonically ascending numbering consisting of \( n \) numbers, generate at most \( (n - \text{max}_c) \) clauses and no extra variables.

**4.3 Translation of Counting Constraints**

We developed an efficient general procedure for translating counting constraints into CNF clauses. First we elaborate on the procedure in general. Then we show how to translate total and pert constraint types using this procedure.

**4.3.1 General Procedure**

This procedure uses an iterative process of introducing variables and clauses. We start by some elementary definitions.

**Definition 11.** \( V \) is the set of variables \( v_i \) for which we want to count the number of variables that are assigned true.

**Definition 12.** \( U_{\alpha,\beta} \) is the set of indices (of time units) \( k \) between \( \alpha \) and \( \beta \) for which \( v_k \) in \( V \) is assigned true.

**Definition 13.** \( F_{\alpha,\beta} \) is the set of indices (of time units) \( k \) between \( \alpha \) and \( \beta \) for which \( v_k \) in \( V \) is assigned false.
We then introduce the variables $u_i$, respectively $f_i$, denoting whether there are at least $i$ elements in the set $U_{1,n}$, respectively $F_{1,n}$, with $n$ the number of elements in $V$:
\[ u_i \iff |U_{1,n}| \geq i, \quad f_i \iff |F_{1,n}| \geq i \quad \text{for } i \in \{1, \ldots, n\} \] (1)

More generally, we introduce the variables $u_{x,y,z}$ ($f_{x,y,z}$) denoting whether there are at least $z$ events in the set $U_{x,y,z}$ ($F_{x,y,z}$):
\[ u_{x,y,z} \iff |U_{x,y,z}| \geq z \quad \text{and} \quad f_{x,y,z} \iff |F_{x,y,z}| \geq z \] (2)

In the remainder of this section, we only consider the first equivalence in (2). As $u_i = u_{1,n,i}$, for representing equivalence (1), we show we only need to translate:
\[ u_{1,n,i} \Rightarrow |U_{1,n}| \geq i \] (3)

By contradiction, the other direction of the equivalence is equivalent with
\[ u_{1,n,i} \Rightarrow \neg(|U_{1,n}| \geq i) \] (4)

If no $i$ variables may be assigned $true$, at least $n-i+1$ variables should be assigned $false$:
\[ (|U_{1,n}| < i) \Rightarrow (|F_{1,n}| \geq n-i+1) \iff f_{n-i+1} \] (5)

Using (5), implication (4) becomes:
\[ f_{n-i+1} \Rightarrow |F_{1,n}| \geq n-i+1 \] (6)

This can be translated in the same way as we translate (3).

4.3.2 Translation of $u_{1,n,i} \Rightarrow |U_{1,n}| \geq i$

First we prove the following equivalence:
\[ u_{1,n,i} \iff (u_{1,\frac{z}{2},k} \lor u_{\frac{z}{2}+1,n,i}) \lor k + l = i+1. \]

Lemma 1. $u_{1,n,i} \iff (u_{1,\frac{z}{2},k} \lor u_{\frac{z}{2}+1,n,i})$ for $k + l = i+1$

In words, when $U_{1,n}$ contains at least $i$ elements, then $U_{1,\frac{z}{2}}$ contains at least $k$ elements or $U_{\frac{z}{2}+1,n,i}$ contains at least $l$ elements. We will prove this by contradiction.

Proof. Let $k_0$ be the number of elements in $U_{1,\frac{z}{2}}$ and $l_0$ the number of elements in $U_{\frac{z}{2}+1,n,i}$, then $i_0 = k_0 + l_0$ is the number of elements in $U_{1,n}$. Suppose that the right part of the implication does not hold, so we have $k_0 < k$ and $l_0 < l$

\[
\begin{align*}
( & k_0 < k ) \quad \text{and} \quad ( l_0 < l ) \\
\iff ( k_0 \leq k-1 ) \quad \text{and} \quad ( l_0 \leq l-1 ) \\
\Rightarrow ( k_0 + l_0 ) \leq ( k + l - 1 - 1 ) \\
\iff i_0 \leq ( i - 1 ) \\
\iff i_0 < i \\
\end{align*}
\]

which is per definition $\neg u_{1,n,i}$ and thus contradicts to the left part of the implication. \qed

Using the above lemma we can split up the event sets and rewrite the definition of $u_{1,n,i}$ as:
\[ u_{1,n,i} \Rightarrow (u_{1,\frac{z}{2},k} \lor u_{\frac{z}{2}+1,n,i}) \lor k + l = i+1 \quad \text{for all } k \geq 0, l \geq 0 \]

s.t. \quad $k + l = i+1, i \in \{1, \ldots, n\}$

This is equivalent to:
\[ u_{1,n,i} \Rightarrow \bigwedge_{k,l} (u_{1,\frac{z}{2},k} \lor u_{\frac{z}{2}+1,n,i}) \lor k + l = i+1 \quad \text{for all } k \geq 0, l \geq 0 \]

s.t. \quad $k + l = i+1, i \in \{1, \ldots, n\}$

This implication is then formulated as a CNF formula:
\[ \bigwedge_{k,l} (\neg u_{1,n,i} \lor u_{1,\frac{z}{2},k} \lor u_{\frac{z}{2}+1,n,i}) \lor k + l = i+1 \quad \text{for all } k \geq 0, l \geq 0 \]

s.t. \quad $k + l = i+1$

Given the limitations we can find $(i+2)$ pairs of values for $k$ and $l$. Hence we need a total of $(i+2)$ clauses to represent this implication as a CNF formula. This procedure essentially expresses the variable $u_{1,n,i}$ in terms of lower level variables $u_{1,\frac{z}{2},k}$ and $u_{\frac{z}{2}+1,n,i}$.

We need $2(i+2)$ lower level variables to represent the clauses. We must note here that we do not always need all of these lower level variables, indeed the variables $u_{1,\frac{z}{2},k}$ and $u_{\frac{z}{2}+1,n,i}$, with $z > (\frac{z}{2} + 1)$ will trivially be false, since there can not be more than $(\frac{z}{2} + 1)$ indices in the sets $U_{1,\frac{z}{2}}$ or $U_{\frac{z}{2}+1,n,i}$. In general, a variable $u_{x,y,z}$ with $z > (y-x + 1)$ will always be trivially false. This brings the actual number of lower level to at most $2(\frac{z}{2} + 1)$.

Generally, we need $\min((n+2), 2(i+2))$ lower level variables. In the worst case, when $i$ equals $n$, this is $(n+2)$.

Whereas the original definition of $u_i$ used sets of size $n$, we are now left with sets of size $\frac{z}{2}$. We repeat this recursively until the sets are of size 1. The variables $u_{x,0,0}$ and $u_{x,x,1}$ then correspond to $\neg v_x$ and $v_x$ respectively.

The ‘left’ variables $u_{1,\frac{z}{2},k}$ with different $k$ will in their turn all use the same lower level variables $u_{1,\frac{z}{2},k'}$ and $u_{\frac{z}{2}+1,n,i'}$. This is similar for the ‘right’ variables. The second iteration will thus introduce $2(n/2 + 2)$ lower level variables. The total process will ultimately introduce the following number of lower level variables, for $2^k = n$:
\[
(n+2) + 2(n/2 + 2) + \cdots + 2^{k-1}(n/2^{k-1} + 2) = (n+2) + n + 4 + \cdots + n + 2n
\]

which equals
\[
(n+n+\cdots+n) + 2(2^0 + 2^1 + 2^2 + \cdots + 2^{k-1})
\]

with twice $k = \log_2 n$ terms
\[
= n\log n + \sum_{i=1}^{k} (2^i)
\]
\[
= n\log n + 2n - 2
\]
\[
= O(n\log n + n)
\]
\[
= O(n\log n)
\]
4.3.3 Translation of Total and perType Constraints

For an employee \( p \) and numbering \( N_i \), the total constraints impose restrictions on the minimum and maximum number of decision variables \( v_{p,j} \) (for which \( N_i \) is defined) that can be assigned true in a given personal schedule \( S_p \). Definition 11 to 13 can (assuming \( V \) contains \( n \) elements) be rewritten:

\[
\begin{align*}
V & = \{ v_{p,i} | i \in N_i \} \\
U_{\alpha \beta} & = \{ k | v_{p,k} \land k \in N_i \land (\alpha \leq k \leq \beta) \} \\
F_{\alpha \beta} & = \{ k | v_{p,k} \land k \in N_i \land (\alpha \leq k \leq \beta) \}
\end{align*}
\]

A \( \text{min}_\text{total} = \text{min}_t \) constraint can be translated to SAT by using the general procedure from the previous section to express:

\[
\begin{align*}
u_{\text{min}_t} & \Leftrightarrow |U_{1,n}| \geq \text{min}_t
\end{align*}
\]

A \( \text{max}_\text{total} = \text{max}_t \) constraint means at least \( n - \text{max}_t \) variables should be assigned false. Analogously, this can be expressed by translating

\[
\begin{align*}
f_{n - \text{max}_t} & \Leftrightarrow |F_{1,n}| \geq n - \text{max}_t
\end{align*}
\]

The translation of perType constraints is similar. Only difference is that the set of variables \( V \) is limited to those corresponding to a specific value \( j \).

\[
V = \{ v_{p,i} | i \in N_i \land N_i(i) = j \}
\]

5 APPLICATION: HARDNESS STUDY FOR NURSE ROSTERING PROBLEM INSTANCES

Empirical hardness can be understood as the complexity of a problem instance, when solved with a particular solution method, measured by some performance criteria. This hardness is a combination of the intrinsic hardness of the instance and the quality of the procedure on this specific instance. This implies that empirical hardness is inherently linked to the algorithm that is used. Empirical hardness models typically map a set of instance features onto a performance criterion such as the running time of an algorithm.

Instance features are often derived from expert knowledge. However, experts may be biased and good experts in the problem domain may not always be available. Therefore, by translating the problem instances under study to SAT, we examine the feasibility of a more general approach based on well known SAT-features (Nudelman et al., 2004).

For the hardness studies presented in this paper, we follow the methodology presented by (Leyton-Brown et al., 2006). Six steps are to be followed:

1. Identification of the problem instance distribution.
2. Selection of one or more algorithms and performance criteria.
3. Selection of a set of inexpensive, distribution independent features.
4. Sampling the instance distribution for generating a training set.
5. Elimination of redundant and uninformative features.
6. Learning models that map the feature space onto the performance criteria.

As hardness studies are not the scope of this paper, we refer to the aforementioned paper for more details on the method used and the technical report (Mes-selis et al., 2012) for specific and detailed information on the above steps in the light of the nurse rostering problem. We present the results of two performed experiments to demonstrate the applicability of the SAT translation schemes presented in this paper.

The first experiment focusses on the applicability of the SAT features proposed in (Nudelman et al., 2004). We designed a problem instance distribution consisting of “smaller” problem instances\(^2\).

The second experiment demonstrates the applicability of hardness models to real world like optimisation problems. Testbed were randomisations of the instances found in the First international nurse rostering competition (Haspeslagh et al., 2012).

5.1 Applicability of Hardness Studies using SAT Features for NRP

As stated before, a problem distribution of “smaller” instances is generated. This allows the use of IBM CPLEX to solve the instances. Besides an optimal

\(^2\)Smaller denotes a small planning horizon, a limited number of employees available and a limited number of shift to be covered.
Figure 1: Prediction of the optimal solution quality.

Figure 2: Prediction of the solution quality obtained by the metaheuristic.

The main contribution of this paper is the automated translation of nurse rostering problem instances into SAT problems instances. To support automated translation, a formal description of nurse rostering problem instances is required. We used numberings (Burke et al., 2001) for clearly and unambiguously describing instances (Haspeslagh et al., 2012). We defined eight generic constraint types allowing the representation of a large number of real world constraints found in literature. For each constraint type, we presented efficient translation schemes to SAT. The preprocessing step for both the SAT translation introduces $O(n)$ variables and $O(n)$ clauses. The SAT translation of the consecutive and between constraints generates $O(n)$ clauses. The SAT translation of counting constraints generates $O(n^2)$ clauses and $O(n \log n)$ variables. Summarized, all constraints that can be expressed as monotonic ascending numberings can be translated to SAT. The automated translation allows for an error safe representation of nurse rostering problem instances as instances of other problem domains. A manual translation can be very time consuming and complete equivalence between the original and the translated model may be hard to attain.

This approach allows studying problems from an other point of view. As shown in Section 5, the SAT translation scheme was successfully used to study the hardness of nurse rostering problem in-
stances. Although the aims of the algorithms for which the SAT feature set (Nudelman et al., 2004) was developed is very different from the aims of algorithms within the field of nurse rostering research (satisfiability versus optimality), the relevance of SAT features for the hardness analysis of nurse rostering problem instances was demonstrated. As we translate to SAT, information on the objective function of the original problem is lost. As a natural extension, the SAT translation scheme can be adapted to produce MAX-SAT instances thereby ‘incorporating information on the objective function’. In a first effort, MAX-SAT solvers (Argelich and Manyà, 2006) can be applied to study the solution quality obtained by those solvers. Another research direction is to design hybrid solvers. Current efforts first try to solve a partial problem with an exact solver (Burke and Curtois, 2011; Valouxis et al., 2012). The obtained solution is then optimised using for example a metaheuristic. One interesting research challenge is to study the opposite. SAT solvers are able to search the entire solution space. A metaheuristic search method only explores the solution space partially. In a sense, metaheuristics are designed for trying to escape local optimal in the solution space, e.g. a metaheuristic is used to incorporate diversification in the search process. By adding extra constraints, based on the solutions obtained by a metaheuristic search method, we can force the (MAX-)SAT solvers not to explore those parts of the solution space covered by the metaheuristic search and therefore try to intensify diversification.

REFERENCES


