Importance of Variables Semantic in CNF Encoding of Cardinality Constraints

Anbulagan and Alban Grastien
NICTA & The Australian National University
Canberra, Australia
{anbulagan|alban.grastien}@nicta.com.au

Abstract
In the satisfiability domain, it is well-known that a SAT algorithm may solve a problem instance easily and another instance hardly, whilst these two instances are equivalent CNF encodings of the original problem. Moreover, different algorithms may disagree on which encoding makes the problem easier to solve. In this paper, we focus on the CNF encoding of cardinality constraints, which states that exactly $k$ propositional variables in a given set are assigned to true. We demonstrate the importance of the semantics of the SAT variables in the encoding of this constraint. We implement several variants of the CNF encoding in which the close semantic variables are grouped. We then examine these new encodings on problems generated from diagnosis of discrete-event system.

Our results demonstrate that both stochastic and systematic SAT algorithms can now solve most of the problem instances, which were unreachable before (Grastien et al. 2007). These results also indicate that, on average cases, there is an encoding that suits well both SLS and DPLL algorithms.

Introduction
The fast growing research in propositional satisfiability (SAT) has a positive impact on solving an increasing number of practical applications, including diagnosis, planning, scheduling, hardware and software verification, among many others. Basically, an application problem will be encoded into a CNF formula, which will then be solved using a SAT solver. It has been shown in SAT planning (Ernst, Millstein, and Weld 1997) that the SAT encoding of a problem can have huge impact on the runtime. This paper focuses on CNF encodings of cardinality constraints, which state that a given number of propositional variables within a specified subset of the variables in the SAT problem is assigned to true. The constraint is defined on a set of variables, and can be encoded by several equivalent ways depending on the order in which the variables are integrated in the constraint. We show that this ordering has a huge impact on the runtime of both SLS and DPLL algorithms.

We examine the encoding of cardinality constraints in discrete-event system (DES) diagnosis problems, but this result can be generalized to other problems. DES diagnosis is the problem of determining whether the behavior of a system is normal or faulty according to the observations generated by this system. The use of SAT algorithms in better solving the DES diagnosis problems was first proposed in (Grastien et al. 2007), where the results demonstrated that SAT algorithms outperformed the traditional diagnosis algorithms. However, the SAT algorithms were still unable to solve about 30% of the SAT-encoded instances examined in that study (within 1200 seconds each), particularly the diagnosis problem under partially ordered observations. Therefore, in this paper we propose several variants of CNF encodings of cardinality constraints in which the close semantic variables are grouped. Experimental results indicate that both stochastic and systematic SAT algorithms can now solve most of the problem instances, which were unreachable before (Grastien et al. 2007). Another important finding is that, on average cases, there is an encoding that suits well the SAT algorithms.

CNF Encoding of Cardinality Constraints
The cardinality constraint in a SAT problem is the following: given a set $S$ of $n$ propositional variables, exactly $k$ variables of $S$ are assigned to true, where $k \leq n$.

The following set of naive rules can be used to encode the constraint into CNF form without auxiliary variables:

i) For any subset $\{v_1, \ldots, v_{k+1}\}$ of $k+1$ variables of $S$, specify that at least one variable must be assigned to false: $\neg v_1 \lor \cdots \lor \neg v_{k+1}$.

ii) For any subset $\{v_1, \ldots, v_{n-k+1}\}$ of $n-k+1$ variables of $S$, specify that at least one variable must be assigned to true: $v_1 \lor \cdots \lor v_{n-k+1}$.

However, it requires $n! \left(\frac{1}{(k+1)!} + \frac{1}{(k-1)!} \times n\right)$ clauses which makes it impractical for any $k > 1$. For the special case where $k = 1$, Marques-Silva and Lynce (2007) proposed an encoding that is better than the naive one.

Another encoding method introduced by Bailleux and Boufkhad (2003) and further studied by Sinz (2005) is the one based on a totalizer (also called circuit). The totalizer is a tree (see Figure 2) whose leaves are labeled with the variables of $S$. The nodes of the tree are labeled with auxiliary variables modeling a number; the constraints on the
totalizer ensure that this number equals the number of variables assigned to true in the leaves of this node subtree. The variables on the root are assigned to ensure the constraint.

To model an integer between 0 and a maximum value \( K \) (for instance \( K_i = k \)) at node \( i \) of the totalizer, the literature proposes the binary and the unary encodings.

**Binary Encoding** A number is modeled with the usual binary encoding as proposed in (Warners 1998). Let \( a_1, \ldots, a_n \) be the set of variables used to model the integer \( a \); the value of \( a \) is \( \sum_{j \in \{1, \ldots, n\}} (\text{val}(a_j) \times 2^{j-1}) \) where \( \text{val}(a_j) = 1 \) if \( a_j \) is set to true or \( \text{val}(a_j) = 0 \) otherwise. For instance, the assignment \( \{a_1 \rightarrow \text{true}, a_2 \rightarrow \text{false}, a_3 \rightarrow \text{true}, a_4 \rightarrow \text{true}\} \) of the four variables of a number \( a \) corresponds to value \( 1 + 0 + 4 + 8 = 13 \). This encoding requires \( \lfloor \log_2(K_a) \rfloor \) variables.

The constraint \( a + b = c \) requires \( \lfloor \log_2(K_a + K_b) \rfloor - 1 \) intermediate variables \( z_j \) representing the carry numbers in the addition, and is modeled by \( O(\log_2(K_a + K_b)) \) clauses representing the following Boolean constraints: \( z_j \leftrightarrow (a_j \land b_j) \lor (z_{j-1} \lor a_j \lor b_j) \) and \( c_j \leftrightarrow a_j \lor b_j \lor z_{j-1} \).

**Unary Encoding** This encoding was proposed by Bailleux and Boufkhad (2003). Let \( a_1, \ldots, a_n \) be the set of variables used to model the integer \( a \); \( \text{val}(a_i) = 1 \) if \( a_i \) is set to true. For instance, the assignment \( \{a_1 \rightarrow \text{true}, a_2 \rightarrow \text{false}, a_3 \rightarrow \text{true}, a_4 \rightarrow \text{false}\} \) of the four variables of a number \( a \) corresponds to value 3. This encoding requires \( K_a \) variables.

The addition \( a + b = c \) is modeled based on the following properties: \( (a \geq p) \land (b \geq q) \rightarrow (c \geq p + q) \) and \( (a < p + 1) \land (b < q + 1) \rightarrow (c < p + q + 1) \). The encoding of these properties requires \( O((K_a + K_b)^2) \) ternary clauses plus \( O((K_a + K_b)^2) \) binary clauses but no additional variables.

**Diagnosis by SAT**

This study takes place in the context of *discrete-event system* (DES) diagnosis (Lamperti and Zanella 2003). We briefly present the diagnosis problem and show how it is related to cardinality constraint in SAT encoding.

We consider a plant completely (including faulty behaviours) modeled by a DES, a finite automaton whose transitions are labeled by the events that occur when the transition is triggered. The DES is represented in a symbolic manner: a state is modeled by the assignment of state variables and the transitions are described by rules that indicate (i) what precondition the state must satisfy to enable the transition; (ii) what effect the transition has on the valuation of the state variables; and (iii) which events are associated with the transition. A sequence of states and transitions on the DES is called a trajectory; it models a behavior of the plant.

Some events are observable which means that an observation is emitted when they occur, for instance an alarm to the supervisor. The supervisor compares the model with the sequence of observations to retrieve what happened on the plant, such as which parts of the system are broken. Let \( \{f_1, \ldots, f_m\} \) be the set of \( m \) faulty events in the DES. The goal is to find a trajectory on the DES consistent with the observations and that minimizes the number of faults.

Grastien et al. (2007) proposed to solve the diagnosis problem using SAT algorithms. Basically, given a maximum length \( n \) for the trajectory that models what actually happened on the plant, propositional variables \( s^i \) (resp. \( r^i \)) are created to represent the valuation of state variable \( s \) (resp. the occurrence of event \( e \) and the triggering of rule \( r \)) at timestep \( i \). The model of the system and the observations implicitly define constraints \( \text{Mod} \) and \( \text{Obs} \) on the variables, which are encoded into CNF clauses.

Given \( k \in \mathbb{N} \), the cardinality constraint \( \text{Card}_k \) defines that exactly \( k \) variables of \( s^i \) are set to true, where \( (f_j, i) = f_j^i \). A satisfying assignment of \( \text{Mod} \cup \text{Obs} \cup \text{Card}_k \) models a trajectory with \( k \) faults; if the CNF formula is unsatisfiable, there is no solution with \( k \) faults. The diagnosis algorithm starts with 0 fault, and increments \( k \) by 1 until a solution is found. In this paper, we focus on the various encodings of the constraint.

**New Encodings of Cardinality Constraints**

We claim in this paper that the encoding of a totalizer requires two parameters. The first parameter corresponds to the node encoding. The second one, which to our best knowledge was not clearly identified in the literature, is the addition ordering: whether we should specify \( (a + b) + (c + d) = k \) or \( (a + c) + d + b = k \). We present several variants of encodings based on the combination of these parameters. We also propose two hybrid encodings.

**The Three Node Encodings**

The node encoding is the representation of the integer value associated with each node of the totalizer. For the purpose of this encoding, we use the binary (denoted \( B \)) and the unary (denoted \( U \)) encodings presented above. Moreover, we propose a new unary-based encoding of the property \( a + b = c \) (denoted \( A \)). In this encoding, each variable \( b_j \) of \( B \) is interpreted as a number that equals to 1 if the variable is set to true and 0 otherwise; the value of \( b \) is the sum of these numbers. Rather than computing directly \( c = a + b \), we compute \( c = ((a + b_1) + b_2) + \cdots + b_k \). Let \( c^i \), encoded by variables \( c^i_1, c^i_2, \ldots, \) denote the number corresponding to the addition of first \( i \) variables of \( a \) to \( b \). Thus, \( c^i_0 = a \). Since \( c^i + b_{i+1} = c^{i+1} \), then \( c^{i+1}_j \leftrightarrow c^i_j \lor (c^i_{j-1} \land b_{i+1}) \) where \( c^0_0 = \text{true} \) for all \( i \). This is described in Figure 1, where each box corresponds to a propositional variable. This figure represents the addition of a number \( a \) in \([0, \ldots, 5]\) with a number \( b \) in \([0, \ldots, 3]\). Since \( b_i \rightarrow b_{i-1} \), the property \( c^i_j = c^{i+1}_j \) stands for \( j > i \); thus, the variables represented in grey color in the figure can be removed. If \( K_a = K_b = K_c = K \), this encoding requires \( O(K^2 \times (K-1)/2) \) intermediate variables, and \( O(K^2) \) ternary clauses and \( O(K^2) \) binary clauses.

**The Seven Addition Orderings**

Given the set \( S \) of variables for which the constraint must be enforced, we now design a tree (the totalizer) where every variable of \( S \) is assigned to exactly one leaf. We propose seven addition orderings of which four of them are illustrated in Figure 2, by using a diagnosis problem with \( m = 3 \) faulty events \( \{d, e, f\} \), and \( n = 5 \) timesteps \( \{1, 2, 3, 4, 5\} \).
Figure 2: Examples of addition ordering in the totalizer for three events and five timesteps
A balanced tree is generated with leaves assigned randomly by CNF variables; this ordering is denoted \( R \) and sketched in Figure 2a. The other six addition orderings are defined as the combination of two variable groupings and three tree shapes.

The variable grouping tends to assign the variables in the tree in such a way as to put together the variables with close semantic. Variables in the diagnosis problem have two parameters: on which component the event occurred and at which timestep. We group the variables based on the same component (\( C \)) or the same timestep (\( T \), Figures 2b–d).

The tree shape indicates how the tree is built, given the variable grouping. The three shapes we considered are:

- \( F \) a balanced tree where the leaves are filled according to the grouping method chosen (Figure 2b);
- \( I \) incremental addition of subtrees, where each subtree corresponds to one group (Figure 2c);
- \( G \) a balanced tree of subtrees, where each subtree corresponds to one group (Figure 2d).

### The Two Hybrid Modelings

In a sub-problem and for a given SAT solver, an encoding \( Qw_{e_i} \) of the cardinality constraint may be easier to solve than another encoding \( Qw_{e_2} \), and conversely for another sub-problem. Thus, the diagnosis problem can be modeled by using several encodings of the cardinality constraint: \( Qw_{e_1} \cup \cdots \cup Qw_{e_k} \). In case the SAT solver is able to determine, thanks to its own heuristics, which \( Qw_{e_i} \) encoding is the most efficient, the SAT solver may reason only on this encoding; when this constraint is solved, the variables in the other encoding will be automatically fixed through unit propagation. Such an approach would potentially take benefit from both encodings. In this study, we present the hybrid modelings \( UCI—UTI \) (UCTI), and \( UCG—ACG \) (UACG) that vary only one parameter of the encoding. In our initial experiments, the other hybrid modelings showed the same performance as the ones studied here.

### The New Encodings

Table 1 lists the new encodings proposed in the study, where \( \alpha \) and \( \beta \) represent the node encoding and the addition ordering respectively. We defined 21 combinations of three node encodings (\( B, U \) and \( A \)) with seven addition orderings (\( R, CF, CI, CG, TF, TI \) and \( TG \)). We also defined two hybrid modelings (UCTI and UACG).

#### Empirical Validation

### Generating Various CNF-encoded Instances

We evaluated the 23 encodings presented in Table 1 on the hardest satisfiable and unsatisfiable CNF-encoded diagnosis problems examined in (Grastien et al. 2007):

- satisfiable problems: timed-hard-s, total-medium-s, total-hard-s, partial-medium-s, partial-hard-s;

For each problem, we generate 20 instances corresponding to a number of faults ranging from 1 to 20. The number of variables, on which the cardinality constraint is defined, is about 300 times the number of faults.

### Variable Numbering Based Encodings

The CNF-encoded problem represents each variable by an integer. We considered that a SAT solver may be influenced by these numbers, e.g., a SAT solver may branch on the variables with a small number first. Thus, we proposed two numberings of the variables: in the first case (denoted \( nT \)), the first numbers are given for timestep 0, then for timestep 1, etc. In the second case (denoted \( nC \)), the first numbers are given for the first component, then for the second component, etc.

### Hyper-Resolution in Modeling

We extend the encoding of the diagnosis problem with hyper-resolution rule. This extension, denoted \( +H \), generates additional binary clauses and appears when all the rules associated with a specific event have the same effect \( a \). Formally, we have the following clauses: \( \neg e \lor r_1 \lor \cdots \lor r_k \) and \( \forall i \in \{1, \ldots, k\} \), \( \neg r_i \lor a \), which implies \( \neg e \lor a \). This feature has a very little cost and increases the number of clauses by about 1%.

### SAT Solver Selection

From a number of state-of-the-art SAT solvers, we selected \( R+DDFW^+ \) (Ishtiaiwi et al. 2006) and \( MiniSAT v2 \) (Eén and Sörensson 2004) to represent stochastic local search (SLS) and DPLL-based systematic search, respectively. The idea behind choosing both solvers is to observe whether they behave asymmetrically with respect to the various encodings.

\( DDFW^+ \) is a clause weighting SLS, which adapts clause weights according to the degree of stagnation in the search. The \( R+DDFW^+ \) solver is an enhanced version of \( DDFW^+ \) by incorporating a resolution-based preprocessing, which adds resolvents of length \( \leq 3 \) into the original formula and then applies unit propagation to the formula. We selected

---

**Table 1:** List of the 23 encodings of cardinality constraint

<table>
<thead>
<tr>
<th>( \beta \rightarrow \alpha )</th>
<th>( R )</th>
<th>( C )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td>( U )</td>
<td>( AR )</td>
<td>( UC )</td>
</tr>
<tr>
<td>( UCI—UTI ) (UCTI)</td>
<td>( UCG—ACG ) (UACG)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure 1:** The \( A \) modeling of the addition for unary encoding

---

**Figure 2a:** A balanced tree where the leaves are filled according to the grouping method chosen. The three shapes we considered are: \( F \) a balanced tree where the leaves are filled according to the grouping method chosen; \( I \) incremental addition of subtrees, where each subtree corresponds to one group; \( G \) a balanced tree of subtrees, where each subtree corresponds to one group.

---

**Figure 2b:** The tree shape indicates how the tree is built, given the variable grouping. The three shapes we considered are: \( F \) a balanced tree where the leaves are filled according to the grouping method chosen; \( I \) incremental addition of subtrees, where each subtree corresponds to one group; \( G \) a balanced tree of subtrees, where each subtree corresponds to one group.

---

**Figure 2c:** A balanced tree is generated with leaves assigned randomly by CNF variables. The variable grouping tends to assign the variables in the tree in such a way as to put together the variables with close semantic.

---

**Figure 2d:** A balanced tree of subtrees, where each subtree corresponds to one group.
R+DDFW+ for its excellent performances shown in (Ish-taiwi et al. 2006), where it outperforms the best SLS solvers over a range of random and structured benchmark problems.

Clause learning DPLL solvers are remarkable in solving CNF-encoded industrial problems, which can be large in number of clauses and variables, and contain certain hard structures. In this category, MINISAT is well-known as one of the best solvers. Therefore, in our study, we use MINISAT v2 featuring variable elimination style simplification, as it outperforms the other versions in MINISAT family.

### Results and Analysis

The experiments were conducted on a cluster of 16 Intel Duo Core processors running at 2.4 GHz with 4 GB of RAM, as we had to run 31280 processes, which were allowed to use 1200 seconds each. In Table 2, we present the general total of solvers runtime per modeling heuristic on all satisfiable or all unsatisfiable problem instances. We then zoom in some selected results in Figure 3 and Tables 3–4. In Tables 2–4, the number of instances on which each solver failed is indicated in brackets before the total runtime. Each unsolvable instance contributes 1200 seconds to the total runtime.

#### From the Point of View of CNF Encodings

On Node Encodings Table 2 confirms the results presented in (Bailleux and Boughkald 2003) that, while the binary encoding (B*) creates fewer variables and clauses than the unary encodings (U* and A*), the latter are easier to solve. The runtime of R+DDFW+ increases by about 30% when using A* rather than U*, while MINISAT has almost the same performances on both encodings.

On Addition Orderings Both Table 2 and Figure 3 show that the random ordering (*R) is more difficult for MINISAT to solve. Figure 3a confirms that the runtime of MINISAT is improved at least by two orders of magnitude in some cases, when comparing UR encoding with the other U* ones. Our additional tests also show that MINISAT cannot solve most of the UR-encoded partial-medium-u problem instances when allowed five hours per instance. A further analysis of the results shows that the clauses learnt during the search in case of UR are approximately two times longer than for the other U* encodings.

Moreover, Table 4 shows that the variable ordering C (*C) makes the problems easier for MINISAT. Our explanation is that the solver benefits from the useful semantics of the intermediate variables. For instance, if a node of the totalizer is set to zero by branching, then this information will be propagated to the leaves of the corresponding subtree. If all the leaves relate to a specific event, the current hypothesis is that the event did not occur which will be easily negated if it is impossible. In the random totalizer, the branching implies that some events did not occur at specific times, while they may have happened at the previous timestep. We observe that the improvement achieved by the DPLL solver from the variable ordering *R to *CG is more important than the improvement achieved from the encoding of numbers B* to U* or A*. It should be noted that any

<table>
<thead>
<tr>
<th>SOLVE</th>
<th>Heuristic</th>
<th>n</th>
<th>H</th>
<th>R+H</th>
<th>H+H</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BR</strong></td>
<td>(23) 40 960</td>
<td>(15) 39 471</td>
<td>(15) 26 248</td>
<td>(15) 25 453</td>
<td></td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>(34) 39 982</td>
<td>(17) 37 881</td>
<td>(13) 23 805</td>
<td>(13) 23 805</td>
<td></td>
</tr>
<tr>
<td><strong>BTC</strong></td>
<td>(24) 41 601</td>
<td>(19) 40 638</td>
<td>(16) 25 073</td>
<td>(17) 26 660</td>
<td></td>
</tr>
<tr>
<td><strong>BDF</strong></td>
<td>(19) 41 238</td>
<td>(16) 38 538</td>
<td>(16) 27 570</td>
<td>(16) 25 943</td>
<td></td>
</tr>
<tr>
<td><strong>BTG</strong></td>
<td>(25) 39 174</td>
<td>(21) 37 643</td>
<td>(16) 26 355</td>
<td>(16) 25 574</td>
<td></td>
</tr>
<tr>
<td><strong>BTT</strong></td>
<td>(33) 53 574</td>
<td>(32) 54 793</td>
<td>(21) 35 902</td>
<td>(18) 35 024</td>
<td></td>
</tr>
<tr>
<td><strong>BTF</strong></td>
<td>(18) 37 732</td>
<td>(23) 32 782</td>
<td>(18) 27 072</td>
<td>(13) 24 751</td>
<td></td>
</tr>
</tbody>
</table>

| **TOTAL B** | (15) 28 735 | (18) 28 940 | (19) 29 720 | (19) 29 720 |

### Table 2: Summary of SAT solvers’ performance, after various modeling, on diagnosis problems. Each data represents the total runtime (in seconds) of 100 runs for satisfiable problems or of 140 runs for unsatisfiable problems. **Total α** represents the general total of solvers, per node encoding α and per variable numbering encoding (nT, nC, nT+H or nC+H). The best result based on α is represented in bold.
R+DDFW+ has a different behaviour. As with MINISAT, the $U^*$ and $A^*$ are preferable to $B^*$, with a slighter preference to $U^R$. The main difference is that, for a fixed node encoding, R+DDFW+ performs almost equally for most of the addition encodings, in particular for $^*R$. There is however an interesting counter example in $^*T$. The reason is the following: whenever the assignment of a leaf is changed, it generates a chain of changes to the root. All but one encodings lead to balanced or nearly balanced trees with a chain of at most $\lceil \log_2(20n) \rceil$ elements; the length can be up to $n + \lceil \log_2(20) \rceil$ in $^*Tj$ totalizers (compare $d^j$ in Figures 2b and 2c). Therefore, the chains in $^*Tj$ are longer, which makes the SAT problem harder to solve by the SLS solver (Wei and Selman 2002; Prestwich 2007).

**On Variable Numbering: Timestep vs Component**

Without considering hyper-resolution during modeling, timestep-based variable numbering approach is slightly better than the component-based variable numbering approach. But, the reverse phenomenon is shown when we run hyper-resolution during modeling. Variable numbering has little impact on the performances of the SAT solvers in general.

**On the Impact of Hyper-Resolution in Modeling**

The additional clauses generated by the hyper-resolution rule, about 1% of original problem, do not impact the performance of MINISAT, but they contribute to an important reduction of about 30% of the R+DDFW+ solver’s runtime. The results can be explained as following. The simple resolution preprocessor integrated in R+DDFW+ reduces the size of the original problem by 30% in average case, as the effect of running hyper-resolution when modeling. While the preprocessor integrated in MINISAT gives almost no reaction to the additional clauses, as shown by the results presented in Table 2. Figure 3c shows the performance of R+DDFW+ on total-hard-s problem, which are encoded using $UR$ and $UCG$, with or without hyper-resolution.

**On Hybrid Modeling**

We expected the hybrid modeling instances to be easier to solve than the best original one. However, our experiments show the opposite tendency, except for R+DDFW+ solver on $UCTI(nC+H)$ encoding.

---

**From the Point of View of SAT Solving**

**On Solvers’ Performance**

Table 3 presents the runtime of MINISAT and R+DDFW+ on various satisfiable problems. The results show that MINISAT has a better performance than R+DDFW+ on the timed-* and total-* problems, while R+DDFW+ is better on the partial-* problems. We observe that MINISAT’s runtime evolves more quickly than that of R+DDFW+. With certain encodings, R+DDFW+ is even able to solve the hardest satisfiable problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>MINISAT</th>
<th>R+DDFW+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$UCI(nT)$</td>
<td>$UTF(nT)$</td>
</tr>
<tr>
<td>timed-hard-s</td>
<td>105</td>
<td>95</td>
</tr>
<tr>
<td>total-medium-s</td>
<td>42</td>
<td>59</td>
</tr>
<tr>
<td>total-hard-s</td>
<td>139</td>
<td>121</td>
</tr>
<tr>
<td>partial-medium-s</td>
<td>2 792</td>
<td>(7) 10 142</td>
</tr>
<tr>
<td>partial-hard-s</td>
<td>(7) 10 902</td>
<td>(10) 13 601</td>
</tr>
</tbody>
</table>

**On Solving Unsatifiable Problems**

Table 4 presents MINISAT’s runtime on unsatisfiable problems based on some selected encodings. The results show the same evolution as for the satisfiable problems where $UC^*$ and $UT^*$ encoded instances are significantly easier than the $UR$ ones. With some encodings, MINISAT is now able to prove the unsatisfiability of most problems except the hardest partial-hard- $u$ problem instances. The results also show that the

---

**Figure 3:** SAT solvers’ runtime on selected diagnosis problems, after various $U^*$ encoding
main difference between $UC^*$ and $UT^*$ encodings for MINISAT appears in the partial-hard-s problem instances, which are difficult under the latter encoding.

On Solving the Hardest Problem Instances

We now present the results of solving the hardest instances of partial-hard-s and partial-hard-u problems under some $U^*$ encodings. We allocate 5 hours for solving each instance by a given solver. Table 5 presents solvers’ runtime (in seconds) under UCG on partial-hard-s for the highest values of $k$, which are the hardest satisfiable problem instances in the study. The results show that both solvers are able to solve these instances and the solver R+DDFW ++ persists when the hardness of problem instance increases.

Table 5: Solvers’ runtime on partial-hard-s problems for a given number of faults, using UCG encoding

<table>
<thead>
<tr>
<th>Instance</th>
<th>MINISAT</th>
<th>R+DDFW ++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$UCG(nC)$</td>
<td>$UCG(nC+H)$</td>
</tr>
<tr>
<td>18 faults</td>
<td>5 773</td>
<td>4 118</td>
</tr>
<tr>
<td>19 faults</td>
<td>7 790</td>
<td>8 078</td>
</tr>
<tr>
<td>20 faults</td>
<td>13 542</td>
<td>7 465</td>
</tr>
</tbody>
</table>

Table 6: Runtime of MINISAT on partial-hard-u problems for a given number of faults, using UR and UCG encodings

Table 6 compares MINISAT’s runtime (in seconds) under UR and UCG encodings on partial-hard-u problem for the highest values of $k$, which are the hardest unsatisfiable problem instances. MINISAT was given 50 hours for solving each problem instance. The results show that the instances of UR encoding are significantly harder for MINISAT than the ones of UCG, despite the fact that their sizes are almost the same. We observe that the hardness comes from the nature of the problem, where the length of the clauses learnt under UR encoding increases faster than that of UCG encoding, which usually differentiates the random from the structured SAT problems solving by clause learning SAT solvers. The results also confirm the importance of considering problem semantic in CNF encoding of cardinality constraints, particularly for the clause learning SAT solvers.

On Encodings versus Solvers

In order to show the benefit of the semantic-based encoding, we run more experiments on partial-medium and total-hard problems with $UR(nT)$ and $UCG(nT)$ encodings, by using DPLL (RSat (Pipatsrisawat and Darwiche 2007) and march_kks) and SLS

Table 7: Runtime of DPLL solvers on unsatisfiable problems in comparing $UR(nT)$ to $UCG(nT)$ encodings

- R+RSAPS 2 and R+adaptg2wsat0 3 solvers. We allocate one hour (3600 seconds) for solving each instance by a given solver. MINISAT and R+DDFW ++ were re-run with this time limit. Tables 7 and 8 present the results, where the number of unsolvable instances is indicated in brackets.

- The RSat solver was run without the SatElite simplifier. With the simplifier, the performance of RSat degrades on total-hard-s problem with $UR(nT)$ encoding, where it cannot solve one of the instances in the given time limit. The partial-medium-s and total-hard-s problems are very challenging for R+adaptg2wsat0 solver, which cannot solve any instance of the problems. In general, the results in average case show that DPLL solvers significantly benefit from the present of semantic-based encoding, which gives only a small impact to the SLS solvers.

On Problem Hardness by Increasing Number of Faults

Figure 4 shows that the difficulty of solving the problem instances between 1 fault and 20 faults increases drastically. Here, we study the difficulty of solving problem instances encoded by UCG, when the number of faults increases one

Figure 4: SAT solvers’ runtime when the hardness increases on a given problem, using $UCG(nC+H)$ encoding

3RSAP is available at www.satlib.org/ubsat/
3adaptg2wsat0 is available at www.laria.upicardie.fr/~cli/EnglishPage.html
by one until 100 faults. We present the results of running
MINISAT on timed-hard-s, total-hard-s and timed-hard-u
problems in Figure 4a. We also present the results of running
R+DDFW+ on timed-hard-s and total-hard-s problems
in Figure 4b. The results show that after 20 faults, the diffi-
culty of solving a problem instance increases linearly.

Summary
The new variants of encoding enable DPLL
and SLS algorithms to solve better most of the diagnosis
problems. The results demonstrate that the best encoding of
cardinality constraint is based on the unary representation
(U* and A*). SLS algorithms may use any variable ordering
in the totalizer as long as it does not generate a long chain
of variable dependencies, while the *CG or *CF orderings
should be used for DPLL algorithms. We propose to use the
UCG encoding as it suits well the SAT algorithms. More
generally we stressed, based on the solvers’ runtime, that
the addition ordering in cardinality constraints is important.

Application to Other Problem Domains
Semantic-based CNF encodings of cardinality constraints
have a significant impact on the time spent to solve diagnosis
problems. The results presented in this paper show several
orders of magnitude of improvement. A legitimate question
is whether equivalent results can be obtained for other prob-
lems that require cardinality constraints. We present several
such problems in the following.

The resolution of Pseudo-Boolean (PB) constraints with a
SAT approach is presented among many others in (Eén and
Sörensson 2006). In general, this problem does not provide
the semantics of the variables: the PB constraints input sim-
ply declares the variables by a character x followed by a
number. Still, it might be more efficient to group variables
that appear together in many constraints.

Bailleux and Boufkhad (2003) used the discrete tomog-
raphy problem to validate the unary encoding of numbers.
Here the semantics attached to each variable is known but
there is no other constraint apart from the cardinality con-
straints. Moreover, no two variables appear together twice
in the same constraint. Thus, using semantic-based encod-
ing of the constraint seems to have little impact.

The cardinality constraint can also appear in SAT-
planning (Böttner and Rintanen 2005) and SAT-scheduling
problems. As for diagnosis, the goal of the problems is to
find a minimal sequence of actions/events. We speculate that
our approach on these problems can have the same benefit
as the diagnosis problem studied in this paper.

Conclusion and Perspective
We presented several variants of semantic-based CNF en-
codings of cardinality constraints, based on the totalizer. We
then examined how the encoding of each node and the addi-
tion ordering impact the runtime of the DPLL and the SLS
SAT algorithms. The results demonstrate that the problem
is easier to solve when using a unary encoding. On the one
hand the performance of the enhanced DPLL algorithms
is boosted when the variables are adequately grouped; our
case study on diagnosis of DES shows more than two orders
of magnitude improvement when ordering the variables by

References
Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF En-
coding of Boolean Cardinality Constraints. In Proc. of the
9th CP, 108–122.
Böttner, M., and Rintanen, J. 2005. Satisfiability planning
with constraints on the number of actions. In Proc. of the
15th ICAPS, 292–299.
Eén, N., and Sörensson, N. 2004. An Extensible SAT-
Eén, N., and Sörensson, N. 2006. Translating Pseudo-
Boolean Constraints into SAT. Journal on Satisfiability,
SAT-Compilation of Planning Problems. In Proc. of the
15th IJCAI, 1169–1177.
Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E.
2007. Diagnosis of Discrete-Event Systems using Satisfia-
Ishtaiwi, A.; Thornton, J.; Anbulagan; Sattar, A.; and
Pham, D. N. 2006. Adaptive Clause Weight Redistribu-
Lamperti, G., and Zanella, M. 2003. Diagnosis of Active
CNF Encodings of Cardinality Constraints. In Proc. of the
13th CP, 483–497.
Pipatsrisawat, K., and Darwiche, A. 2007. Rsat 2.0: Sat
Reasoning Group, Computer Science Department, UCLA.
Prestwich, S. 2007. Variable Dependency in Local Search:
Prevention is Better than Cure. In Proc. of the 10th SAT,
volume LNCS 4501, 107–120.
Sinz, C. 2005. Towards an Optimal CNF Encoding of
Boolean Cardinality Constraints. In Proc. of the 11th CP,
827–831.
Warners, J. 1998. A Linear-time Transformation of Linear
Inequalities into Conjunctive Normal Form. Information
Wei, W., and Selman, B. 2002. Accelerating Random