Importance of Variables Semantic in CNF Encoding of Cardinality Constraints

Anbulagan
NICTA & The Australian National University
Canberra, Australia
anbulagan@nicta.com.au

Alban Grastien
NICTA & The Australian National University
Canberra, Australia
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, & 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 the results 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.

Copyright © 2009, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

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*: $\neg v_1 \vee \cdots \vee \neg v_{k+1}$.

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*: $v_1 \vee \cdots \vee v_{n-k+1}$.

However, it requires $\frac{n!}{(k+1)! \times (n-k-1)!} + \frac{n!}{(k-1)! \times (n-k+1)!}$ 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 $K = k$) at a node, 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 $a_1, \ldots, a_i$ be the set of variables used to model the integer a; the value of a is $\Sigma_{j\in\{1,\ldots,i\}}(val(a_j) \times 2^{j-1})$ where $val(a_j) = 1$ if a_j is set to *true* or $v(a_j) = 0$ otherwise. For instance, the assignement $\{a_1 \rightarrow true, a_2 \rightarrow false, a_3 \rightarrow true, a_4 \rightarrow true\}$ of the four variables of a number a corresponds to value $1 + 0 + 4 + 8 = 13$. This encoding requires $\lceil \log_2(K)\rceil$ variables.

The constraint $a + b = c$ requires $\lceil \log_2(K)\rceil - 1$ intermediate variables z_j representing the carry numbers in the addition, and is modeled by a $\log_2(K)$ clauses representing the following Boolean constraints: $z_j \leftrightarrow (a_j \wedge b_j) \vee (z_{j-1} \wedge (a_j \vee b_j))$ and $c_j \leftrightarrow a_j \oplus b_j \oplus z_{j-1}$.

Unary Encoding This encoding was proposed by Bailleux and Boufkhad (2003). Let $a_1, \ldots, a_i$ be the set of variables used to model the integer a; $val(a) \geq p$ iff a_p is set to *true*. For instance, the assignement $\{a_1 \rightarrow true, a_2 \rightarrow true, a_3 \rightarrow true, a_4 \rightarrow false\}$ of the four variables of a number a corresponds to value 3. This encoding requires K variables.

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

Diagnosis by SAT

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

We consider a plant modeled by a DES, which is basically a finite automaton where transitions are labeled by the events that occur when the transition is triggered. The automaton 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.

Grastien *et al.* (2007) proposed to solve this 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. e^i and r^i) are created to represent the valuation $s =$ *true* 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 define constraints on the variables, which are encoded into CNF clauses.

Let $\{f_1, \ldots, f_m\}$ be the set of m faulty events that can possibly happen on the system. The approach consists in finding a trajectory that minimizes the number k of occurrences of faulty events. This cardinality constraint denoted Que_k is expressed by ensuring that exactly k variables are set to *true* in $\mathcal{F} = \{f_1, \ldots, f_m\} \times \{1, \ldots, n\}$ where $\langle f_i, j\rangle = f_i^j$. 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 $c_1^i, c_2^i, \ldots$, denote the number corresponding to the addition of first i variables of b to a. Thus, $c^0 = a$. Since $c^i + b_{i+1} = c^{i+1}$, then $c_j^{i+1} \leftrightarrow c_j^i \vee (c_{j-1}^i \wedge b_{i+1})$ where $c_0^i =$ *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 $c_i^j = c_i^i$ stands for $j > i$; thus, the variables represented in grey color in the figure can be removed. If the maximum value K is the same for each number, this encoding requires $\frac{K\times(K-1)}{2}$ intermediate variables and K^2 ternary clauses plus K^2 binary clauses.

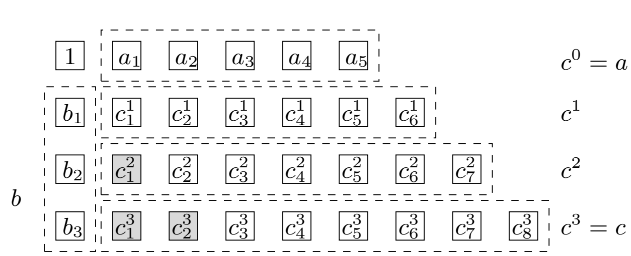
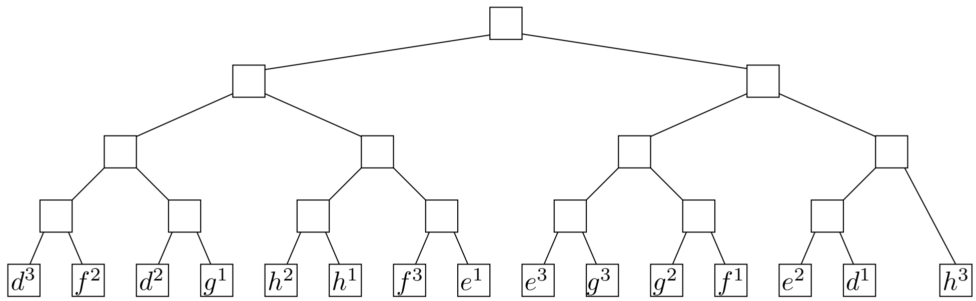
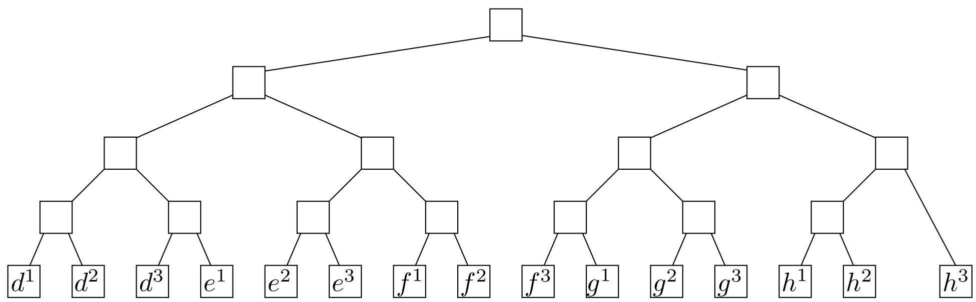


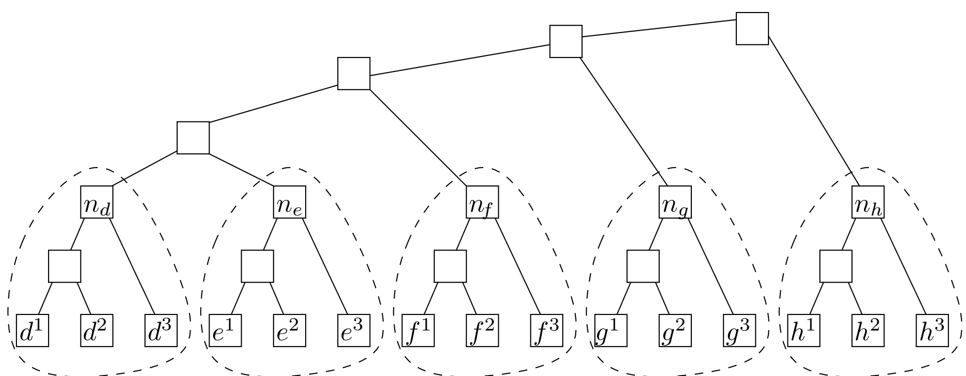
Figure 1: The *A* modeling of the addition for unary encoding



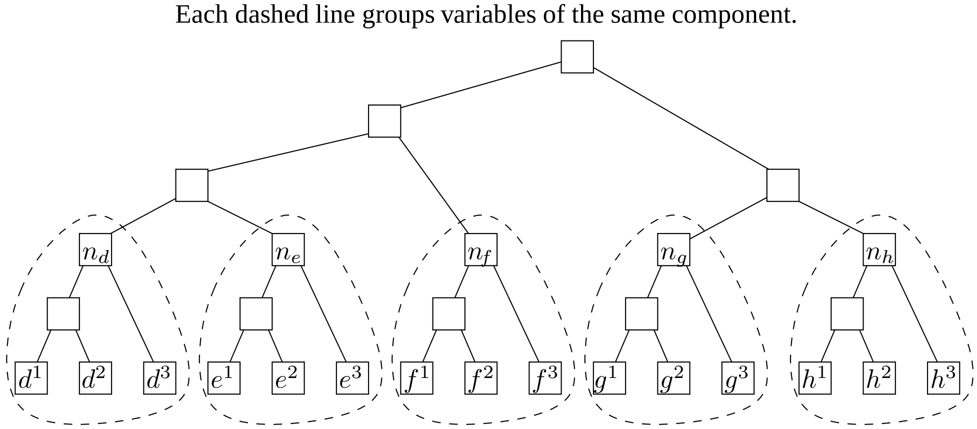
a. Addition ordering *R* (random on a balanced tree)



b. Ordering based on variable grouping *C* with tree shape *F* (*CF*)



c. Ordering based on variable grouping *C* with tree shape *I* (*CI*).
Each dashed line groups variables of the same component.



d. Ordering based on variable grouping *C* with tree shape *G* (*CG*).
Each dashed line groups variables of the same component.

Figure 2: Examples of addition ordering in the totalizer for five events and three timesteps

The Seven Addition Orderings

Given the set S of CNF 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 7 *addition orderings*. We illustrate 4 out of the 7 orderings in Figure 2, by using a diagnosis problem with $m = 5$ faulty events $\{d, e, f, g, h\}$, and $n = 3$ timesteps $\{1, 2, 3\}$.

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*, Figures 2b–d) or the same timestep (*T*).

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 Que_k^1 of the cardinality constraint may be easier to solve than another encoding Que_k^2, and conversely for another sub-problem. Thus, the diagnosis problem can be modeled by using several encodings of the cardinality constraint: $Que_k^1 \cup \cdots \cup Que_k^n$. In case the SAT solver is able to determine, thanks to its own heuristics, which Que_k^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 α and β 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*).

$\beta \rightarrow$	*R*	*C*			*T*		
$\alpha \downarrow$		*F*	*I*	*G*	*F*	*I*	*G*
B	*BR*	*BCF*	*BCI*	*BCG*	*BTF*	*BTI*	*BTG*
U	*UR*	*UCF*	*UCI*	*UCG*	*UTF*	*UTI*	*UTG*
A	*AR*	*ACF*	*ACI*	*ACG*	*ATF*	*ATI*	*ATG*
hybrid	*UCI—UTI* (*UCTI*)				*UCG—ACG* (*UACG*)		

Table 1: List of the 23 encodings of cardinality constraint

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*;
- unsatisfiable problems: *timed-medium-u*, *timed-hard-u*, *total-easy-u*, *total-medium-u*, *total-hard-u*, *partial-medium-u*, *partial-hard-u*.

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 file that encodes the SAT 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 $n\mathcal{T}$), the first numbers are given for timestep 0, then for timestep 1, etc. In the second case (denoted $n\mathcal{C}$), 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: $\neg e \vee r_1 \vee \cdots \vee r_k$ and $\forall i \in \{1, \ldots, k\}$, $\neg r_i \vee a$, which implies $\neg e \vee a$. 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$^+$ (Ishtaiwi *et al.* 2006) and MINISAT v2 (Eén & 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 algorithm, 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 ≤ 3 into the original formula and then applies unit propagation to the formula. We selected R+DDFW$^+$ for its excellent performances shown in (Ishtaiwi *et al.* 2006), where it outperforms the other best SLS solvers over a range of random and structured benchmark problems.

Clause learning DPLL solvers are reputable 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 & Boufkhad 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** encodings rather than *U** encodings, while MINISAT has almost the same performances on both encodings.

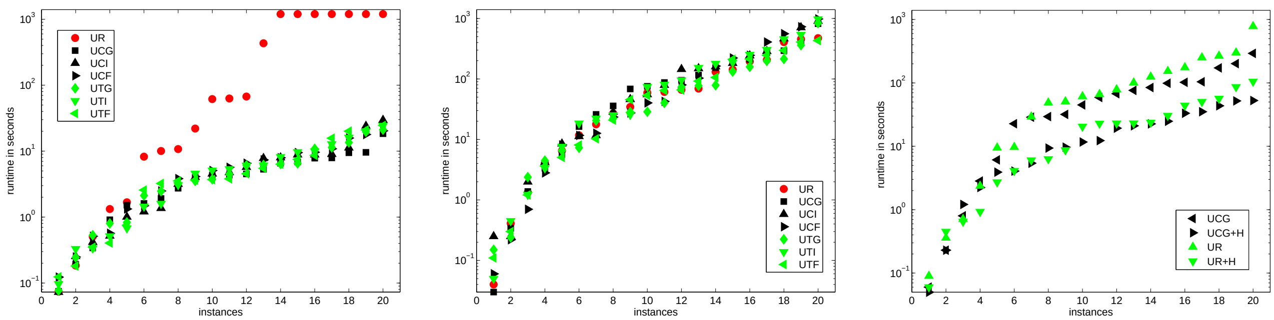
On Addition Orderings Both Table 2 and Figure 3 show that the random ordering (**R*) is more difficult for MINISAT to solve. Figure 3a confirms that the runtime of MINISAT is improved at least by two orders of magnitude in some cases, when comparing *UR* encoding with the other *U** ones. Our additional 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.

It also clearly appears that the variable ordering *C* (**C**) makes the problem easier for MINISAT (see Table 4). Our explanation is that the solver benefits from the fact that the intermediate variables in the encoding of cardinality constraints have useful semantics. Indeed in a diagnosis problem at some point of the search, it is often known that a given fault f did not occur between timestep i and j. Consider that fault e did not occur: the node n_e, in the totalizer of Figure 2d, can be automatically assigned to *false* by unit propagation; while with the random totalizer, nothing can be propagated. 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**.

On the other hand, as shown in Figure 3b, R+DDFW$^+$ performs almost equally for most of the encodings. There is however an interesting counter example in **TI*. Using tree shape *I* can generate long chains of dependencies. This

Solver	Heuristic	$n\mathcal{T}$	$n\mathcal{C}$	$n\mathcal{T}$+H	$n\mathcal{C}$+H
	BR	(23) 42 960	(17) 39 471	(15) 26 248	(15) 25 435
	BCG	(23) 43 446	(18) 37 881	(14) 25 982	**(13) 23 805**
	BCI	(22) 41 601	(19) 40 638	(16) 25 073	(17) 26 660
	BCF	(19) 41 238	(16) 38 538	(16) 27 570	(16) 25 943
	BTG	(20) 40 383	(19) 37 174	(16) 25 643	(16) 26 355
	BTI	(33) 53 574	(32) 54 793	(21) 35 902	(18) 35 024
	BTF	(18) 37 732	(21) 38 722	(18) 27 072	(13) 24 751
	Total B	(158) 300 934	(142) 287 217	(116) 193 490	**(108) 187 973**
R+DDFW$^+$ on 5 SAT problems with 100 instances	UR	(1) 14 311	(3) 14 144	7 643	**7 151**
	UCG	(1) 14 012	(2) 12 782	10 019	(1) 9 380
	UCI	(1) 14 165	(1) 14 139	(1) 9 859	(3) 11 016
	UCF	(2) 17 330	(2) 14 970	(1) 10 325	(1) 9 861
	UTG	(1) 11 208	11 050	(1) 9 916	(2) 10 245
	UTI	(1) 14 813	(2) 15 267	(2) 11 478	(1) 10 453
	UTF	(2) 12 993	(1) 10 567	8 151	(1) 9 728
	Total U	(9) 98 832	(11) 92 919	**(5) 67 391**	(9) 67 834
	AR	(3) 17 241	16 973	(1) 10 819	(2) 12 399
	ACG	(5) 17 858	(3) 16 844	(3) 14 003	12 257
	ACI	(4) 18 341	(2) 16 952	(4) 14 831	(2) 13 204
	ACF	(4) 21 349	(1) 16 022	(2) 13 660	(1) 13 150
	ATG	(1) 16 128	15 427	**10 810**	(3) 12 958
	ATI	(5) 24 930	(5) 23 903	(5) 15 492	(5) 16 691
	ATF	(1) 14 708	(2) 15 060	(3) 12 857	(1) 11 888
	Total A	(23) 130 555	(13) 121 181	(18) 92 472	**(14) 92 547**
	UCTI	(3) 19 267	(2) 15 518	(2) 12 578	10 151
	UACG	(6) 24 550	(6) 24 142	(5) 17 292	(4) 16 705
	BR	(51) 68 229	(52) 67 177	(50) 65 518	(50) 67 963
	BCG	(15) 25 312	(16) 24 319	(16) 24 884	(14) 24 039
	BCI	(18) 25 162	(16) 23 981	(18) 25 560	**(14) 22 948**
	BCF	(17) 25 445	(17) 25 976	(18) 26 059	(16) 25 574
	BTG	(24) 32 910	(23) 31 875	(24) 33 742	(23) 32 936
	BTI	(23) 31 460	(23) 32 128	(23) 32 155	(23) 31 865
	BTF	(24) 33 489	(25) 32 371	(26) 33 402	(24) 32 232
	Total B	(172) 242 007	(172) 237 827	(175) 241 320	**(164) 237 557**
MINISAT on 5 SAT problems with 100 instances	UR	(38) 51 706	(42) 55 159	(40) 53 515	(42) 54 407
	UCG	(8) 14 421	(7) 14 287	(8) 14 482	(8) 14 534
	UCI	**(7) 13 980**	(9) 15 002	(9) 15 620	(8) 15 303
	UCF	(9) 16 186	(9) 15 534	(8) 15 228	(7) 15 361
	UTG	(14) 22 337	(12) 20 459	(13) 20 113	(13) 20 013
	UTI	(17) 24 595	(15) 24 507	(15) 24 163	(15) 23 228
	UTF	(15) 22 791	(17) 24 679	(17) 24 018	(15) 24 818
	Total U	**(108) 166 016**	(111) 169 627	(110) 167 139	(108) 167 664
	AR	(40) 52 215	(42) 53 683	(36) 48 631	(40) 52 262
	ACG	(9) 16 209	(9) 15 874	(9) 16 037	(9) 15 824
	ACI	(11) 16 624	**(8) 15 588**	(10) 16 370	(10) 15 814
	ACF	(9) 17 098	(10) 16 989	(10) 17 561	(10) 17 943
	ATG	(17) 25 235	(13) 24 188	(17) 24 247	(14) 21 654
	ATI	(16) 23 522	(15) 24 393	(16) 23 931	(17) 24 891
	ATF	(16) 23 962	(16) 23 999	(17) 24 972	(18) 24 407
	Total A	(118) 174 865	**(113) 174 714**	(115) 171 749	(118) 172 795
	UCTI	(11) 18 835	(11) 17 239	(10) 17 474	(9) 17 495
	UACG	(8) 17 143	(10) 16 454	(9) 17 499	(8) 16 559
	BR	(66) 87 739	(67) 87 505	(65) 86 273	(65) 86 590
	BCG	(15) 23 853	(14) 23 828	(13) 23 411	(13) 23 283
	BCI	(13) 21 615	(13) 21 987	(14) 23 882	(15) 24 023
	BCF	(15) 24 867	(14) 23 638	(14) 24 889	**(12) 23 492**
	BTG	(26) 34 367	(25) 33 285	(26) 33 947	(26) 34 635
	BTI	(26) 34 257	(23) 32 920	(26) 33 740	(23) 33 034
	BTF	(26) 34 458	(27) 35 374	(27) 35 979	(26) 34 910
	Total B	(187) 261 156	(183) 258 537	(185) 262 121	**(180) 259 967**
MINISAT on 7 UNSAT problems with 140 instances	UR	(60) 79 666	(59) 79 510	(59) 79 204	(59) 79 900
	UCG	(8) 13 234	(8) 13 698	(7) 13 693	**(6) 13 331**
	UCI	(8) 14 818	(7) 13 376	(7) 14 551	(8) 13 844
	UCF	(7) 14 401	(9) 14 349	(7) 14 960	(9) 15 086
	UTG	(16) 22 859	(16) 23 406	(17) 24 433	(16) 23 029
	UTI	(15) 24 327	(17) 24 731	(17) 24 650	(15) 23 238
	UTF	(16) 25 234	(19) 26 699	(19) 27 011	(17) 25 436
	Total U	(130) 194 539	(135) 195 769	(133) 198 502	**(130) 193 864**
	AR	(62) 82 713	(64) 82 574	(62) 82 885	(60) 81 965
	ACG	(9) 15 358	(8) 15 200	(8) 15 574	**(8) 14 356**
	ACI	(8) 15 134	(9) 15 479	(8) 15 729	(8) 15 526
	ACF	(9) 17 206	(8) 15 786	(9) 16 635	(9) 16 508
	ATG	(19) 27 189	(19) 26 035	(17) 26 177	(18) 28 117
	ATF	(18) 26 578	(17) 25 343	(18) 25 089	(17) 24 842
	ATF	(18) 27 965	(19) 27 088	(18) 26 738	(20) 27 913
	Total A	(143) 212 143	(144) 207 505	**(140) 208 827**	(140) 209 227
	UCTI	(10) 19 070	(9) 15 788	(9) 16 171	(10) 18 306
	UACG	(7) 15 527	(8) 14 368	(7) 14 651	(9) 15 638

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 runtime, per node encoding α and per variable numbering encoding ($n\mathcal{T}$, $n\mathcal{C}$, $n\mathcal{T}$+H or $n\mathcal{C}$+H). The best result based on α is represented in bold.



a. MINISAT on *total-hard-s* after $U*$ ($n\mathcal{C}$+H) b. R+DDFW$^+$ on *partial-medium-s* after $U*$ ($n\mathcal{T}$) c. R+DDFW$^+$ on *total-hard-s* after $U*$ ($n\mathcal{C}$)

Figure 3: SAT solvers' runtime on selected diagnosis problems, after various $U*$ encoding

is not the case for **CI* as the number of components is only 20: the size of the chain is 20. However, for **TI*, the length of the chain is the number of timesteps, which reaches up to 300 (20 faults $\times$ $\sim$8 observations/fault $\times$2 timesteps/observation) in the worst cases of the instances (see Figure 2c). This corroborates Wei and Selman (2002), and Prestwich (2007) conjectures that long chains make the SAT problem hard for SLS.

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, in general our experiments show the opposite tendency, except for R+DDFW$^+$ solver on *UCTI*($n\mathcal{C}$+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.

Problem	MINISAT		R+DDFW$^+$	
	UCI($n\mathcal{T}$)	*UTF*($n\mathcal{T}$+H)	*UCI*($n\mathcal{T}$)	*UTF*($n\mathcal{T}$+H)
timed-hard-s	105	95	1 117	178
total-medium-s	42	59	1 253	881
total-hard-s	139	121	2 868	581
partial-medium-s	2 792	(7) 10 142	3 500	2 991
partial-hard-s	(7) 10 902	(10) 13 601	(1) 5 427	3 520

Table 3: Solvers' runtime comparison when problem hardness increases

Problem	*UR*($n\mathcal{T}$)	*UCG*($n\mathcal{T}$)	*UCF*($n\mathcal{T}$)	*UTG*($n\mathcal{T}$)	*UTF*($n\mathcal{T}$)
timed-medium-u	(2) 3 990	30	30	34	36
timed-hard-u	(7) 10 075	100	85	115	149
total-easy-u	(4) 5 891	18	22	24	22
total-medium-u	(4) 6 542	50	51	57	59
total-hard-u	(11) 14 155	117	133	172	185
partial-medium-u	(16) 19 354	2 019	2 086	(7) 10 465	(7) 12 269
partial-hard-u	(16) 19 659	(8) 10 900	(7) 11 994	(9) 11 992	(9) 12 514

Table 4: MINISAT runtime on unsatisfiable problem, based on some $U*$ encodings

On Solving Unsatisfiable 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 main difference between *UC** and *UT** encodings for MINISAT appears in the *partial-medium-u* 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.

Instance	MINISAT		R+DDFW$^+$	
	UCG($n\mathcal{C}$)	*UCG*($n\mathcal{C}$+H)	*UCG*($n\mathcal{C}$)	*UCG*($n\mathcal{C}$+H)
18 faults	5 773	4 118	1 439	1 799
19 faults	7 790	8 078	984	690
20 faults	13 542	7 465	1 936	819

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

Instance	*UR*($n\mathcal{C}$)		*UCG*($n\mathcal{C}$)	
	#Vars/#Cls	Time	#Vars/#Cls	Time
13 faults	164326/803514	$>$180 000	164354/803598	918
14 faults	180345/890920	$>$180 000	180351/890938	4 201
15 faults	178170/889037	$>$180 000	178194/889109	1 625
16 faults	209887/1057748	$>$180 000	209889/1057754	1 706
17 faults	220554/1122352	$>$180 000	220554/1122352	3 352
18 faults	235147/1208987	$>$180 000	235153/1209005	2 686
19 faults	254060/1319555	$>$180 000	254086/1319633	4 574
20 faults	265910/1394987	$>$180 000	265949/1395104	4 369

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 differenciates 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.

Solver	partial-medium-s		total-hard-s	
	UR($n\mathcal{T}$)	*UCG*($n\mathcal{T}$)	*UR*($n\mathcal{T}$)	*UCG*($n\mathcal{T}$)
MINISAT	(12) 46 034	3 909	(6) 29 561	90
RSat	(5) 21 361	5 996	1 454	184
march_ks	(18) 64 817	(10) 43 160	(15) 54 541	9 118
R+DDFW$^+$	3 022	5 336	1 914	2 072
R+RSAPS	(15) 54 529	(16) 57 606	(15) 54 029	(14) 50 860

Table 7: Runtime of DPLL and SLS solvers on satisfiable problems in comparing *UR*($n\mathcal{T}$) to *UCG*($n\mathcal{T}$) encodings

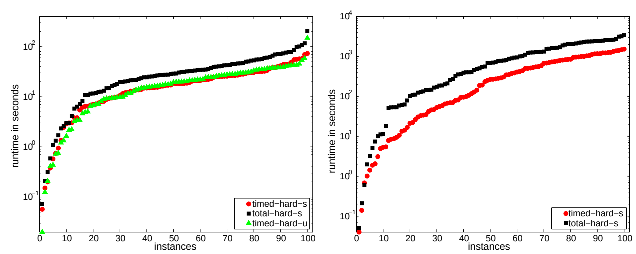
On Encodings versus Solvers In order to show the benefit of the semantic-based encoding, we run more experiments

Solver	partial-medium-u		total-hard-u	
	UR($n\mathcal{T}$)	*UCG*($n\mathcal{T}$)	*UR*($n\mathcal{T}$)	*UCG*($n\mathcal{T}$)
MINISAT	(16) 57 939	2 608	(9) 33 354	131
RSat	(15) 57 490	6 781	(8) 33 286	207
march_ks	(17) 61 305	(8) 48 402	(14) 51 131	8 372

Table 8: Runtime of DPLL solvers on unsatisfiable problems in comparing *UR*($n\mathcal{T}$) to *UCG*($n\mathcal{T}$) encodings

on *partial-medium* and *total-hard* problems with *UR*($n\mathcal{T}$) and *UCG*($n\mathcal{T}$) encodings, by using DPLL (RSat (Pipatsrisawat & Darwiche 2007) and march_ks[1]) and SLS (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*($n\mathcal{T}$) 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.



a. MINISAT's performance b. R+DDFW$^+$'s performance

Figure 4: SAT solvers' runtime when the hardness increases on a given problem, using *UCG*($n\mathcal{C}$+H) encoding

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 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 difficulty of solving a problem instance increases linearly.

[1]Available from http://www.st.ewi.tudelft.nl/sat/download.php

[2]RSAPS is part of UBCSAT 1.1, which is available from http://www.satlib.org/ubcsat/

[3]adaptg2wsat0 is available from http://www.laria.u-picardie.fr/∼cli/EnglishPage.html

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 problems 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 & Sörensson 2006). In general, this problem does not provide the semantics of the variables: the PB constraints input simply 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 & Boufkhad (2003) used the discrete tomography 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 constraints. Moreover, no two variables appear together twice in the same constraint. Thus, using semantic-based encoding of the constraint seems to have little impact.

The cardinality constraint can also appear in SAT-planning (Büttner & 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 encodings of cardinality constraints, based on the totalizer. We then examined how the encoding of each node and the addition ordering impact the runtime of the DPLL and the SLS SAT algorithms. The results demonstrate that the problem is easier to solve when using an 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 discrete-event systems shows more than two orders of magnitude improvement when ordering the variables by component compared to a random ordering. On the other hand the SLS algorithm runtime is reduced by ensuring a balanced tree while the order of the variable has no impact.

The encodings proposed in this study can be applied to other domains' problems that contain cardinality constraints, such as in classical planning and circuit verification problems. They can also be extended to more general arithmetic constraints. In general, our results emphasize that encoding a problem is as critical as solving the problem.

Finally, this study can also be a complementary to the study realized by Marques-Silva and Lynce (2007) in terms of using semantical knowledge of a problem for better choosing decision variables in SAT solving.

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Proc. of the 9th CP*, 108–122.

Büttner, 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-solver. In *Proc. of 6th SAT*, volume LNCS 2919, 502–518.

Eén, N., and Sörensson, N. 2006. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)* 2:1–26.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic 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 Satisfiability Algorithms. In *Proc. of the 22nd AAAI*, 305–310.

Ishtaiwi, A.; Thornton, J.; Anbulagan; Sattar, A.; and Pham, D. N. 2006. Adaptive Clause Weight Redistribution. In *Proc. of the 12th CP*, 229–243.

Lamperti, G., and Zanella, M. 2003. *Diagnosis of Active Systems*. Kluwer Academic Publishers.

Marques-Silva, J., and Lynce, I. 2007. Towards Robust CNF Encodings of Cardinality Constraints. In *Proc. of the 13th CP*, 483–497.

Pipatsrisawat, K., and Darwiche, A. 2007. Rsat 2.0: Sat solver description. Technical Report D–153, Automated 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 Processing Letters* 68:63–69.

Wei, W., and Selman, B. 2002. Accelerating Random Walks. In *Proc. of the 8th CP*, 216–232.