To Encode or to Propagate? The Best Choice for Each Constraint in SAT

Ignasi Abío[1], Robert Nieuwenhuis[2], Albert Oliveras[2], Enric Rodríguez-Carbonell[2], Peter J. Stuckey[3]

[1] Theoretical Computer Science, TU Dresden, Germany
[2] Technical University of Catalonia, Barcelona
[3] National ICT Australia and the University of Melbourne

Abstract. Sophisticated compact SAT encodings exist for many types of constraints. Alternatively, for instances with many (or large) constraints, the SAT solver can also be extended with built-in propagators (the SAT Modulo Theories approach, SMT). For example, given a cardinality constraint $x_1 + \ldots + x_n \leq k$, as soon as k variables become true, such a propagator can set the remaining variables to false, generating a so-called explanation clause of the form $x_1 \wedge \ldots \wedge x_k \rightarrow \overline{x_i}$. But certain "bottle-neck" constraints end up generating an exponential number of explanations, equivalent to a naive SAT encoding, much worse than using a compact encoding with auxiliary variables from the beginning. Therefore, Abío and Stuckey proposed starting off with a full SMT approach and *partially* encoding, on the fly, only certain "active" parts of constraints. Here we build upon their work. Equipping our solvers with some additional bookkeeping to monitor constraint activity has allowed us to shed light on the effectiveness of SMT: many constraints generate very few, or few different, explanations. We also give strong experimental evidence showing that it is typically unnecessary to consider partial encodings: it is competitive to encode the few really active constraints entirely. This makes the approach amenable to any kind of constraint, not just the ones for which *partial* encodings are known.

1 Introduction

The "SAT revolution" [Var09] has made SAT solvers a very appealing tool for solving constraint satisfaction and optimization problems. Apart from their efficiency, SAT tools are push-button technology, with a single fully automatic variable selection heuristic. For many types of constraints, sophisticated compact SAT encodings exist. Such encodings usually introduce auxiliary variables, which allows one to obtain succinct formulations. Auxiliary variables frequently also have a positive impact on the size and reusability of the *learned* clauses (lemmas), and, in combination with the possibility of deciding (splitting) on them, on the quality of the search.

Building in constraints: SAT Modulo Theories (SMT). On problem instances with many (or very large) constraints, where encodings lead to huge numbers of clauses and variables, it may be preferable to follow an alternative approach: in SMT [NOT06,BHvMW09], the SAT solver is extended with a built-in

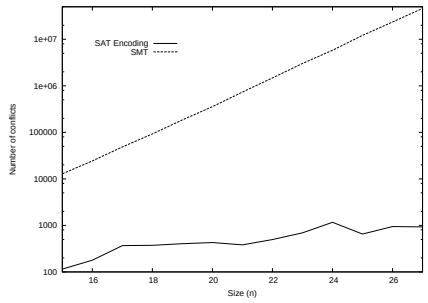
propagator for each constraint, making it amenable to sophisticated constraint-specific reasoning (as in Constraint Programming). For example, given a cardinality constraint $x_1 + \ldots + x_n \leq k$, as soon as k of its variables become true, such a propagator can set any other variable x_j to false. If at some later point this propagated literal $\overline{x_j}$ takes part in a conflict, a so-called *explanation* clause of the form $x_{i_1} \wedge \ldots \wedge x_{i_k} \rightarrow \overline{x_j}$ is used, thus fully integrating such propagators in the SAT solver's conflict analysis and backjumping mechanisms. As usual in SMT, here we consider that such explanations are (i) only produced when needed during conflict analysis and (ii) are not learned (only the resulting lemma is).

The remarkable effectiveness of SMT. SMT is remarkably effective. The intuitive reason is that, while searching for a solution for a given problem instance, some constraints only block the current solution candidate very few times, and moreover they do this almost always in the same way. In this paper we shed some more light on this intuitive idea. We perform experiments with a number of notions of *constraint activity* in this sense, that is, the (recent or total) number of (different or all) explanations that each constraint generates. Indeed, as we will see: A) many constraints generate very few, or few different, explanations, and B) generating only these explanations can be much more effective than dealing with a full encoding of the constraint.

The dark side of SMT. Frequently, there are also certain "bottle-neck" constraints that end up generating an exponential number of explanations, equivalent to a naive SAT encoding. A theoretical but illustrative example is:

Lemma 1. *An SMT solver will generate an exponential number of explanations when proving the unsatisfiability of the input problem consisting of only the two cardinality constraints* $x_1 + \ldots + x_n \leq n/2$ *and* $x_1 + \ldots + x_n > n/2$.

This lemma holds because any SMT solver, when proving unsatisfiability, generates a propositionally unsatisfiable set of clauses (the input ones plus the lemmas), and if a single one of the all $\binom{n}{k+1} + \binom{n}{n-k}$ explanations (where $k = n/2$) has not been generated, say, the explanation $\overline{x_1} \vee \ldots \vee \overline{x_{k+1}}$, then the assignment that sets $x_1, \ldots, x_{k+1}$ to true and the remaining $n - k - 1$ variables to false is a model. Such situations indeed happen in practice: for some constraints SMT ends up generating a full or close to full encoding, which is moreover a very naive exponential one, with no auxiliary variables. If a polynomial-size encoding for such a constraint exists (possibly with auxiliary variables), using it right from the beginning is a much better alternative. This is shown in the following figure:



It gives the number of conflicts needed to prove unsatisfiability of the previous example, varying n, with our Barcelogic solver in SMT mode and with a SAT encoding based on Cardinality Networks. SMT exhibits exponential behavior (note the logarithmic scale). The encoding-based version scales up much better; in fact, a polynomial-size refutation for it exists, although it is not clear from the figure whether the solver always finds it or not.

Getting the best of both. In their *conflict-directed lazy decomposition (LD)* approach [AS12], Abío and Stuckey proposed starting off the solver using an SMT approach for all constraints of the problem instance, and *partially* encoding (or *decomposing*), on the fly, only the "active" *parts of* some constraints. The decision of when and which auxiliary variables to introduce during the solving process is taken with a particular concrete encoding in mind: if, according to the explanations that are being generated, it is observed that an auxiliary variable of the encoding and its corresponding part of the encoding would have been "active", then it is added to the formula, together with all of the involved clauses of the encoding. In this way, fully active constraints end up being completely encoded using the compact encoding with auxiliary variables, and less active ones are handled by SMT. In [AS12] it is shown that this can be done for the Cardinality/Sorting Network encoding of cardinality constraints, and, although in a complicated way, for BDD-encodings of pseudo-Boolean constraints, performing essentially always at least as well as the best of SMT and encoding.

Going beyond. A shortcoming of [AS12] is that it is highly dependent on the constraint to be dealt with and the chosen encoding, making it unlikely to be applicable to other more complex constraints, and in any case equipping the theory solver with the required features is a highly non-trivial task. Here we propose another technique that is much simpler. It does not depend on the concrete constraint under consideration and can in fact be applied to any class of constraints that can be either encoded or built in. As mentioned previously, we have devised and analyzed bookkeeping methods for different notions of constraint activity that are cheap enough not to slow down solving appreciably. As a result, here we show, giving strong experimental evidence, that it is typically unnecessary to consider partial encodings: the few really active constraints can usually be encoded –on the fly– entirely. This makes the approach amenable to any kind of constraint, not just the ones for which *partial* encodings are known. Results on problems containing cardinality and pseudo-Boolean constraints are comparable, and frequently outperform all three of its competitors: SMT, encoding, and the partial lazy decomposition method of [AS12].

2 SAT and SAT Encoding

Let $\mathcal{X} = \{x_1, x_2, \ldots x_n\}$ be a finite set of propositional *variables*. If $x \in \mathcal{X}$ then x and $\overline{x}$ are *literals*. The *negation* of a literal l, written $\overline{l}$, denotes $\overline{x}$ if l is x, and x if l is $\overline{x}$. A *clause* is a disjunction of literals $l_1 \vee \ldots \vee l_n$. A *(CNF) formula* is a conjunction of clauses.

An *assignment* A is a set of literals such that $\{x, \overline{x}\} \subseteq A$ for no x. A literal l is *true* in A if $l \in A$, is *false* if $\bar{l} \in A$, and is *undefined* otherwise. A clause C is true in A if at least one of its literals is true in A. A formula F is true in A if all its clauses are true in A, and then A is a *model* of F. The *satisfiability (SAT) problem* consists in, given a formula F, to decide if it has a model. Systems that decide the SAT problem are called *SAT solvers*.

A function $C : \{0,1\}^n \to \{0,1\}$ is called a *constraint*. Given a constraint C, a *(SAT) encoding* for it is a formula F (possibly with auxiliary variables) that is equisatisfiable. An important class of constraints are *cardinality constraints*, which state that at most (or at least, or exactly) k out of n variables can be true. Common encodings for it are based on networks of adders [BB03,uR05,AG09,Sin05], or Sorting Networks [ES06,CZI10,ANORC09,ANORC11]. Cardinality constraints are generalized by *pseudo-Boolean constraints*, of the form $a_1x_1 + \cdots + a_nx_n \# k$, where the a_i and k are integer coefficients, and $\#$ belongs to $\{\leq, \geq, =\}$. Again, several encodings exist, based on performing arithmetic [War98,BBR09,ES06] or computing BDD's [BBR06,ES06,ANO$^+$12]. Most convenient encodings are the ones for which the SAT solver's unit propagation mechanism preserves domain-consistency.

3 To Encode or Not to Encode?

In this section we will discuss situations where encoding a constraint is better than using a propagator for it or vice versa, and how to detect them. The reasoning will consist of both theoretical insights and experimental evaluation. For the latter, 5 benchmarks suites will be used, in which all benchmarks solvable in less than 5 seconds by both methods have been removed.

1.-MSU4: 5729 problems generated in the execution of the *msu4* algorithm [MSP08] for Max-SAT. Each benchmark contains very few $\leqslant$-cardinality constraints.
2.-Discrete-event system diagnosis: 4526 discrete-event system (DES) diagnosis [AG09] problems. Each benchmark contains a single very large $\leqslant$-cardinality constraint.
3.-Tomography: 2021 tomography problems introduced in [BB03]. Each problem contains many $=$-cardinality constraints.
4.-PB evaluation: 669 benchmarks from the pseudo-Boolean Competition[1] 2011 (category DEC-SMALLINT-LIN), with multiple cardinality and pseudo-Boolean constraints.
5.-RCPSP: 577 benchmarks coming from the PSP-Lib[2]. These are scheduling problems with a fixed makespan. Several pseudo-Boolean constraints are present.

To start with, let us experimentally confirm that SMT and encoding-based methods are complementary, and so a hybrid method getting the best of both is worth pursuing. For this purpose, we implemented an SMT-based system

[1] `http://www.cril.univ-artois.fr/PB11/`
[2] `http://webserver.wi.tum.de/psplib`

and encodings into SAT. For cardinality constraints, we used the Cardinality Networks encoding of [ANORC11], whereas for pseudo-Boolean constraints, the BDD-based one of [ANO+12]. The reason for this choice is that, according to the experimental results of [ANORC11,ANO+12], these two encodings are the globally most robust ones in practice. However, any other choice would have been possible, since the approach we will present is encoding-independent. A time limit of 600 seconds was set per benchmark and, in order to have a fair comparison, in both systems the same underlying SAT solver was used (Barcelogic). Results can be seen in Table 1, where one can observe that the encoding performs very well in the MSU4 and DES suite, and is significatively worse in the other three.[3]

			Geometric mean	
Benchmark suite	**Encoding $\geq$ 1.5x faster**	**SMT $\geq$ 1.5x faster**	**Encoding**	**SMT**
MSU4	39.37%	15.39%	1.71	23.53
DES	92.06%	0.28%	2.3	56.02
Tomography	5.93%	86.49%	46.95	4.37
PB evaluation	7.02%	43.49%	25.53	3.79
RCPSP	0.69%	46.62%	106.65	5.8

Table 1: Comparison between encoding and SMT. Table on the left indicate the percentage of benchmarks where each method outperforms (is at least 1.5 times faster than) the other. On the right, the geometric mean (in seconds) of the instances solves by both methods.

Lemma 1 explains why SMT is worse in some suites, but not why it is better in some others. The latter happens on benchmarks with many constraints. A possible explanation could be that many of these constraints are not very active, i.e. they produce very few, if any, explanations. If this is the case, SMT has an advantage over an encoding: only active constraints will generate explanations, whereas an encoding approach would also have to encode all low-activity constraints right from the beginning. This notion of constraint activity, counting the number of times the propagator generates an explanation, is very similar to earlier activity-based lemma deletion policies in SAT solvers [GN02]. In order to evaluate how often this situation happens, we ran our SMT system computing the number of explanations each constraint generates. Results can be seen in Table 2, where we considered a constraint to have low activity if it generates less than 100 (possibly repeated) explanations.

Each row contains the data for each suite: e.g., in 74.6% of the MSU4 benchmarks between 0 and 5% of the constraints had low activity. In the PB evaluation and in the RCPSP benchmarks, the number of low-activity constraints is high and hence, this might explain why SMT behaves better than the encoding on these suites. However, in the Tomography suite, constraints tend to be very active, which refutes our conjecture of why SMT performs so well on these benchmarks.

[3] Note that rows do not add up to 100 % as benchmarks in which the two methods are comparable are not shown.

	Perc. of benchs with this perc. of low-act. constr.							
Suite	0-5%	5-10%	10-20%	20-40%	40-60%	60-80%	80-95%	95-100%
MSU4	74.6	0	0	0	24.9	0.5	0	0
DES	99.9	0	0	0	0	0	0	0.1
Tomography	100	0	0	0	0	0	0	0
PB evaluation	54	21.6	20.5	0.6	1.1	0.6	1.7	20.5
RCPSP	0	0	2.2	13.2	51.1	31.3	2.2	0

Table 2: Number of low-activity constraints in distinct benchmarks suites.

What happens in the Tomography suite is that although constraints are very active, the SMT solver does not end up generating the whole naive encoding because many explanations are repeated. Hence, a sophisticated encoding would probably generate many more clauses, as the whole constraint would be decomposed, even irrelevant parts.

To confirm this hypothesis we ran our SMT solver counting repeated explanations. Results can be seen in Table 3. Each row[4] corresponds to a different suite: e.g., the 100 in the third row indicates that all benchmarks in the Tomography suite had at least half of its constraints producing between 80 and 95% of repeated explanations. In general, if a constraint produces many repeated explanations, it is unlikely that it might end up generating its whole naive encoding. This explains why SMT has good results in this suite, as well as in PB evaluation and RCPSP. Hence, the number of repeated explanations seems to be a robust indicator of whether we should encode a constraint or use a propagator.

	Benchs with >50% of the ctrs. w./ this perc. of rep. expl.							
Suite	0-5%	5-10%	10-20%	20-40%	40-60%	60-80%	80-95%	95-100%
MSU4	53.8	9.1	11.6	8.5	2	0.8	0.2	0
DES	21.4	29.8	35.2	13.6	0	0	0	0
Tomography	0	0	0	0	0	0	100	0
PB evaluation	6.2	0	0	0	0	0.6	14.2	51.7
RCPSP	0	0	0	0	0	5.5	52.7	1.1

Table 3: The percentage of benchmark instances where more than half the constraints have a given percentage of repeated explanations.

4 Implementation and Experimental Evaluation

Taking into account Section 3, we implemented a system that processes SAT problems augmented with cardinality and pseudo-Boolean constraints. Although our approach is easily applicable much more generally, here we focus on these two types of constraints in order to be able to compare with [AS12]. Our aim is to show that a very simple approach gets the best of SMT and encoding methods. The starting point for our implementation is an SMT solver equipped with the ability of encoding cardinality constraints via Cardinality Networks and pseudo-Boolean constraints via BDDs.

In order to know which constraints to encode we need to keep track of the percentage of different explanations that the constraints generate. To do this we

[4] Note that the percentages in each row do not need to add 100.

attach to each constraint all the different explanations it produces. When an explanation is generated, we traverse the list of previous explanations, checking if it already exists. To speed up comparison, we first compare the size and only if they are equal, we compare the explanations, which are sorted to make comparison faster. This would be very expensive if constraints with many different explanations existed, but those constraints end up being encoded and after that do not cause any further bookkeeping overhead. Hence, more complex data structures would not help here. In our implementation, we only collect information during the first 2 minutes, since, according to our experiments, after that the information stabilizes.

Another important source of information to consider is how large the ad-hoc encoding of each constraint would be. If the number of generated explanations becomes close to the number of clauses the encoding requires, according to our experiments then it is advantageous to encode the constraint. Besides, if a constraint is producing many different explanations, we found that it is likely to end up generating the full (or a large part of the) naive encoding. Discovering and avoiding this situation is highly beneficial.

We also experimented with different ways of counting the number of *recent* occurrences of a given explanation in conflicts, without any significant findings.

Finally, following all previous observations, we encode a constraint if at least one of two conditions holds: (i) the number of different explanations is more than half the number of clauses of the compact, sophisticated encoding, (ii) more than 70% of the explanations are new and more than 5000 explanations have already been generated.

We compared the resulting system (**New** in the tables) with an SMT system, another one which encodes all constraints from the start (**Enc.**) and Lazy Decomposition [AS12] (**LD**). Results can be seen in Table 4. Each cell contains the number of problems that could be solved in less than the number of seconds of the corresponding column.

The first important conclusion is that we can obtain comparable, in some cases better, results than the LD method. This is worth mentioning since our approach is much simpler to implement and does not pose any requirement on the encodings to be used. Secondly, our approach always solves a very similar number of problems to the best option for each suite. Only in the DES suite, there is some difference that can be explained by the fact that SMT has extraordinarily poor performance on those benchmarks. Thus, just running the system in SMT mode for few seconds before encoding the constraints, as our new system does on these instances, has a strong negative impact because the many explanations generated in this early stage hinder the search later on. This could be mitigated by using more aggressive lemma deletion policies.

5 Conclusions and Other Related Work

This work is part of a project with the aim of deepening our understanding of what choices between SMT and encodings are optimal in practical problems.

MSU4

	$<10s$	$<30s$	$<60s$	$<120s$	$<300s$	$<600s$
Enc.	5374	5525	5578	5621	5659	5677
SMT	4322	4530	4603	4667	4737	4767
LD	5196	5414	5528	5598	5655	5674
New	5222	5479	5585	5636	5666	**5679**

DES

	$<10s$	$<30s$	$<60s$	$<120s$	$<300s$	$<600s$
Enc.	2521	3333	3692	3903	4102	**4228**
SMT	362	654	850	1023	1256	1452
LD	570	1230	1761	2525	3558	4019
New	836	2156	3293	3800	4053	4166

Tomography

	$<10s$	$<30s$	$<60s$	$<120s$	$<300s$	$<600s$
Enc.	773	1112	1314	1501	1759	1932
SMT	1457	1748	1858	1962	2014	**2021**
LD	1027	1239	1399	1561	1763	1918
New	1556	1818	1935	1971	2012	**2021**

PB evaluation

	$<10s$	$<30s$	$<60s$	$<120s$	$<300s$	$<600s$
Enc.	268	337	358	376	399	414
SMT	364	377	386	392	409	414
LD	352	371	379	388	403	**416**
New	269	341	360	381	404	415

RCPSP

	$<10s$	$<30s$	$<60s$	$<120s$	$<300s$	$<600s$
Enc.	7	22	52	91	139	175
SMT	132	179	206	224	249	**272**
LD	114	160	178	189	216	228
New	111	169	202	225	249	271

Table 4: Comparison among different methods on all benchmarks suites.

Here we have seen that the use of adaptive strategies is clearly advantageous. Moreover, we have given a simpler approach for which it becomes possible to handle many other types of constraints, as we plan to investigate next.

Another possibility for future work concerns the version of SMT in which explanation clauses are generated and learned immediately when a constraint propagates, as in the initial version of Lazy Clause Generation [OSC07], which worked remarkably well on resource-constrained project scheduling problems (RCPSPs). Indeed, we have now discovered that in these problems the number of different explanations is specially low. It may turn out to be advantageous to handle these constraints with clauses, which are prioritized in the solver with respect to constraint propagators.

Related Work. Apart from [AS12], another related proposal is [MP11]: to solve a propositional formula F plus additional pseudo-Boolean constraints, the SAT solver first finds a model M for F ("unsat" if there is none); then a few of the constraints that are false in M are picked ("sat" if there is none), simplified using the unit clauses found so far, encoded, and added to F; and the process is iterated. A drawback of this method is that it may fully encode low-activity constraints just because they happen to be false in M, whereas we really monitor activity. Also, we only need one run of the solver. Finally, it is clear that any method that encodes on the fly (including ours) can simplify constraints with the unit clauses available at that point.

Acknoledgments. First four authors partially supported by MEC/MICINN under SweetLogics project (TIN 2010-21062-C02-01). Abío is also supported by DFG Graduiertenkolleg 1763 (QuantLA). 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

AG09. Anbulagan and Alban Grastien. Importance of Variables Semantic in CNF Encoding of Cardinality Constraints. In V. Bulitko and J. C. Beck, editors, *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA '09*. AAAI, 2009.

ANO+12. Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. *J. Artif. Intell. Res. (JAIR)*, 45:443–480, 2012.

ANORC09. Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *Int. Conf. Theory and Applications of Satisfiability Testing (SAT), LNCS 4501*, pages 167–180, 2009.

ANORC11. Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

AS12. Ignasi Abío and Peter J. Stuckey. Conflict directed lazy decomposition. In Michela Milano, editor, *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2012.

BB03. Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In F. Rossi, editor, *Principles and Practice of Constraint Programming, 9th International Conference, CP '03*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.

BBR06. Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo boolean constraints to sat. *JSAT*, 2(1-4):191–200, 2006.

BBR09. Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In O. Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing, SAT '09*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2009.

BHvMW09. Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

CZI10. Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 154–172. Springer, 2010.

ES06. Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

GN02. Eugene Goldberg and Yakov Novikov. BerkMin: A Fast and Robust SAT-Solver. In *2002 Conference on Design, Automation, and Test in Europe, DATE '02*, pages 142–149. IEEE Computer Society, 2002.

MP11. Panagiotis Manolios and Vasilis Papavasileiou. Pseudo-Boolean solving by incremental translation to SAT. In *Formal Methods in Computer-Aided Design*, FMCAD '11, 2011.

MSP08. J. Marques-Silva and J. Planes. Algorithms for Maximum Satisfiability using Unsatisfiable Cores. In *2008 Conference on Design, Automation and*

Test in Europe Conference, DATE '08, pages 408–413. IEEE Computer Society, 2008.

NOT06. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM, JACM*, 53(6):937–977, 2006.

OSC07. Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = lazy clause generation. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer, 2007.

Sin05. C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In P. v. Beek, editor, *Principles and Practice of Constraint Programming, 11th International Conference, CP '05*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

uR05. M. Büttner and J. Rintanen. Satisfiability planning with constraints on the number of actions. In S. Biundo, K. L. Myers, and K. Rajan, editors, *15th International Conference on Automated Planning and Scheduling, ICAPS '05*, pages 292–299. AAAI, 2005.

Var09. Moshe Y. Vardi. Symbolic techniques in propositional satisfiability solving. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 2–3. Springer, 2009.

War98. Joost P. Warners. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Information Processing Letters*, 68(2):63–69, 1998.