Generalized Totalizer Encoding for Pseudo-Boolean Constraints

Saurabh Joshi[1], Ruben Martins[1], and Vasco Manquinho[2]

[1] Department of Computer Science, University of Oxford, UK
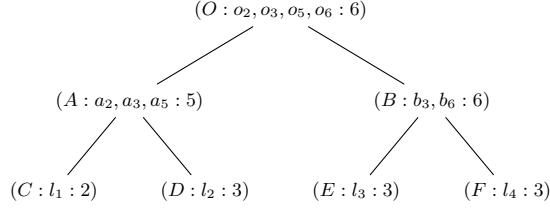{saurabh.joshi,ruben.martins}@cs.ox.ac.uk
[2] INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
vasco.manquinho@inesc-id.pt

Abstract. Pseudo-Boolean constraints, also known as 0-1 Integer Linear Constraints, are used to model many real-world problems. A common approach to solve these constraints is to encode them into a SAT formula. The runtime of the SAT solver on such formula is sensitive to the manner in which the given pseudo-Boolean constraints are encoded. In this paper, we propose generalized Totalizer encoding (GTE), which is an arc-consistency preserving extension of the Totalizer encoding to pseudo-Boolean constraints. Unlike some other encodings, the number of auxiliary variables required for GTE does not depend on the magnitudes of the coefficients. Instead, it depends on the number of distinct combinations of these coefficients. We show the superiority of GTE with respect to other encodings when large pseudo-Boolean constraints have low number of distinct coefficients. Our experimental results also show that GTE remains competitive even when the pseudo-Boolean constraints do not have this characteristic.

1 Introduction

Pseudo-Boolean constraints (PBCs) or 0-1 Integer Linear constraints have been used to model a plethora of real world problems such as computational biology [13,24], upgradeability problems [3,15,16], resource allocation [27], scheduling [26] and automated test pattern generation [22]. Due to its importance and a plethora of applications, a lot of research has been done to efficiently solve PBCs. One of the popular approaches is to convert PBCs into a SAT formula [7,11,21] thus making them amenable to off-the-shelf SAT solvers. We start by formally introducing PBC, followed by a discussion on how to convert a PBC into a SAT formula.

A PBC is defined over a finite set of Boolean variables $x_1, \ldots, x_n$ which can be assigned a value 0 ($false$) or 1 ($true$). A literal l_i is either a Boolean variable x_i (positive literal) or its negation $\neg x_i$ (negative literal). A positive (resp. negative) literal l_i is said to be assigned 1 if and only if the corresponding variable x_i is assigned 1 (resp. 0). Without a loss of generality, PBC can be defined as a linear inequality of the following normal form:

$$(O : o_2, o_3, o_5, o_6 : 6)$$

$(A : a_2, a_3, a_5 : 5)$ $\qquad\qquad$ $(B : b_3, b_6 : 6)$

$(C : l_1 : 2)$ $\quad$ $(D : l_2 : 3)$ $\qquad$ $(E : l_3 : 3)$ $\quad$ $(F : l_4 : 3)$

Fig. 1: Generalized Totalizer Encoding for $2l_1 + 3l_2 + 3l_3 + 3l_4 \leq 5$

$$\sum w_i l_i \leq k \tag{1}$$

Here, $w_i \in \mathbb{N}^+$ are called coefficients or weights, l_i are input literals and $k \in \mathbb{N}^+$ is called the bound. Linear inequalities in other forms (e.g. other inequality, equalities or negative coefficients) can be converted into this normal form in linear time [8]. *Cardinality constraint* is a special case of PBC when all the weights have the value 1. Many different encodings have been proposed to encode cardinality constraints [4,5,25,28]. Linear pseudo-Boolean solving (PBS) is a generalization of the SAT formulation where constraints are not restricted to clauses and can be PBCs. A related problem to PBS is the linear pseudo-Boolean optimization (PBO) problem, where all the constraints must be satisfied and the value of a linear cost function is optimized. PBO usually requires an iterative algorithm which solves a PBS in every iteration [11,18,19,21]. Considering that the focus of the paper is on encodings rather than algorithms, we restrict ourselves to the decision problem (PBS).

This paper makes the following contributions.

– We propose an arc-consistency [12] preserving extension of Totalizer encoding [5] called Generalized Totalizer encoding (GTE) in Section 2.
– We compare various PBC encoding schemes that were implemented in a common framework, thus providing a fair comparison. After discussing related work in Section 3, we show GTE as a promising encoding through its competitive performance in Section 4.

2   Generalized Totalizer Encoding

The Totalizer encoding [5] is an encoding to convert cardinality constraints into a SAT formula. In this section, the generalized Totalizer encoding (GTE) to encode PBC into SAT is presented. GTE can be better visualized as a binary tree, as shown in Fig. 1. With the exception of the leaves, every node is represented as (*node_name* : *node_vars* : *node_sum*). The *node_sum* for every node represents the maximum possible weighted sum of the subtree rooted at that node. For any node A, a node variable a_w represents a weighted sum w of the underlying

subtree. In other words, whenever the weighted sum of some of the input literals in the subtree becomes w, a_w must be set to 1. Note that for any node A, we would need one variable corresponding to every distinct weighted sum that the input literals under A can produce. Input literals are at the leaves, represented as ($node\_name : literal\_name : literal\_weight$) with each of the terms being self explanatory.

For any node P with children Q and R, to ensure that weighted sum is propagated from Q and R to P, the following formula is built for P:

$$
\left(\bigwedge_{\substack{q_{w_1} \in Q.node\_vars \\ r_{w_2} \in R.node\_vars \\ w_3 = w_1 + w_2 \\ p_{w_3} \in P.node\_vars}} (\neg q_{w_1} \vee \neg r_{w_2} \vee p_{w_3}) \right) \wedge \left(\bigwedge_{\substack{s_w \in (Q.node\_vars \cup R.node\_vars) \\ w = w' \\ p_{w'} \in P.node\_vars}} (\neg s_w \vee p_{w'}) \right) \quad (2)
$$

The left part of Eqn. (2) ensures that, if node Q has witnessed a weighted sum of w_1 and R has witnessed a weighted sum of w_2, then P must be considered to have witnessed the weighted sum of $w_3 = w_1 + w_2$. The right part of Eqn. (2) just takes care of the boundary condition where weighted sums from Q and R are propagated to P without combining it with their siblings. This represents that Q (resp. R) has witnessed a weighted sum of w but R (resp. Q) may not have witnessed any positive weighted sum.

Note that node O in Fig. 1 does not have variables for the weighted sums larger than 6. Once the weighted sum goes above the threshold of k, we represent it with $k + 1$. Since all the weighted sums above k would result in the constraint being not satisfied, it is sound to represent all such sums as $k + 1$. This is in some sense a generalization of k-simplification described in [9,17]. For k-simplification, w_3 in Eqn. (2) would change to $w_3 = min(w_1 + w_2, k + 1)$.

Finally, to enforce that the weighted sum does not exceed the given threshold k, we add the following constraint at the root node O :

$$
\neg o_{k+1} \quad (3)
$$

Encoding properties: Let A_{I_w} represent the multiset of weights of all the input literals in the subtree rooted at node A. For any given multiset S of weights, let $Weight(S) = \sum_{e \in S} e$. For a given multiset S, let $unique(S)$ denote the set with all the multiplicity removed from S. Let $|S|$ denote the cardinality of the set S. Hence, the total number of node variables required at node A is:

$$
|unique\left(\{Weight(S) | S \subseteq A_{I_w} \wedge S \neq \emptyset\}\right)| \quad (4)
$$

Note that unlike some other encodings [7,14] the number of auxiliary variables required for GTE does not depend on the magnitudes of the weights. Instead, it depends on how many unique weighted sums can be generated. Thus, we claim that for pseudo-Boolean constraints where the distinct weighted sum combinations are low, GTE should perform better. We corroborate our claim in Section 4 through experiments.

Nevertheless, in the worst case, GTE can generate exponentially many auxiliary variables and clauses. For example, if the weights of input literals $l_1, \ldots, l_n$ are respectively $2^0, \ldots, 2^{n-1}$, then every possible weighted sum combination would be unique. In this case, GTE would generate exponentially many auxiliary variables. Since every variable is used in at least one clause, it will also generate exponentially many clauses.

Though GTE does not depend on the magnitudes of the weights, one can use the magnitude of the largest weight to categorize a class of PBCs for which GTE is guaranteed to be of polynomial size. If there are n input literals and the largest weight is a polynomial $P(n)$, then GTE is guaranteed to produce a polynomial size formula. If the largest weight is $P(n)$, then the total number of distinct weight combinations (Eqn. (4)) is bounded by $nP(n)$, resulting in a polynomial size formula.

The best case for GTE occurs when all of the weights are equal, in which case the number of auxiliary variables and clauses is, respectively, $\mathcal{O}(n\,log_2 n)$ and $\mathcal{O}(n^2)$. Notice that for this best case with k-simplification, we have $\mathcal{O}(nk)$ variables and clauses, since it will behave exactly as the Totalizer encoding [5].

Note also that the generalized arc consistency (GAC) [12] property of Totalizer encoding holds for GTE as well. GAC is a property of an encoding which allows the solver to infer maximal possible information through propagation, thus helping the solver to prune the search space earlier. The original proof [5] makes an inductive argument using the left subtree and the right subtree of a node. It makes use of the fact that, if there are q input variables set to 1 in the left child Q and r input variables are set to 1 in the right child R, then the encoding ensures that in the parent node P, the variable p_{q+r} is set to 1. Similarly, GTE ensures that if the left child Q contributes w_1 to the weighted sum (q_{w_1} is set to 1) and the right child R contributes w_2 to the weighted sum (r_{w_2} is set to 1), then the parent node P registers the weighted sum to be at least $w_3 = w_2 + w_1$ (p_{w_3} is set to 1). Hence, the GAC proof still holds for GTE.

3 Related Work

The idea of encoding a PBC into a SAT formula is not new. One of the first such encoding is described in [11,30] which uses binary adder circuit like formulation to compute the weighted sum and then compare it against the threshold k. This encoding creates $\mathcal{O}(n\,log_2 k)$ auxiliary clauses, but it is not arc-consistent. Another approach to encode PBCs into SAT is to use sorting networks [11]. This encoding produces $\mathcal{O}(N\,log_2^2 N)$ auxiliary clauses, where N is bounded by $\lceil log_2 w_1 \rceil + \ldots + \lceil log_2 w_n \rceil$. This encoding is also not arc-consistent for PBCs, but it preserves more implications than the adder encoding, and it maintains GAC for cardinality constraints.

The Watchdog encoding [7] scheme uses the Totalizer encoding, but in a completely different manner than GTE. It uses multiple Totalizers, one for each bit of the binary representation of the weights. The Watchdog encoding was the first polynomial sized encoding that maintains GAC for PBCs and it only gen-

erates $\mathcal{O}(n^3 log_2 n\, log_2 w_{max})$ auxiliary clauses. Recently, the Watchdog encoding has been generalized to a more abstract framework with the Binary Merger encoding [20]. Using a different translation of the components of the Watchdog encoding allows the Binary Merger encoding to further reduce the number of auxiliary clauses to $\mathcal{O}(n^2 log_2^2 n\, log_2 w_{max})$. The Binary Merger is also polynomial and maintains GAC.

Other encodings that maintain GAC can be exponential in the worst case scenario, such as BDD based encodings [1,6,11]. These encodings share quite a lot of similarity to GTE, such as GAC and independence from the magnitude of the weight. One of the differences is that GTE always has a tree like structure amongst auxiliary variables and input literals. However, the crucial difference lies in the manner in which auxiliary variables are generated, and what they represent. In BDD based approaches, an auxiliary variable D_i attempts to reason about the weighted sum of the input literals either $l_i, \ldots, l_n$ or $l_1, \ldots, l_i$. On the other hand, an auxiliary variable a_w at a node A in GTE attempts to only reason about the weighted sum of the input literals that are descendants of A. Therefore, two auxiliary variables in two disjoint subtrees in GTE are guaranteed to reason about disjoint sets of input literals. We believe that such a localized reasoning could be a cause of relatively better performance of GTE as reported in Section 4. It is worth noting that the worst case scenario for GTE, when weights are of the form a^i, where $a \geq 2$, would generate a polynomial size formula for BDD based approaches [1,6,11].

As GTE generalizes the Totalizer encoding, the Sequential Weighted Counter (SWC) encoding [14] generalizes sequential encoding [28] for PBCs. Like BDD based approaches and GTE, SWC can be exponential in the worst case.

4   Implementation and Evaluation

All experiments were performed on two AMD 6276 processors (2.3 GHz) running Fedora 18 with a timeout of 1,800 seconds and a memory limit of 16 GB. Similar resource limitations were used during the last pseudo-Boolean (PB) evaluation of 2012[3]. For a fair comparison, we implemented GTE (gte) in the PBLib [29] (version 1.2) open source library which contains a plethora of encodings, namely, Adder Networks (adder) [11,30], Sorting Networks (sorter) [11], watchdog (watchdog) [7], Binary Merger (bin-merger) [20], Sequential Weighted Counter (swc) [14], and BDDs (bdd) [1]. A new encoding in PBLib can be added by implementing `encode` method of the base class `Encoder`. Thus, all the encodings mentioned above, including GTE, only differ in how `encode` is implemented while they share the rest of the whole environment. PBLib provides parsing and normalization [11] routines for PBC and uses Minisat 2.2.0 [10] as a back-end SAT solver. When the constraint to be encoded into CNF is a cardinality constraint, we use the default setting of PBLib that dynamically selects a cardinality encoding based on the number of auxiliary clauses. When the constraint to be encoded into CNF is a PBC, we specify one of the above encodings.

---

[3] http://www.cril.univ-artois.fr/PB12/

Table 1: Characteristics of pseudo-Boolean benchmarks

Benchmark	#PB	#lits	k	max w_i	$\sum w_i$	#diff w_i
PB'12	164.31	32.25	27.94	12.55	167.14	6.72
pedigree	1.00	10,794.13	11,106.69	456.28	4,665,237.38	2.00

Table 2: Number of solved instances

Benchmark	Result	sorter	swc	adder	watchdog	bin-merger	bdd	gte
PB'12	SAT	72	74	73	79	79	**81**	**81**
(214)	UNSAT	74	77	83	**85**	**85**	84	84
pedigree	SAT	2	7	6	25	43	82	**83**
(172)	UNSAT	0	7	6	23	35	72	**75**
Total	SAT/UNSAT	146	165	172	212	242	319	**323**

Benchmarks: Out of all 355 instances from the DEC-SMALLINT-LIN category in the last PB evaluation of 2012 (PB'12), we only considered those 214 instances[4] that contain at least 1 PBC. We also consider an additional set of pedigree benchmarks from computational biology [13]. These benchmarks were originally encoded in Maximum Satisfiability (MaxSAT) and were used in the last MaxSAT Evaluation of 2014[5]. Any MaxSAT problem can be converted to a corresponding equivalent pseudo-Boolean problem [2]. We generate two pseudo-Boolean decision problems (one satisfiable, another unsatisfiable) from the optimization version of each of these benchmarks. The optimization function is transformed into a PBC with the value of the bound k set to a specific value. Let the optimum value for the optimization function be k_{opt}. The satisfiable decision problem uses k_{opt} as the value for the bound k, whereas the unsatisfiable decision problem uses $k_{opt}-1$ as the value for the bound k. Out of 200 generated instances[6], 172 had at least 1 PBC and were selected for further evaluation.

Tab. 1 shows the characteristics of the benchmarks used in this evaluation. #PB denotes the average number of PBCs per instance. #lits, k, max w_i, $\sum w_i$ and #diff w_i denote the per constraint per instance average of input literals, bound, the largest weight, maximum possible weighted sum and the number of distinct weights. PB'12 benchmarks are a mix of crafted as well as industrial benchmarks, whereas all of the pedigree benchmarks are from the same biological problem [13]. The PB'12 benchmarks have on average several PBCs, however, they are relatively small in magnitude. In contrast, the pedigree benchmarks contain one large PB constraint with very large total weighted sum. pedigree benchmarks have only two distinct values of weights, thus making them good candidates for using GTE.

Results: Tab. 2 shows the number of instances solved using different encodings. sorter, adder and swc perform worse than the remaining encodings for both sets of

---

[4] Available at `http://sat.inesc-id.pt/~ruben/benchmarks/pb12-subset.zip`

[5] `http://www.maxsat.udl.cat/14/`

[6] Available at `http://sat.inesc-id.pt/~ruben/benchmarks/pedigrees.zip`

(a) # Variables on PB'12 benchmarks

(b) # Variables on pedigree benchmarks

(c) # Clauses on PB'12 benchmarks

(d) # Clauses on pedigree benchmarks

(e) Runtime on PB'12 benchmarks

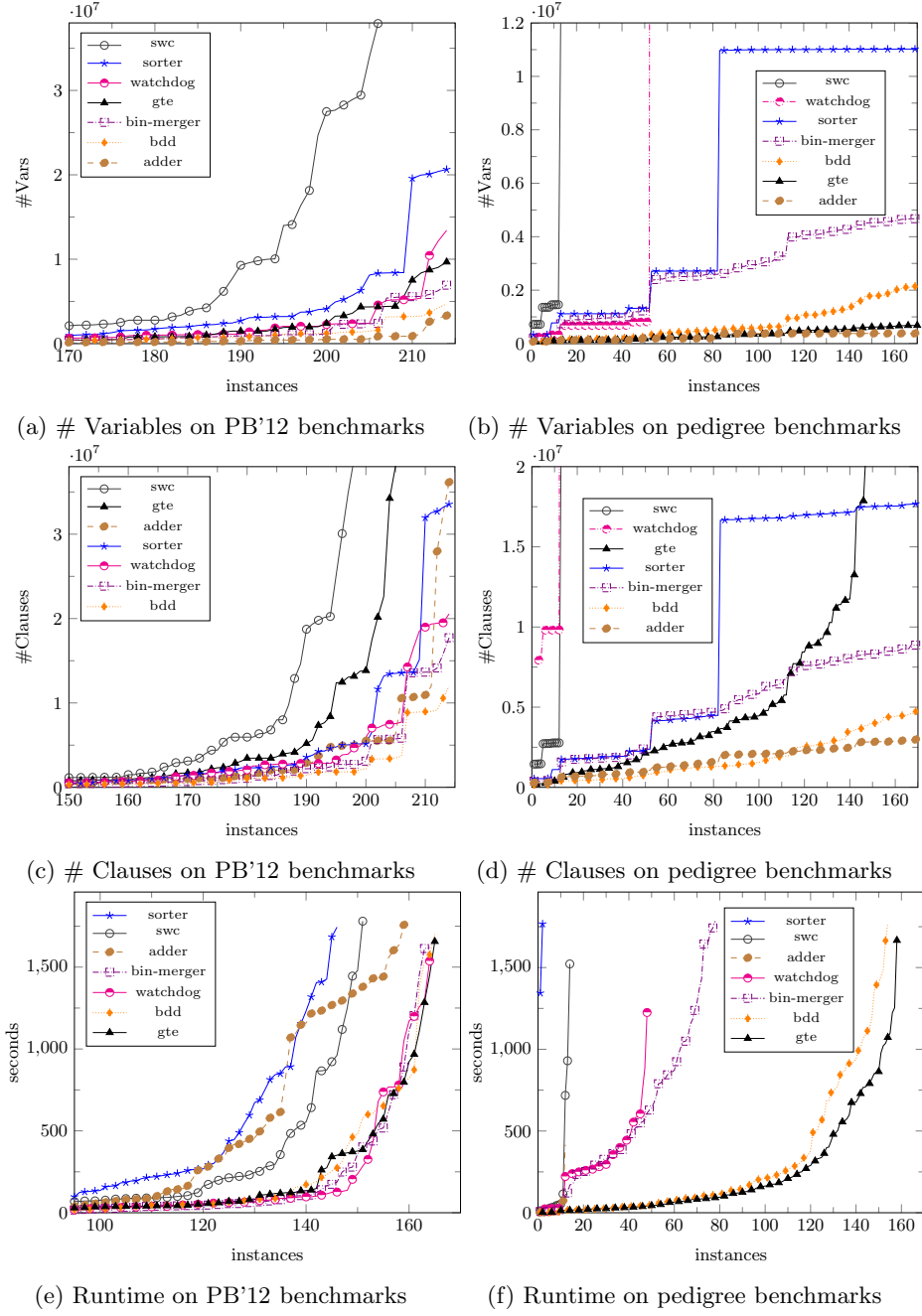(f) Runtime on pedigree benchmarks

Fig. 2: Cactus plots of number of variables, number of clauses and runtimes

benchmarks. The first two are not arc-consistent therefore the SAT solver is not able to infer as much information as with arc-consistent encodings. swc, though arc-consistent, generates a large number of auxiliary variables and clauses, which deteriorates the performance of the SAT solver.

gte provides a competitive performance to bdd, bin-merger and watchdog for PB'12. However, only the gte and bdd encodings are able to tackle pedigree benchmarks, which contain a large number of literals and only two different coefficients. Unlike other encodings, gte and bdd are able to exploit the characteristics of these benchmarks.

swc requires significantly large number of variables as the value of k increases, whereas bdd and gte keep the variable explosion in check due to reuse of variables on similar combinations (Figs. 2a and 2b). This reuse of auxiliary variables is even more evident on pedigree benchmarks (Fig. 2b) as these benchmarks have only two different coefficients resulting in low number of combinations. k-simplification also helps gte in keeping the number of variables low as all the combinations weighing more than $k + 1$ are mapped to $k + 1$.

Number of clauses required for gte is quite large as compared to some other encodings (Figs. 2c and 2d). gte requires clauses to be generated for all the combinations even though most of them produce the same value for the weighted sum, thus reusing the same variable. Though bdd has an exponential worst case, in practice it appears to generate smaller formulas (Figs. 2c and 2d).

Fig. 2e shows that gte provides a competitive performance with respect to bin-merger, watchdog and bdd. Runtime on pedigree benchmarks as shown in Fig. 2f establishes gte as the clear winner with bdd performing a close second. The properties that gte and bdd share help them perform better on pedigree benchmarks as they are not affected by large magnitude of weights in the PBCs.

5   Conclusion

Many real-world problems can be formulated using pseudo-Boolean constraints (PBC). Given the advances in SAT technology, it becomes crucial how to encode PBC into SAT, such that SAT solvers can efficiently solve the resulting formula.

In this paper, an arc-consistency preserving generalization of the Totalizer encoding is proposed for encoding PBC into SAT. Although the proposed encoding is exponential in the worst case, the new Generalized Totalizer encoding (GTE) is very competitive in relation with other PBC encodings. Moreover, experimental results show that when the number of different weights in PBC is small, it clearly outperforms all other encodings. As a result, we believe the impact of GTE can be extensive, since one can further extend it into incremental settings [23].

References

1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A New Look at BDDs for Pseudo-Boolean Constraints. Journal of Artificial Intelligence Research 45, 443–480 (2012)
2. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Generic ILP versus specialized 0-1 ILP: an update. In: International Conference on Computer-Aided Design. pp. 450–457. IEEE Press (2002)
3. Argelich, J., Berre, D.L., Lynce, I., Marques-Silva, J., Rapicault, P.: Solving Linux Upgradeability Problems Using Boolean Optimization. In: Workshop on Logics for Component Configuration. pp. 11–22 (2010)
4. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality Networks: a theoretical and empirical study. Constraints 16(2), 195–221 (2011)
5. Bailleux, O., Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints. In: Principles and Practice of Constraint Programming. LNCS, vol. 2833, pp. 108–122. Springer (2003)
6. Bailleux, O., Boufkhad, Y., Roussel, O.: A Translation of Pseudo Boolean Constraints to SAT. Journal on Satisfiability, Boolean Modeling and Computation 2(1-4), 191–200 (2006)
7. Bailleux, O., Boufkhad, Y., Roussel, O.: New Encodings of Pseudo-Boolean Constraints into CNF. In: International Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 5584, pp. 181–194. Springer (2009)
8. Barth, P.: Logic-based 0-1 Constraint Programming. Kluwer Academic Publishers (1996)
9. Büttner, M., Rintanen, J.: Satisfiability Planning with Constraints on the Number of Actions. In: International Conference on Automated Planning and Scheduling. pp. 292–299. AAAI Press (2005)
10. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: International Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 2919, pp. 502–518. Springer (2003)
11. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2(1-4), 1–26 (2006)
12. Gent, I.P.: Arc consistency in SAT. In: European Conference on Artificial Intelligence. pp. 121–125. IOS Press (2002)
13. Graça, A., Lynce, I., Marques-Silva, J., Oliveira, A.: Efficient and Accurate Haplotype Inference by Combining Parsimony and Pedigree Information. In: Algebraic and Numeric Biology. LNCS, vol. 6479, pp. 38–56. Springer (2010)
14. Hölldobler, S., Manthey, N., Steinke, P.: A Compact Encoding of Pseudo-Boolean Constraints into SAT. In: KI 2012: Advances in Artificial Intelligence. LNCS, vol. 7526, pp. 107–118. Springer (2012)
15. Ignatiev, A., Janota, M., Marques-Silva, J.: Towards efficient optimization in package management systems. In: International Conference on Software Engineering. pp. 745–755. ACM (2014)
16. Janota, M., Lynce, I., Manquinho, V.M., Marques-Silva, J.: PackUp: Tools for Package Upgradability Solving. JSAT 8(1/2), 89–94 (2012)
17. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: A Partial Max-SAT Solver. Journal on Satisfiability, Boolean Modeling and Computation 8, 95–100 (2012)
18. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation 7(2-3), 59–6 (2010)

19. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for Weighted Boolean Optimization. In: International Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 5584, pp. 495–508. Springer (2009)
20. Manthey, N., Philipp, T., Steinke, P.: A More Compact Translation of Pseudo-Boolean Constraints into CNF Such That Generalized Arc Consistency Is Maintained. In: KI 2014: Advances in Artificial Intelligence. LNCS, vol. 8736, pp. 123–134. Springer (2014)
21. Manthey, N., Steinke, P.: npSolver – A SAT Based Solver for Optimization Problems. In: Pragmatics of SAT (2012)
22. Marques-Silva, J.: Integer Programming Models for Optimization Problems in Test Generation. In: Asia and South Pacific Design Automation. pp. 481–487. IEEE Press (1998)
23. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental Cardinality Constraints for MaxSAT. In: Principles and Practice of Constraint Programming. LNCS, vol. 8656, pp. 531–548. Springer (2014)
24. Miranda, M., Lynce, I., Manquinho, V.: Inferring phylogenetic trees using pseudo-Boolean optimization. AI Communications 27(3), 229–243 (2014)
25. Ogawa, T., Liu, Y., Hasegawa, R., Koshimura, M., Fujita, H.: Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. In: International Conference on Tools with Artificial Intelligence. pp. 9 – 17. IEEE Press (2013)
26. Prestwich, S., Quirke, C.: Boolean and Pseudo-Boolean Models for Scheduling. In: International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (2003)
27. Ribas, B., Suguimoto, R., Montaño, R., Silva, F., De Bona, L., Castilho, M.: On Modelling Virtual Machine Consolidation to Pseudo-Boolean Constraints. In: Ibero-American Conference on Artificial Intelligence. LNCS, vol. 7637, pp. 361–370. Springer (2012)
28. Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: Principles and Practice of Constraint Programming. LNCS, vol. 3709, pp. 827–831. Springer (2005)
29. Steinke, P., Manthey, N.: PBLib–A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF. Tech. rep., Technische Universität Dresden (2014), `http://tools.computational-logic.org/content/pblib.php`
30. Warners, J.: A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. Information Processing Letters 68(2), 63–69 (1998)