Approximation Strategies for Incomplete MaxSAT

Saurabh Joshi[1(✉)], Prateek Kumar[1], Ruben Martins[2], and Sukrut Rao[1]

[1] Indian Institute of Technology Hyderabad, Sangareddy, India
{sbjoshi,cs15btech11031,cs15btech11036}@iith.ac.in
[2] Carnegie Mellon University, Pittsburgh, USA
rubenm@andrew.cmu.edu

Abstract. Incomplete MaxSAT solving aims to quickly find a solution that attempts to minimize the sum of the weights of the unsatisfied soft clauses without providing any optimality guarantees. In this paper, we propose two approximation strategies for improving incomplete MaxSAT solving. In one of the strategies, we cluster the weights and approximate them with a representative weight. In another strategy, we break up the problem of minimizing the sum of weights of unsatisfiable clauses into multiple minimization subproblems. Experimental results show that approximation strategies can be used to find better solutions than the best incomplete solvers in the MaxSAT Evaluation 2017.

Keywords: MaxSAT · Incomplete · Approximation

1 Introduction

Given a set of Boolean constraints in a conjunctive normal form (CNF), the problem of Maximum Satisfiability (MaxSAT) asks to provide valuation of variables so that maximum number of constraints are satisfied. These constraints can be assigned weights to prioritize some set of constraints over others, which would give rise to a weighted MaxSAT problem where the goal is to find a valuation which maximizes the sum of the weights of the satisfied constraints. Any improvements in MaxSAT solving have a huge impact because many real world problems can be encoded as MaxSAT problems (e.g., [5,14,16]).

Often, the application may be able to tolerate a suboptimal solution but requires this solution to be computed in a very short amount of time. For such cases, it tremendously helps if there are techniques and tools that can very quickly find a solution which is close enough to an optimal solution. Incomplete MaxSAT solvers [4,9,10,19,20,26] strive to find a good solution in a limited time frame. The solution, thus provided, need not be an optimal one. Therefore, for improvement, we need to develop tools and techniques that can find better solutions (closer to an optimal solution) in the same time frame.

J. Hooker (Ed.): CP 2018, LNCS 11008, pp. 219–228, 2018.
https://doi.org/10.1007/978-3-319-98334-9_15

As part of this paper, we contribute the following:

- An approximation strategy based on weight relaxation (Sect. 3), which modifies the weights of the clauses in a manner so that it is easier for the solver to find a solution quickly.
- An approximation strategy which breaks up the problem of minimizing the sum of weights of unsatisfied clauses into multiple minimization subproblems and attempts to minimize these subproblems in a greedy order (Sect. 3). This strategy can also be combined with the weight relaxation strategy.
- Empirical results on how the accuracy of the solver gets affected as we vary the weight relaxation parameter (Sect. 5).
- An implementation of these strategies using the Open-WBO framework. We also demonstrate the advantage of these approximation strategies by showing its prowess against state-of-the-art incomplete MaxSAT solvers (Sect. 5).

2 Preliminaries

Let x be a Boolean variable which can take values *true* or *false*. A literal l is a variable x or its negation $\neg x$. A clause ω is a disjunction of literals and a formula φ is a conjunction of clauses. Notationally, we will treat a clause ω and a formula φ as sets containing literals and clauses respectively.

An assignment ν maps variables to either *true* or *false*. An assignment is said to satisfy a positive literal x (resp. a negative literal $\neg x$) if $\nu(x) = true$ (resp. $\nu(x) = false$). A clause is said to be satisfied if at least one of its literals is satisfied. A formula is said to be satisfied by an assignment if all of its clauses are satisfied by the assignment. A formula is called *satisfiable* if there exists a satisfying assignment for that formula, otherwise it is called *unsatisfiable*. Boolean satisfiability problem (SAT) asks to find a satisfying assignment (i.e., model) to a formula. Maximum satisfiability (MaxSAT) problem is an optimization version where the goal is to find an assignment which satisfies the maximum number of clauses of a formula. In a partial MaxSAT problem, a partition of φ is given as two mutually exclusive sets φ_h (*hard* clauses) and φ_s (*soft* clauses), where the goal is to satisfy all the clauses in φ_h[1] while maximizing the number of clauses satisfied in φ_s. Let $weight : Clauses \rightarrow \mathbb{N}^+$ be a map from a set of clauses to positive integers. In a partial weighted MaxSAT problem, the goal is to find an assignment that maximizes the sum of weights of the satisfied soft clauses. From now on, we will refer to a weighted partial MaxSAT problem as MaxSAT.

A clause ω can be relaxed by adding a relaxation variable r so that the relaxed clause becomes $\omega \cup \{r\}$. The relaxed clause can be satisfied by either satisfying the original clause or its relaxation variable. For a formula φ, when all of its soft clauses are relaxed, we will denote it as φ^r. We define the cost of a relaxation variable r to be the weight of the clause that it relaxed, $cost(r) = weight(\omega)$. The cost of an assignment ν is defined as $cost(\nu) = \sum_{r_i:\nu(r_i)=1} cost(r_i)$. The goal of MaxSAT is to find a satisfying assignment with the minimum cost.

[1] For simplicity, we will assume that φ_h is always satisfiable.

Input : Formula φ_s, Map $weight$, partitioning parameter m
Output: Partition $P(m)$, new weight map $weight_m$

1 $n \leftarrow |\varphi_s|$
2 sort clauses of φ_s in the ascending order of weights
3 **for** $i \leftarrow 1$ **to** $n-1$ **do**
4 $\quad diff_i \leftarrow weight(\omega_{i+1}) - weight(\omega_i)$
5 $\langle i_1, \ldots, i_{m-1} \rangle \leftarrow$ sorted indices where top $(m-1)$ difference $diff_i$ occurs
6 $c_1 \leftarrow \{\omega_1, \ldots, \omega_{i_1}\}$
7 **for** $j \leftarrow 2$ **to** $m-1$ **do**
8 $\quad c_j \leftarrow \{\omega_{i_{j-1}+1}, \ldots, \omega_{i_j}\}$
9 $c_m \leftarrow \{\omega_{i_{m-1}+1}, \ldots, \omega_n\}$
10 $P(m) \leftarrow \{c_1, \ldots, c_m\}$
11 **foreach** $c_i \in P(m)$ **do**
12 $\quad$ **foreach** $\omega_j \in c_i$ **do**
13 $\quad\quad weight_m[\omega_j] \leftarrow RepresentativeWeight(c_i)$
14 **return** $\langle P(m), weight_m \rangle$

Algorithm 1. Partitioning and weight approximation

Input: Formula φ^r, weight maps $weight_m$, and $weight$
Output: model to φ

1 $(\mathsf{model}, \mu, \varphi_W) \leftarrow (\emptyset, +\infty, \varphi^r)$
2 status = SAT
3 **while** status = SAT **do**
4 $\quad(\mathsf{status}, \nu) \leftarrow \mathtt{SAT}(\varphi_W)$
5 $\quad$ **if** status = SAT **then**
6 $\quad\quad$ **if** $cost(\nu) < cost(\mathsf{model})$ **then**
7 $\quad\quad\quad \mathsf{model} \leftarrow \nu$
8 $\quad\quad \mu \leftarrow cost_m(\nu)$
9 $\quad\quad \varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}((\sum_{r \in V_R}(cost_m(r) \cdot r)) \leq \mu - 1)\}$
10 **return** model

Algorithm 2. Linear search Sat-Unsat algorithm for MaxSAT

3 Approximation Strategies

In this section, we describe two approximation strategies that can allow MaxSAT algorithms to converge faster to lower cost solutions. Note that the best model found by approximation strategies is not guaranteed to be an optimal solution of the original MaxSAT formula.

Weight-Based Approximation. Let $P_m(\varphi_s) = \{c_1, \ldots, c_m\}$ be a partition of φ_s into m mutually exclusive sets $c_1, \ldots, c_m$ such that $\bigcup_{1 \leq i \leq m} c_i = \varphi_s$ and $\forall_{i \neq j} : c_i \cap c_j = \emptyset$. We will call sets $c_1, \ldots, c_m$ as clusters of the partition.

Given a formula φ_s, Algorithm 1 partitions the clauses into clusters as follows. All soft clauses are sorted by their weights (Line 2). Then, differences in weights between two consecutive clauses are calculated (Line 4). $m-1$ indices are picked where the weight differences are amongst the top $m-1$ weight differences (Line 5). These indices are used as boundaries to create clusters (Lines 6–9). This way of clustering is similar to single-link agglomerative clustering [15]. Finally, a new weight map $weight_m$ is created, where all the clauses in the same cluster get the same weight (Lines 11–13). *RepresentativeWeight* (Line 13) indicates any representative weight for the cluster. In this paper, we use the *arithmetic mean* of the weights of the clauses in a cluster as the representative weight. In principle, other representative weights can also be chosen which may have different effect on

Input: $\varphi = \varphi_h \cup \varphi_s$, weight maps $weight$ and $weight_m$, Partition $P(m)$
Output: model to φ
1 $(\mathsf{model}, \boldsymbol{\mu}, \varphi_W, \mathcal{C}) \leftarrow (\emptyset, +\boldsymbol{\infty}, \varphi^r, P_m(\varphi_s)))$
2 **foreach** $c_i \in \mathcal{C}$ *in the descending order of* $weight_m(c_i)$ **do**
3 $\quad V_i \leftarrow V_R \cap c_i$
4 $\quad \mathsf{status} = \mathsf{SAT}$
5 $\quad$ **while** status $=$ SAT **do**
6 $\quad\quad (\mathsf{status}, \nu) \leftarrow \mathtt{SAT}(\varphi_W)$
7 $\quad\quad$ **if** status $=$ SAT **then**
8 $\quad\quad\quad$ **if** $cost(\nu) < cost(\mathsf{model})$ **then**
9 $\quad\quad\quad\quad \mathsf{model} \leftarrow \nu$
10 $\quad\quad\quad \mu_i \leftarrow |\{r \in V_i \mid \nu(r) = 1\}|$
11 $\quad\quad\quad \varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_i} r \leq \mu_i - 1)\}$
12 $\quad\quad$ **else**
13 $\quad\quad\quad \varphi_W \leftarrow \varphi^r$
14 $\quad\quad\quad$ **foreach** $c_j \in \mathcal{C}$ *such that* $weight_m(c_j) \geq weight_m(c_i)$ **do**
15 $\quad\quad\quad\quad \varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_j} r \leq \mu_j)\}$
16 **return** model

Algorithm 3. Clustering-based algorithm for MaxSAT

how much an algorithm can deviate from finding the minimum cost assignment. It is redundant to have $m > \#weights$, where $\#weights$ are different number of weights, because for $m \geq \#weights$, $weight = weight_m$. Algorithm 1 can be combined with any search algorithm and as m increases the deviation of the search algorithm from an optimal solution decreases. If $m = 0$ it is assumed that no partitioning is done.

There are encodings which perform better when $\#weights$ is small [12,17]. Such encodings can benefit from approximation of weights because it results in a smaller size formula when converted to CNF. This can be used with a cost minimization algorithm for MaxSAT such as the linear search Sat-Unsat algorithm [8,18] shown in Algorithm 2. In this algorithm, all the clauses in φ_s are initially relaxed, and the set of corresponding relaxation variables is denoted as V_R. A working formula φ_W is initialized with the relaxed formula φ^r. The cost of an empty model is assumed to be $+\infty$. Our primary goal is to find a satisfying assignment ν to φ with the minimum $cost(\nu)$. Algorithm 2 iteratively asks a SAT solver if there is a satisfying assignment, with its cost at most $\mu - 1$ (Line 9). The approximation comes from Algorithm 2 using $cost_m$ instead of $cost$ to encode a pseudo-Boolean (PB) constraint that restricts the cost of relaxation variables being set to *true* (Line 9). Since $cost_m$ is an approximation of $cost$, minimizing $cost_m$ does not necessarily translate to minimization w.r.t. $cost$. Therefore, we update model, only when a satisfying assignment indeed reduces the previous value of $cost(\mathsf{model})$ (Lines 6–7).

Approximation via Subproblem Minimization. Algorithm 3 proceeds in a greedy manner by processing each cluster in the descending order of its representative weight (Line 2). V_i indicates a set of relaxation variables corresponding to the clauses in c_i (Line 3). Minimization of the cost of a satisfying assignment is divided in subproblems by minimizing the number of unsatisfied clauses in clusters, starting from highest representative weight to the lowest (Line 2). For each cluster, the number of unsatisfied clauses are minimized by iteratively

reducing the upper bound μ_i on the number of relaxation variables in V_i that can be set to *true* (Lines 10–11). In the process, the minimum cost assignment seen so far is recorded (Lines 8–9). Since within any cluster, all the clauses have the same $weight_m$, only cardinality constraints are used to restrict the number of unsatisfied clauses within μ_i (Line 11). Once μ_i can not be reduced further, it is frozen by adding upper bound μ_i for all the cluster seen so far (Lines 12–15). Since the minimization is done locally as a minimization subproblem at a cluster level, rather than looking at the whole formula, this procedure is not guaranteed to converge to a globally optimum solution.

4 Related Work

Approaches for incomplete MaxSAT solving can primarily be divided into two categories: (i) stochastic MaxSAT solvers [9,10,13,19,20] and (ii) complete MaxSAT solvers that can find intermediate solutions [4,8,11,18,23,25].

Incomplete MaxSAT. Stochastic solvers start by finding a random assignment ν for φ. Since this assignment is unlikely to satisfy all clauses in φ, they choose a clause ω_i that is unsatisfied by ν and flip the assignment of a variable in ω_i such that ω_i becomes satisfied. When compared to local SAT solvers, stochastic MaxSAT solvers have additional challenges since they must find an assignment ν that satisfies φ_h while attempting to minimize the cost of the unsatisfied soft clauses. Stochastic MaxSAT solvers are particularly effective for random benchmarks but their performance tends to deteriorate for industrial benchmarks. Since the MaxSAT Evaluation 2017 (MSE2017) [2] did not contain any random instances, there were no stochastic MaxSAT solvers in the MSE2017.

Complete MaxSAT. Complete solvers can often find intermediate solutions to φ before finding an optimal assignment ν. MaxSAT solvers based on linear search algorithms [8,18,23] can find a sequence of intermediate solutions that converge to an optimal solution. These solvers use PB constraints to enforce convergence. While SAT4J [8] uses specialized data structures for PB constraints to avoid their conversion to CNF, other solvers such as QMaxSAT [18] convert the PB constraint into clauses using PB encodings [12,17,27]. Some MaxSAT solvers which are based on the implicit hitting set approach [11,25] maintain a lower and an upper bound on the values of the solution. These solvers can also be used for incomplete MaxSAT since they are also able to find intermediate solutions. Another approach for complete MaxSAT solving is to use unsatisfiability-based algorithms [1,4,24]. These algorithms use unsatisfiable subformulas to increase a lower bound on the cost of a solution until they find an optimal solution. For weighted MaxSAT, these algorithms employ a stratified approach [3] where they start by considering only a subset of the soft clauses with the largest weights and iteratively add more soft clauses when the subformula becomes satisfiable. An intermediate solution is found at each iteration. WPM3 [4] is an example of an unsatisfiable-based solver that can be used for incomplete MaxSAT and was the best incomplete MaxSAT solver in the MSE2016 [6]. maxroster [26] was the

winner of the incomplete track for Weighted MaxSAT in the MSE2017. It is a hybrid solver that combines an initial short phase of a stochastic algorithm [13] with complete MaxSAT algorithms [18,24].

Boolean Multilevel Optimization. The clustering-based algorithm presented in Algorithm 3 is closely related to Boolean Multilevel Optimization (BMO) [21]. BMO is a technique for identifying lexicographic optimization conditions, i.e. the existence of an ordered sequence of objective functions. Let M_i be the minimum weight of soft clauses in a cluster c_i. Consider a sequence of clusters $c_1, \ldots, c_m$ arranged in a descending order of M_i. A MaxSAT formula is an instance of BMO if for every cluster c_i, M_i is larger than the sum of the weights of all soft clauses in clusters $c_{i+1}, \ldots, c_m$. If this condition holds then the result of Algorithm 3 is equivalent to solving a BMO formula. However, when using the proposed clustering-based algorithm on partitions that do not preserve the BMO condition, it is not guaranteed that the solution found by Algorithm 3 is an optimal solution for φ. Our approach differs from previous complete approaches in using approximation strategies that do not preserve optimality but are more likely to converge faster to a better solution.

5 Experimental Results

To evaluate incomplete MaxSAT solvers we used the scoring mechanism from MaxSAT Evaluations 2017 (MSE2017) [2]. Given a formula φ, the score for a solver $\mathcal{S}$ is computed by the ratio of the cost (sum of weights of unsatisfied clauses) of the best solution known for φ, denoted as $best(\varphi)$,[2] to the best cost found by $\mathcal{S}$, denoted as $cost^{\mathcal{S}}(\varphi)$.[3] The score for $\mathcal{S}$ for a set of n benchmarks is given by the average score ($[0, 1]$) as follows:

$$\mathsf{score}(\mathcal{S}) = \frac{\sum_{i=1}^{n} \frac{best(\varphi_i)}{cost^{\mathcal{S}}(\varphi_i)}}{n} \tag{1}$$

$\mathsf{score}(\mathcal{S})$ shows how close on average is a solver S to the best known solution.

All the experiments were conducted on Intel® Xeon® E5-2620 v4 processors with a memory limit of 32 GB and time limits of 10, 60 and 300 s. We have used a non-standard timeout of 10 s to show that approximation strategies can find good solutions very quickly. We used the 156 benchmarks for incomplete MaxSAT from MSE2017 [2]. Note that most of these benchmarks are challenging for complete solvers and have unknown optimal solutions.

We have implemented all the algorithms presented in this paper in Open-WBO-Inc. Open-WBO-Inc is built on top of Open-WBO [23] which uses Glucose [7] as the underlying SAT solver. We used Generalized Totalizer Encoding (GTE) [17] and incremental Totalizer encoding [22] to translate PB constraints and cardinality constraints into CNF, respectively.

[2] $best(\varphi)$ is the cost of the best solution found by any solver in this evaluation.

[3] We consider a score of 0 if $\mathcal{S}$ did not find any solution to φ.

We evaluated Open-WBO-Inc by conducting experiments that are designed to answer the following questions: (1) What is the impact of the number of clusters in the quality of the solution found by our approximation strategies? (2) How does Open-WBO-Inc compare against state-of-the-art incomplete MaxSAT solvers?

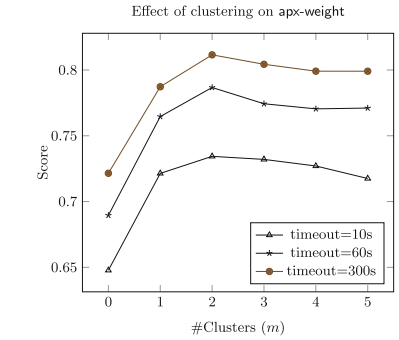
Impact of the Number of Clusters. We measure the impact of partitioning parameter m on the accuracy of the results. Figure 1a shows the score of Algorithm 2 with GTE encoding (henceforth called apx-weight). Figure 1a shows that apx-weight performs the worst when no partitioning is done. This is attributed to the fact that in the absence of any partitioning, the size of the underlying encoding is dictated by $\#weights$, where $\#weights$ are the number of different weights in the weight map. Figure 1c shows a measure of increase in formula size as m varies. The Y-axis shows the ratio of the formula size after the PB encoding to the size of the original input formula. Because of the weight-based approximation, the reduction on the $\#weights$ leads to a smaller encoding, thus making it easier for the underlying SAT solver. As m increases, the possible deviation from an optimal cost also decreases, thereby resulting in increased scores. The degradation for larger m is attributed to larger size of the formula. As the timeout is increased, the score increases because Algorithm 2 has more time and can do more iterations to reduce $cost_m(\mathsf{model})$.

Figure 1b shows that similar scoring trends are witnessed for Algorithm 3 (henceforth called apx-subprob). As apx-subprob uses only cardinality constraints, the formula size is not much sensitive to m. As m increases, the scores also improve, with the best scores achieved when $m = \#weights$. apx-subprob is guaranteed to find optimal solution only if BMO condition holds and $m = \#weights$. Only 3 out of 156 benchmarks have the BMO condition and apx-subprob with $m = \#weights$ does not terminate for any of them. However, apx-subprob using a 300 s time limit terminates for 94 out of 156 benchmarks which shows that apx-subprob quickly finds a good solution.

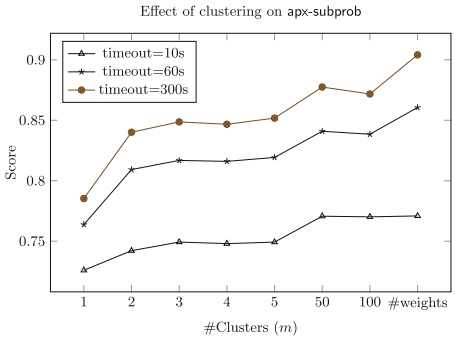
Comparison Against State-of-the-Art MaxSAT Solvers. We compared the best version of Open-WBO-Inc for weight-based approximation, apx-weight with $m = 2$, and subproblem minimization approximation, apx-subprob with $m = \#weights$, with maxroster [26], WPM3 [4] and QMaxSAT [18]. maxroster and WPM3 were the winners of the incomplete weighted category of the MSE2017 and MSE2016, respectively. QMaxSAT was placed second on the complete category of the MSE2017 and uses the algorithm described in Algorithm 2.[4]

As shown in Fig. 1d, for a 10 s timeout, both apx-weight and apx-subprob perform better than all the other solvers with apx-subprob performing the best. This demonstrates that approximation strategies are quite effective when we want to quickly find a solution which is closer to an optimal solution. For 60 and 300 s timeout, apx-subprob performs the best with maxroster being second and apx-weight outperforming WPM3 and QMaxSAT. Even though apx-weight

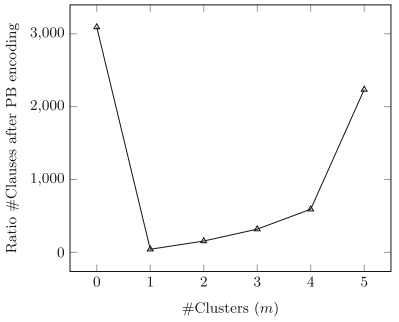
[4] Even though MaxHS [11] placed first in the complete weighted category of the MSE2017, its incomplete version is not as competitive as the other solvers [2].



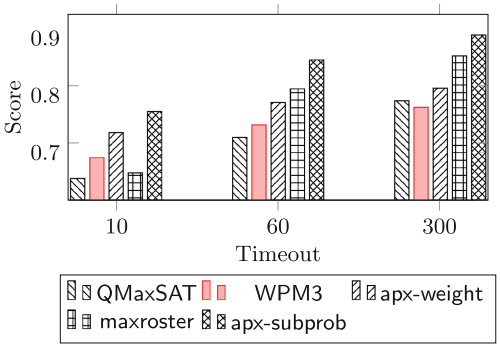
(a) apx-weight Cluster v/s Score



(b) apx-subprob Cluster v/s Score



(c) Effect of clustering on formula size for apx-weight



(d) Comparison of solvers

Fig. 1. Impact of clustering and comparison against state-of-the-art

with $m = 0$ performs worse than QMaxSAT, it outperforms QMaxSAT when clustering is used. apx-subprob outperforms all other solvers and these results prove the efficacy of approximation strategies with respect to the state-of-the-art in incomplete MaxSAT solving.

6 Conclusion and Future Work

Approximation strategies, be it weight-based relaxation or subproblem minimization, are not guaranteed to find an optimal solution even when unlimited time is given. However, they serve the purpose of quickly finding a good solution. Our experiments have successfully demonstrated that with the right parameters, these strategies can outperform the best incomplete solvers. In future, we would like to explore the application of approximation strategies to complete algorithms. In particular, progressively increasing the number of clusters and using approximation strategies to find good initial upper bounds that can later be exploited by complete MaxSAT algorithms.

Acknowledgements. This work is partially funded by ECR 2017 grant from SERB, DST, India, NSF award #1762363 and CMU/AIR/0022/2017 grant. Authors would like to thank the anonymous reviewers for their helpful comments, and Saketha Nath for lending his servers for the experiments.

References

1. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT algorithm using cardinality constraints of bounded size. In: Proceedings of International Joint Conference on Artificial Intelligence, pp. 2677–2683. AAAI Press (2015)
2. Ansótegui, C., Bacchus, F., Järvisalo, M., Martins, R.: MaxSAT Evaluation 2017 (2017). http://mse17.cs.helsinki.fi/. Accessed 18 Apr 2017
3. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSAT solvers. In: Milano, M. (ed.) CP 2012. LNCS, pp. 86–101. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_9
4. Ansótegui, C., Gabàs, J.: WPM3: an (in)complete algorithm for weighted partial MaxSAT. Artif. Intell. **250**, 37–57 (2017)
5. Argelich, J., Le Berre, D., Lynce, I., Marques-Silva, J., Rapicault, P.: Solving linux upgradeability problems using Boolean optimization. In: Proceedings of Workshop on Logics for Component Configuration, pp. 11–22. EPTCS (2010)
6. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT Evaluation 2016 (2016). http://maxsat.ia.udl.cat/. Accessed 18 Apr 2016
7. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of International Joint Conference on Artificial Intelligence, pp. 399–404. AAAI Press (2009)
8. Le Berre, D., Parrain, A.: The SAT4J library, release 2.2. JSAT **7**(2–3), 59–64 (2010)
9. Cai, S., Luo, C., Thornton, J., Su, K.: Tailoring local search for partial MaxSat. In: Proceedings of AAAI Conference on Artificial Intelligence, pp. 2623–2629. AAAI Press (2014)
10. Cai, S., Luo, C., Zhang, H., From decimation to local search and back: a new approach to MaxSAT. In: Proceedings of AAAI Conference on Artificial Intelligence, pp. 571–577. AAAI Press (2017)
11. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_19
12. Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. JSAT **2**(1–4), 1–26 (2006)
13. Fan, Y., Ma, Z., Kaile, S., Sattar, A., Li, C.: Ramp: a local search solver based on make-positive variables. In: Proceedings of MaxSAT Evaluation (2016)
14. Feng, Y., Bastani, O., Martins, R., Dillig, I., Anand, S.: Automated synthesis of semantic malware signatures using maximum satisfiability. In: Proceedings of Network and Distributed System Security Symposium (2017)
15. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**(3), 241–254 (1967)
16. Jose, M., Majumdar, R.: Cause clue clauses: error localization using maximum satisfiability. In: Proceedings of Conference on Programming Language Design and Implementation, pp. 437–446. ACM (2011)

17. Joshi, S., Martins, R., Manquinho, V.: Generalized totalizer encoding for pseudo-Boolean constraints. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 200–209. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23219-5_15
18. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: a partial Max-SAT solver. JSAT **8**(1/2), 95–100 (2012)
19. Luo, C., Cai, S., Kaile, S., Huang, W.: CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artif. Intell. **243**, 26–44 (2017)
20. Luo, C., Cai, S., Wei, W., Jie, Z., Kaile, S.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Comput. **64**(7), 1830–1843 (2015)
21. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. Artif. Intell. **62**(3–4), 317–343 (2011)
22. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental Cardinality Constraints for MaxSAT. In: O?Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_39
23. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular MaxSAT solver'. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_33
24. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Cham (2014)
25. Saikko, P., Berg, J., Järvisalo, M.: LMHS: a SAT-IP hybrid MaxSAT solver. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 539–546. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_34
26. Sugawara, T.: MaxRoster: solver description. In: Proceedings MaxSAT Evaluation 2017: Solver and Benchmark Descriptions, vol. B-2017-2, p. 12. University of Helsinki, Department of Computer Science (2017)
27. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. **68**(2), 63–69 (1998)