Journal on Satisfiability, Boolean Modeling, and Computation 11 (2019) 73-97

Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT

Saurabh Joshi sbjoshi@iith.ac.in
Prateek Kumar cs15btech11031@iith.ac.in
Sukrut Rao cs15btech11036@iith.ac.in
Department of Computer Science and Engineering, *
Indian Institute of Technology Hyderabad,
Kandi, Sangareddy, Telangana - 502285, India

Ruben Martins rubenm@andrew.cmu.edu
Department of Computer Science,
Carnegie Mellon University,
5000 Forbes Ave, Pittsburgh, PA 15213, USA

Abstract

Incomplete MaxSAT solving aims to quickly find a solution that attempts to minimize the sum of the weights of unsatisfied soft clauses without providing any optimality guarantees. In this paper, we propose two approximation strategies for improving incomplete weighted 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. We have implemented these strategies in a tool Open-WBO-Inc. Using the subproblem minimization strategy, Open-WBO-Inc placed first and second in the weighted incomplete tracks in the MaxSAT Evaluation 2018 whereas the strategy based on weight approximation was placed fourth. We compare these strategies with the best incomplete MaxSAT solvers on benchmarks taken from MaxSAT Evaluation 2017 and MaxSAT Evaluation 2018 and show that the strategies proposed are competitive with the best of the solvers.

KEYWORDS: *MaxSAT, Incomplete, Weighted, Approximation*

Submitted November 2019; revised April 2019; published September 2019

1. Introduction

Given a set of Boolean constraints in conjunctive normal form (CNF), the problem of Maximum Satisfiability (MaxSAT) asks to provide a valuation of variables so that the 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. Weighted MaxSAT is more expressive than unweighted MaxSAT since it allows the usage of weights to prioritize constraints but it is also more challenging for MaxSAT algorithms since these weights can be very large. Any improvements in MaxSAT

* Authors are listed in alphabetical order.

solving have a huge impact because many real world problems can be encoded as MaxSAT problems (e.g., [5, 22, 18]).

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 [39, 4, 11, 29, 28, 12] 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.

Recently, we have proposed novel strategies to approximate weighted MaxSAT [23]. As an extended version of that paper, we contribute the following:

- An approximation strategy based on weight relaxation (Section 3.1), 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 (Section 3.2). This strategy can also be combined with the weight relaxation strategy.
- As an addition to the earlier version, an analysis of deviations from the optimal caused by these strategies under various scenarios (Section 3.1.1 and Section 3.2.1).
- Empirical results on how the accuracy of the solver gets affected as we vary the weight relaxation parameter (Section 5.1).
- An empirical study of the impact of combining an approximation strategy with complete as well as stochastic approaches (Section 5.2). This is novel to this extended version.
- An implementation of these strategies in Open-WBO-Inc [24] which is built on top of the Open-WBO [33] MaxSAT solver framework. We also demonstrate the advantage of these approximation strategies by showing its prowess against state-of-the-art incomplete MaxSAT solvers (Section 5.3).
- An empirical study of the impact of benchmark characteristics on the accuracy of incomplete approaches (Section 5.4). This is novel to this extended version.
- A discussion on deviation from an optimal solution of these techniques in principle and in practice (Section 5.5). This discussion is an addition to this extended version.

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 α is a disjunction of literals and a formula φ is a conjunction of clauses. Notationally, we will treat a clause α 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 $\nu(x) = \mathit{true}$ (resp. $\nu(x) = \mathit{false}$). For simplicity, we use 1 (resp. 0) and *true* (resp. *false*) interchangeably. 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, while maximizing the number of clauses satisfied in φ_s. Hereafter, we will assume that φ_h is always satisfiable. Let $\mathit{weight} : \mathit{Clauses} \rightarrow \mathbb{N}^+$ 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 α can be relaxed by adding a relaxation variable r so that the relaxed clause becomes $\alpha \cup \{r\}$. 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, $\mathit{cost}(r) = \mathit{weight}(\alpha)$. The cost of an assignment ν is defined as $\mathit{cost}(\nu) = \sum_{r_i:\nu(r_i)=1} \mathit{cost}(r_i)$. The goal of MaxSAT is to find a satisfying assignment with the minimum cost.

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 for the original MaxSAT formula.

3.1 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, Alg. 1 partitions the clauses into clusters as follows. All soft clauses are sorted by their weights (Line 2) in ascending order. 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 divisive clustering is effectively identical to single-link agglomerative clustering [21]. Finally, a new weight map weight_m is created, where all the clauses in the same cluster get the same weight (Lines 11–13). $\mathit{RepresentativeWeight}$ (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 a different effect on how much an algorithm can deviate from finding the minimum

Algorithm 1: Partitioning and weight approximation

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

$n \leftarrow |\varphi_s|$
sort clauses of φ_s in the ascending order of weights
for $i \leftarrow 1$ **to** $n-1$ **do**
  $diff_i \leftarrow weight(\alpha_{i+1}) - weight(\alpha_i)$
$\langle i_1, \ldots, i_{m-1} \rangle \leftarrow$ indices sorted in ascending order where top $(m-1)$ differences $diff_i$ occur
$c_1 \leftarrow \{\alpha_1, \ldots, \alpha_{i_1}\}$
for $j \leftarrow 2$ **to** $m-1$ **do**
  $c_j \leftarrow \{\alpha_{i_{j-1}+1}, \ldots, \alpha_{i_j}\}$
$c_m \leftarrow \{\alpha_{i_{m-1}+1}, \ldots, \alpha_n\}$
$P(m) \leftarrow \{c_1, \ldots, c_m\}$
foreach $c_i \in P(m)$ **do**
  foreach $\alpha_j \in c_i$ **do**
    $weight_m(\alpha_j) \leftarrow RepresentativeWeight(c_i)$
return $\langle P(m), weight_m \rangle$

Algorithm 2: Linear search Sat-Unsat algorithm for MaxSAT

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

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

cost assignment. It is redundant to have $m > \#weights$, where $\#weights$ is the number of distinct weights, because for $m \geq \#weights$, $weight = weight_m$. Alg. 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 [25, 16]. 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 [10, 26] shown in Alg. 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 $cost(\nu)$. Alg. 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 Alg. 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 the model only when a satisfying assignment indeed reduces the previous value of $cost$(model) (Lines 6–7).

3.1.1 Analysis and observations on deviation

In this section we will make some observations on the potential deviation caused due to clustering. It is worth noting that the word approximation used in this paper is in a loose sense of the word and should not be construed in the approximation algorithm sense.

Let ν_{opt} denote a model where $cost(\nu_{opt})$ is the least. Let ν_{clu} denote the model where $cost_m(\nu_{clu})$ is the least. If we had used the minimum weight in the cluster as the representative weight, then we know that for any model ν, $cost_m(\nu) \leq cost(\nu)$. Thus, it would always provide a lower bound. Similarly, if we had used the maximum weight in the cluster as the representative weight, then $cost_m(\nu) \geq cost(\nu)$, thus making it an upper bound. Since we are using the arithmetic mean as the representative weight, for any model ν, $cost_m(\nu)$ may be higher or lower than $cost(\nu)$.

Let the number of clauses in cluster c_i be denoted as $|c_i|$. Let $max(c_i)$ and $min(c_i)$ denote the largest and the smallest weight in the cluster c_i. Also let $top(k, c_i)$ and $bot(k, c_i)$ denote the set of the largest and the smallest k weights in c_i respectively.

Now, let us calculate an upper bound on the deviation $\Delta = |cost(\nu_{clu}) - cost(\nu_{opt})|$. An obvious upper bound on the deviation would be:

$$\Delta \leq n \cdot (max(\varphi_s) - min(\varphi_s)) \tag{1}$$

where, n is the total number of soft clauses $|\varphi_s|$.

Observe that Alg. 2 will be the most precise and complete when $m = \#weights$. On the other hand, it will be the least precise when $m = 1$. Therefore, let us compute an upper bound on the deviation for the worst case when $m = 1$. Note that for $m = 1$, Alg. 2 is just going to satisfy the maximum number of clauses, while completely ignoring their weights. Let $nsat(\nu)$ denote the number of clauses satisfied by an assignment ν. For $m = 1$, it has to be the case that $nsat(\nu_{clu}) \geq nsat(\nu_{opt})$, because the algorithm is just maximizing the number of clauses satisfied. Therefore, for every α_j that ν_{clu} could not satisfy but ν_{opt} could satisfy, there has to be at least one clause α_k that ν_{opt} could not satisfy but ν_{clu} could satisfy. Let us assume that $weight(\alpha_j) > weight(\alpha_k)$. Then, ν_{clu} choosing to satisfy α_k instead of α_j would contribute to an error of $weight(\alpha_j) - weight(\alpha_k)$. Note that ν_{clu} can make such wrong choices for only up to $\frac{n}{2}$ unique pairs, because otherwise $nsat(\nu_{opt}) > nsat(\nu_{clu})$, a contradiction. Therefore, the upper bound can be revised to:

$$\Delta \leq \left(\sum_{t \in top\left(\frac{n}{2}, \varphi_s\right)} t \right) - \left(\sum_{b \in bot\left(\frac{n}{2}, \varphi_s\right)} b \right) \tag{2}$$

Algorithm 3: Layered greedy algorithm for weighted MaxSAT

Input: $\varphi = \varphi_h \cup \varphi_s$, weight maps $weight$ and $weight_m$, Partition $P(m)$
Output: model to φ

1 $(\mathsf{model}, \vec{\mu}, \varphi_W, \mathcal{C}) \leftarrow (\emptyset, +\vec{\infty}, \varphi^r, P_m(\varphi_s))$
2 **foreach** $c_i \in \mathcal{C}$ *in the descending order of* $weight_m(c_i)$ **do**
3 $\varphi_i \leftarrow \varphi_W$
4 $V_i \leftarrow V_R \cap Vars(c_i)$
5 $\mathsf{status} = \mathsf{SAT}$
6 **while** $\mathsf{status} = \mathsf{SAT}$ **do**
7 $(\mathsf{status}, \nu) \leftarrow \mathtt{SAT}(\varphi_i)$
8 **if** $cost(\nu) < cost(\mathsf{model})$ **then**
9 $\mathsf{model} \leftarrow \nu$
10 $\mu_i \leftarrow |\{r \in V_i \mid \nu(r) = 1\}|$
11 $\varphi_i \leftarrow \varphi_i \cup \{\mathtt{CNF}(\sum_{r \in V_i} r \leq \mu_i - 1)\}$
12 $\varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_i} r \leq \mu_i)\}$
13 **return** model

Consider a formula $\varphi = \{(x_1, 100), \ldots, (x_{\frac{n}{2}}, 100), (\neg x_1, 1), \ldots, (\neg x_{\frac{n}{2}}, 1)\}$, where $(\alpha, weight(\alpha))$ denotes a clause α with its corresponding weight $weight(\alpha)$. For $m = 1$, Alg. 2 can satisfy exactly half of the clauses in φ. Since there is no way to prioritize between clauses for $m = 1$, ν_{clu} can set all x_i's to *false* whereas ν_{opt} would set all x_i's to *true*. The bound given in Eqn. (2) would be tight in this case.

3.2 Approximation via subproblem minimization

Alg. 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 cluster c_i (Line 4). Here, $Vars(c_i)$ indicates the set of all the variables that are used in the cluster c_i. Minimization of the cost of a satisfying assignment is divided into subproblems by minimizing the number of unsatisfied clauses in clusters, starting from the highest representative weight to the lowest (Line 2). For each cluster, the number of unsatisfied clauses is 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 and added as an upper bound in the working formula φ_W (Line 12). Thus, φ_W contains cardinality constraints with upper bound μ_j for the number of relaxation variables that can be set to *true* for all the clusters c_j seen so far (Line 12). 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 optimal solution. Note that Alg. 3 can use Alg. 1 as a preprocessing step.

3.2.1 Analysis and observations on deviation

The clustering-based algorithm presented in Alg. 3 is closely related to Boolean Multilevel Optimization (BMO) [30]. BMO is a technique for identifying lexicographic optimization conditions, i.e. the existence of an ordered sequence of objective functions. For this section, let us assume that $m = \#weights$, therefore, $cost$ would be the same as $cost_m$. As a result, if there are multiple clauses having the same weight in the original formula, they will form a cluster due to Alg. 1.

Let w_i be the representative weight of soft clauses in cluster c_i. Consider a sequence of clusters $c_1, c_2, \ldots, c_m$ arranged in a descending order of w_i. A MaxSAT formula is an instance of BMO if for every cluster c_i, w_i is larger than the sum of the weights of all soft clauses in clusters $c_{i+1}, c_{i+2}, \ldots, c_m$, i.e.,

$$\begin{aligned} w_1 &> |c_2|w_2 + |c_3|w_3 + \ldots + |c_m|w_m \\ &\vdots \\ w_{m-2} &> |c_{m-1}|w_{m-1} + |c_m|w_m \\ w_{m-1} &> |c_m|w_m \end{aligned}$$

If the above conditions hold then the result of Alg. 3 is equivalent to solving a BMO formula. Let there be a clause in c_i such that satisfying this would force Alg. 3 to falsify a subset of clauses in $c_{i+1}, \ldots, c_m$. Because of the conditions above, the optimal model will be such that the clause in c_i is preferred over this subset. Hence the model given by Alg. 3 would be optimal in this case.

However, when the clusters and their representative weights do not preserve the BMO condition, it is not guaranteed that the solution found by Alg. 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.

Let ν_{bmo} be the model given by Alg. 3 upon its termination and ν_{opt} be an optimal model for φ. Let $cost(\nu, c)$ denote the sum of weights of the unsatisfied clauses in the set of clauses c by the assignment ν. Let c_k be the first cluster in $c_1, c_2, \ldots, c_m$ such that the condition $w_k > \sum_{i=k+1}^{m} |c_i|w_i$ fails, and $cost(\nu_{bmo}, c_k) \neq cost(\nu_{clu})$. Let Δ_k be the associated error for such a c_k. Since the BMO condition holds for all the clusters $c_1, c_2, \ldots, c_{k-1}$, the cost associated with those clusters will be the same by ν_{bmo} and ν_{opt}. Therefore, for the cluster c_k, $cost(\nu_{bmo}, c_k) < cost(\nu_{opt}, c_k)$. This is because Alg. 3 follows a greedy approach, and will try to satisfy the maximum number of clauses possible in c_k. However, since the BMO condition does not hold for c_k, it might be possible to find a solution with a lower $cost$ which has lesser clauses satisfied in c_k but more clauses satisfied in subsequent clusters.

For example, let $\varphi = \{[(x_1, 100), (x_2, 100)]_1, [(x_3, 10)]_2, [(\neg x_1 \vee \neg x_3, 9), (\neg x_2 \vee \neg x_3, 9)]_3\}$ be defined over a set of Boolean variables $V = \{x_1, x_2, x_3\}$. Here, a tuple $(\alpha, weight(\alpha))$ denotes the clause α and its weight. Also $[c]_i$ denotes the set of clauses that forms the i-th cluster c_i. Here, clearly, the BMO condition is satisfied for the first cluster. So, Alg. 3 will set x_1 and x_2 to *true* as it iterates over these clauses. Next, we observe that the cluster c_2 does not satisfy the BMO condition. This is because the sum of weights of subsequent clauses is $9 + 9 = 18 > 10$, with 10 being the weight of the third clause. Since Alg. 3 follows a greedy approach, it will try to satisfy the third clause, and set x_3 to *true*. This will result

in the fourth and fifth clauses being unsatisfiable. However, the optimal solution involves setting x_3 to *false*, which will fail to satisfy the third clause but will satisfy the fourth and fifth, which gives a better value for *cost*. So, in this example, we have $cost(\nu_{bmo}, c_3) = 0$, but $cost(\nu_{opt}, c_3) = 10$.

Alg. 3 can only deviate from the optimal in terms of satisfied clauses in c_k when there is a set $S \subseteq \bigcup_{i=k+1}^{m} c_i$, such that the sum of weights in S is more than w_k and ν_{opt} satisfies the clauses in S instead of some clause in c_k (which is satisfied by ν_{bmo}). In the worst case scenario, as illustrated above, the set $S = \bigcup_{i=k+1}^{m} c_i$. In this case, the difference in the cost associated with ν_{bmo} and ν_{opt} is upper-bounded as follows:

$$\Delta_k \leq \left(\sum_{i=k+1}^{m} |c_i| w_i \right) - w_k \tag{3}$$

This bound is tight for the example provided above. There, we have $cost(\nu_{bmo}) = 9+9 = 18$, and $cost(\nu_{opt}) = 10$. Using Eqn. 3, we get $\Delta_2 \leq \left(\sum_{i=3}^{3} |c_3| w_3 \right) - w_2 = (2 \times 9) - 10 = 8$.

Even though the BMO condition fails for some c_j, it is possible that $cost(\nu_{bmo}, c_j) = cost(\nu_{opt}, c_j)$. Therefore, Eqn. (3) is applicable only if at cluster c_k BMO condition is violated and k is the least index for which $cost(\nu_{bmo}, c_k) \neq cost(\nu_{opt}, c_k)$. For example, let $V = \{x_1, x_2, x_3, x_4\}$. Consider a formula $\varphi = \{[(x_1, 100), (x_2, 100)]_1\,, [(x_3, 20)]_2\,, [(x_4, 10)]_3\,, [(\neg x_1 \vee \neg x_4, 9), (\neg x_2 \vee \neg x_4, 9)]_4\}$. Here, even though the BMO condition is first violated in c_2, we have $cost(\nu_{bmo}, c_2) = cost(\nu_{opt}, c_2) = 0$, since x_3 can always be satisfied. The first cluster c_k such that $cost(\nu_{bmo}, c_k) \neq cost(\nu_{opt}, c_k)$ is c_3. Using similar arguments from above, we can see that the bound in Eqn. (3) is tight for this example.

In general, the first cluster c_k for which BMO condition is violated and $cost(\nu_{bmo}, c_k) \neq cost(\nu_{opt}, c_k)$ it has to be the case that $cost(\nu_{bmo}, c_k) < cost(\nu_{opt}, c_k)$.

Eqn. (3) still relies on the knowledge ν_{opt}. So let us try to derive a bound that does not rely on the knowledge of ν_{opt}. Let $c_{k_1}, \ldots, c_{k_p}$ be the set of clusters where BMO condition is violated where $k_1 < \cdots < k_p$. Remember that $w_{k_1} > \cdots > w_{k_p}$. As we do not know which is going to be the first c_{k_i} for which $cost(\nu_{bmo}, c_{k_i}) \neq cost(\nu_{opt}, c_{k_i})$, we can over-approximate Eqn. (3) to obtain the deviation upper bound as follows:

$$\Delta \leq \left(\left(\max_{j \in \{k_1, \ldots, k_p\}} \left(\sum_{i=j+1}^{m} |c_i| w_i \right) \right) - \min(w_{k_1}, \ldots, w_{k_p}) \right) \leq \left(\sum_{i=k_1+1}^{m} |c_i| w_i - w_{k_p} \right) \tag{4}$$

Note that Eqn. (4) does not depend on the knowledge of ν_{opt} and can be computed just based on the given input formula.

Adversarial Example

Here, we show an example where assuming the BMO condition leads to a poor approximation. Consider a SAT formula φ in CNF form, with N variables. Let the set of variables be V, where $|V| = N$. A clause $\alpha \in \varphi$ evaluates to *false* if each literal in α is *false*. Suppose there exists a partial assignment for φ, where $0 < k \leq N$ variables have been assigned to either *true* or *false*. Let the set of these variables be $K \subseteq V$. Let C be the set of clauses falsified by this assignment. Then, C can contain at most $2^k - 1$ clauses. This is because in

any such clause $\alpha \in C$, for each $v \in K$, there are two possibilities: (i) the literal containing v has a polarity opposite to that of the assignment, or (ii) v does not appear in α. From the resulting 2^k possibilities for α, we ignore the case where the clause is empty. Note that no variable $v \in V \setminus K$ can appear in such a α, since it can be made to evaluate to *true* by appropriately setting the truth value of v. Another observation is that if the assignment of each variable in K is flipped, then every clause in C would be satisfied.

This shows that with a poor assignment, it is possible to falsify a set of clauses that is exponential to the number of variables in the formula. Since the Alg. 3 greedily attempts to satisfy groups of clauses, and fixes the upper bound on the number of unsatisfied clauses in all the preceding clusters, this unravels the possibility of constructing an adversarial formula that will cause Alg. 3 to provide such an assignment.

We now construct such a formula φ. We first provide a concrete example with three variables, after which we describe a procedure for this construction for N variables.

An example with three variables
Let $V = \{x_1, x_2, x_3\}$. Let a $(\alpha, weight(\alpha))$ denote a clause α and its weight. Then, define $\varphi' = \{(x_1, 4), (\neg x_1, 3), (x_2, 3), (\neg x_2, 2), (\neg x_1 \vee \neg x_2, 2), (x_3, 2), (\neg x_3, 1), (\neg x_1 \vee \neg x_3, 1), (\neg x_2 \vee \neg x_3, 1), (\neg x_1 \vee \neg x_2 \vee \neg x_3, 1)\}$. Then, the Alg. 3 works as follows:

1. The cluster with the largest weight is $\{(x_1, 4)\}$. The maximum number of clauses that can be satisfied in this cluster is 1, which can be done by setting $x_1 = \mathit{true}$. This assignment is effectively frozen before moving to the next cluster, since the upper bound on the number of relaxation variables that can be set to *true* is fixed to 0.

2. The next cluster, based on descending order of weights, is $\{(\neg x_1, 3), (x_2, 3)\}$. Since $x_1 = \mathit{true}$ is frozen, $(\neg x_1, 3)$ must remain unsatisfied. By setting $x_2 = \mathit{true}$ the upper bound on the unsatisfied clauses in this cluster can be fixed to 1. This effectively freezes $x_2 = \mathit{true}$.

3. Similarly, one can verify that $x_3 = \mathit{true}$ will be the best assignment found in the third cluster, while no clause can be satisfied in the fourth cluster.

As a result, the cost of this assignment, ν_{bmo}, is $cost(\nu_{bmo}) = (3) + 2(2) + 4(1) = 11$. However, an optimal solution, ν_{opt} would be to set all except one variable in V to be *false*, with any one of them being *true*, leading to a cost of $cost(\nu_{opt}) = 8$.

Constructing φ for N variables
Let $V = \{x_1, x_2, \ldots, x_N\}$. We add clauses iteratively to φ as follows:

1. In the first step $(i = 1)$, $(x_1, N + 1)$ is added to φ.

2. We then iterate for the next $N - 1$ steps (from $i = 2$ to $i = N$). In the i^{th} step, $2^{i-2} + 1$ clauses are added.

   (a) First, $(x_i, N - i + 2)$ is added.

   (b) Here, we consider two possible cases, either of which adds 2^{i-2} clauses:

      i. If $i > 2$, then every possible clause α of the form $(\{\neg x_{i-1}\} \cup E, N - i + 2)$ such that $E \in \mathcal{P}(\{\neg x_1, \ldots, \neg x_{i-2}\})$, with $\mathcal{P}$ being the power set of its argument, is added.

ii. If $i = 2$, a single clause, $(\neg x_1, N)$ is added.

3. In the final, $(N+1)^{th}$ step, we again consider two possible cases, either of which adds 2^{N-1} clauses:
   (a) If $N > 1$, every possible clause α of the form $(\{\neg x_N\} \cup E, 1)$ such that $E \in \mathcal{P}(\{\neg x_1, \ldots, \neg x_{N-1}\})$ is added.
   (b) If $N = 1$, a single clause, $(\neg x_N, 1)$ is added.

It can be observed that the above procedure generates φ' in the example above.

Using this procedure, an adversarial formula φ can be constructed such that setting all its variables except one to *false* will provide the optimal cost. In such an assignment all but one clause among clauses of the form $(x_i, N-i+2), 1 \leq i \leq N$ would remain unsatisfied. Without loss of generality, let ν_{opt} have $x_j = \mathit{true}$. Then $(x_j, N-j+2)$ is satisfied and $(\neg x_j, N-j+1)$ would remain unsatisfied. Thus, irrespective of j, it reduces the cost by 1. All other clauses are satisfied. Such an assignment ν_{opt}, for φ with N variables, gives

$$cost(\nu_{opt}) = \left(\sum_{i=1}^{N}(N-i+2)\right) - 1 = \frac{(N+1)(N+2)}{2} - 2$$

However, because Alg. 3 sets each variable to *true*, an exponential number of clauses are unsatisfied. It can be verified that the predicted optimal cost of the assignment ν_{bmo} found by the algorithm in this case would be

$$cost(\nu_{bmo}) = \sum_{i=2}^{N+1} 2^{i-2}(N-i+2) = 2^{N+1} - N - 2$$

The error, Δ, is thus,

$$\Delta = 2^{N+1} - N - \frac{(N+1)(N+2)}{2} = 2^{N+1} - \frac{N^2+5N+2}{2} \tag{5}$$

The above construction shows that there exist formulas for which the deviation of the optimal found by Alg. 3 from the true optimal is exponential in the number of variables.

It can also be seen that the error bound in Eqn. (5) satisfies Eqn. (4). For $N = 1$, BMO condition will hold. By definition, BMO condition can not be violated for the last cluster because there are no more clusters after that. For $N > 1$, the BMO assumption is violated in every cluster but the last one in the above construction. Then, according to Eqn. (4), we have,

$$\begin{aligned}
\Delta &\leq \sum_{i=2}^{N+1} |c_i| w_i - w_N \\
&= \sum_{i=2}^{N} \underbrace{(2^{i-2}+1)}_{\text{\#clauses from Step-2}} \underbrace{(N-i+2)}_{\text{weight in Step-2}} + \underbrace{2^{N-1}}_{\text{\#clauses from Step-3}} - \underbrace{2}_{\text{weight of last cluster where the BMO condition fails}} \\
&= 2^{N+1} + \frac{N^2-N-10}{2}
\end{aligned}$$

Clearly, for $N > 1$, we have $2^{N+1} + \frac{N^2-N-10}{2} > 2^{N+1} - \frac{N^2+5N+2}{2}$, and hence, the error in Eqn. (5) satisfies Eqn. (4).

The discussion so far in this section was based on the assumption that $m = \#weights$. In this case, the deviation can only happen because of BMO condition violation. If $m \neq \#weights$, the error because of clustering would also contribute to the deviation. Theoretical bound in the deviation is likely to be even worse than what is given in Eqn. (4).

4. Related Work

Approaches for incomplete MaxSAT solving can primarily be divided into three categories: (i) stochastic MaxSAT solvers [11, 29, 28, 12, 17, 27], (ii) complete MaxSAT solvers that can find intermediate solutions [26, 10, 33, 4, 13, 37], and (iii) approximation approaches that may not guarantee optimality [15, 14, 35, 31].

4.1 Stochastic MaxSAT

Stochastic solvers start by finding a random assignment ν for φ. Since this assignment is unlikely to satisfy all the clauses in φ, they choose a clause α_i that is unsatisfied by ν and flip the assignment of a variable in α_i such that α_i becomes satisfied. When compared to local search 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 it. In the meantime, there has been a constant attempt to close the gap between the performance of stochastic solvers and complete solvers for industrial benchmarks. In the MSE2018, there was one submission of a stochastic solver called SATLike [27]. This solver uses a dynamic local search framework that modifies the weights of clauses during the search and exploits the distinction of hard and soft clauses by using a particular weighting scheme. For the MSE2018, the authors from SATLike also submitted a hybrid version that switches from a stochastic algorithm to the Sat-Unsat complete MaxSAT algorithm described in Alg. 2. This version outperformed the pure stochastic version and was the winner of the MSE2018 for incomplete MaxSAT on unweighted problems for both 60 and 300 seconds time limits. However, for weighted problems its performance was subpar. The evaluation conducted in this paper focuses on weighted problems and we restrict ourselves to the stochastic version of SATLike. Even though this version is not competitive with the other approaches it uses a different algorithm that may be competitive for a restrictive set of benchmarks.

4.2 Complete MaxSAT

Complete solvers can often find intermediate solutions to φ before finding an optimal assignment ν. MaxSAT solvers based on linear search algorithms [10, 26, 33] can find a sequence of intermediate solutions that converge to an optimal solution. These solvers use PB constraints to enforce convergence. While SAT4J [10] uses specialized data structures for PB constraints to avoid their conversion to CNF, other solvers such as QMaxSAT [26]

convert the PB constraint into clauses using PB encodings [40, 25, 16]. Some MaxSAT solvers which are based on the implicit hitting set approach [13, 37] 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 [34, 4, 1]. 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 unsatisfiability-based solver that can be used for incomplete MaxSAT. Besides the stratified approach, this solver is also able to find intermediate solutions due to an optimization step that tries to further increase the lower bound value at each iteration. WPM3 extends phase saving [4] for MaxSAT to guide the search towards the last assignment found by the solver and was the best incomplete MaxSAT solver in the MSE2016 [6]. maxroster [39] 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 [17] with complete MaxSAT algorithms [34, 26].

4.3 Approximation approaches

Approximation techniques do not necessarily guarantee optimality on the final solution. An example of such an approach is the enumeration of Minimal Correction Subsets (MCSes) [31]. An MCS of an unsatisfiable set of constraints is a minimal subset that, if removed, makes the constraint set satisfiable. MCSes can be used to approximate MaxSAT solutions [31] and a version of Open-WBO (called Open-WBO-Inc-MCS) was submitted to the MSE2018 incomplete track for unweighted benchmarks using this approach. There has been a recent trend of using approximation approaches for incomplete MaxSAT. Recently, an approach based on bit-vector optimization was proposed for unweighted incomplete MaxSAT with promising results [35]. A similar strategy to this approach is implemented on top of Open-WBO (called Open-WBO-Inc-OBV) and submitted to the unweighted incomplete track of the MSE2018. Note that none of these algorithms performed exceptionally well and were only submitted to the unweighted track which is beyond the scope of this paper.

LinSBPS [15] is a new incomplete MaxSAT solver that also uses an approximation approach that closely resembles the techniques presented in this paper. For weighted instances, they propose to see the formula in *low resolution*, i.e. with all their weights divided by a large value. Note that they use integer division which sets some soft clauses to weight zero, effectively removing them. Whenever the solver terminates with an "optimal" solution to the simplified formula, the resolution is increased by decreasing the value with which they divide all weights. This varying resolution approach resembles our weight approximation approach. Instead of dividing all weights by a given constant, we group the weights into m-partitions which can be efficiently encoded into CNF with the Generalized Totalizer Encoding (GTE) [25], the complexity of which depends on the number of unique weights. On the other hand, LinSBPS keeps more information from the original formula by maintaining the ratio between weights and is able to dynamically increase the resolution to improve their

accuracy over time. Another optimization employed by LinSBPS is the use of solution-based saving [4, 14] for both unweighted and weighted problems to guide the search towards the best solution found so far. LinSBPS was the winner of the weighted incomplete track in the MSE2018 with a time limit of 300 seconds. For the weighted incomplete track in the MSE2018 with a time limit of 60 seconds, the winner was the clustering-based algorithm presented in Alg. 3. In the next section, we present a thorough comparison between our techniques and the state-of-the-art in incomplete MaxSAT solving.

5. Experimental Results

To evaluate incomplete MaxSAT solvers we used the scoring scheme from the previous MaxSAT Evaluations of 2017 [2] (MSE2017) and 2018 [8] (MSE2018). 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 $best(\varphi)$, [1.] to the best cost found by $\mathcal{S}$, denoted as $cost^{\mathcal{S}}(\varphi)$. [2.] The score for $\mathcal{S}$ for a set of n benchmarks is given by the average score ($[0, 1]$) as follows:

$$\mathsf{score}(\mathcal{S}) = \frac{\sum_{i=1}^{n} \frac{best(\varphi_i)}{cost^{\mathcal{S}}(\varphi_i)}}{n} \tag{6}$$

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

All the experiments were conducted on StarExec [38] using Intel® Xeon® E5-2609 processors (2.40GHz) with a memory limit of 32GB and time limits of 10, 60 and 300 seconds. We have used a non-standard timeout of 10 seconds to demonstrate that the approximation strategies can find good solutions very quickly. As our benchmark set, we used the 156 benchmarks for incomplete MaxSAT from the MSE2017 [2] and 172 benchmarks for incomplete MaxSAT from the MSE2018 [8] for a total of 240 benchmarks (benchmarks that appear in both the sets are counted only once).

Note that most of these benchmarks are challenging for complete solvers and have unknown optimal solutions. To improve the best known solution $best(\varphi)$ for every benchmark, we also ran RC2 [20, 19], the best complete solver in the MSE2018 for the complete category, with 1800 seconds timeout. We have implemented all the algorithms presented in this paper in Open-WBO-Inc [24]. Open-WBO-Inc is built on top of Open-WBO [33] which uses Glucose [7] as the underlying SAT solver. We used Generalized Totalizer Encoding (GTE) [25] and incremental Totalizer encoding [32] to translate PB constraints and cardinality constraints into CNF, respectively.

To evaluate Open-WBO-Inc, we performed an extensive experimental evaluation which aims to answer the following questions:

Q1. What is the impact of the number of clusters on the accuracy of Open-WBO-Inc? How does the accuracy of Open-WBO-Inc improve over time?

Q2. Can the accuracy of Open-WBO-Inc be further improved when combined with other MaxSAT solvers?

1. $best(\varphi)$ is the cost of the best solution found by any solver in this evaluation.
2. We consider a score of 0 if $\mathcal{S}$ did not find any solution to φ.

Q3. How does Open-WBO-Inc compare against the state-of-the-art in incomplete MaxSAT solving?

Q4. What is the impact of the number of different weights of soft clauses on the accuracy of Open-WBO-Inc and other state-of-the-art incomplete MaxSAT solvers?

Q5. What is the practical and theoretical deviation in score of Open-WBO-Inc on benchmarks with known optimal value?

5.1 Impact of the number of clusters and analysis of accuracy over time

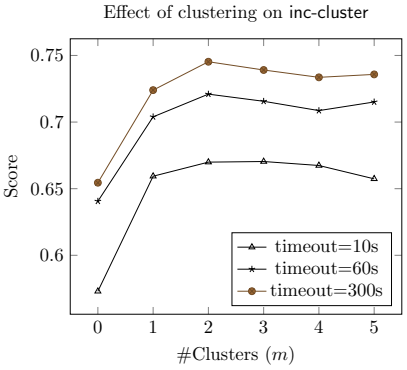
We measure the impact of the partitioning parameter m on the accuracy of the results. Fig. 1 shows how the accuracy of the results is affected as the partitioning parameter m varies. Fig. 1a shows the score of Alg. 2 with GTE encoding (henceforth called inc-cluster) when using the MSE2017 benchmarks and Fig. 1c when using the MSE2018 benchmarks. We can observe a similar trend on the union of the MSE2017 and MSE2018 benchmarks and Fig. 1e shows the overall results on the union of both the benchmark sets. These figures show that inc-cluster performs the worst when no partitioning is done. This is attributed to the fact that the size of the formula in GTE is dictated by $\#weights$ [25], where $\#weights$ are the number of different weights in the weight map. Since Alg. 1 changes weights of all the clauses in a cluster c_i to $RepresentativeWeight(c_i)$, it effectively reduces $\#weights$. All three figures, Fig. 1a, Fig. 1c, and Fig. 1e shows that as m increases the, the possible deviation from an optimal cost also decreases, thereby resulting in increased scores. The degradation for larger m is attributed to a larger size of the formula. As the timeout is increased, the score increases because Alg. 2 has more time and can do more iterations to reduce $cost_m(\mathsf{model})$.

Since inc-cluster performs the best with $m = 3$, as observed in Fig. 1e, we will use this configuration in the remainder of the experimental evaluation.

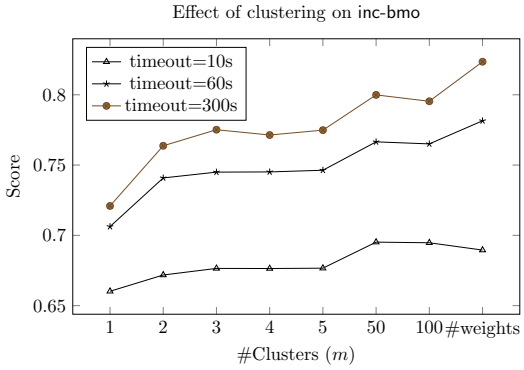
Fig. 1b and Fig. 1d show that similar scoring trends are witnessed for Alg. 3 (henceforth called inc-bmo) when using the MSE2017 benchmarks and the MSE2018 benchmarks, respectively. As inc-bmo uses only cardinality constraints, the formula size is not very sensitive to m. As m increases, the scores also improve due to increased precision, with the best scores achieved when $m = \#weights$. For the remainder of the experimental evaluation, we will consider inc-bmo with $m = \#weights$ since this is when it performed the best. inc-bmo is guaranteed to find an optimal solution only if the BMO condition holds and $m = \#weights$. However, only 3 out of 156 benchmarks in the MSE2017 satisfy the BMO condition and the inc-bmo algorithm does not terminate for any of them. None of the 172 benchmarks in the MSE2018 satisfy the BMO condition.

5.2 Combining Open-WBO-Inc with other MaxSAT algorithms

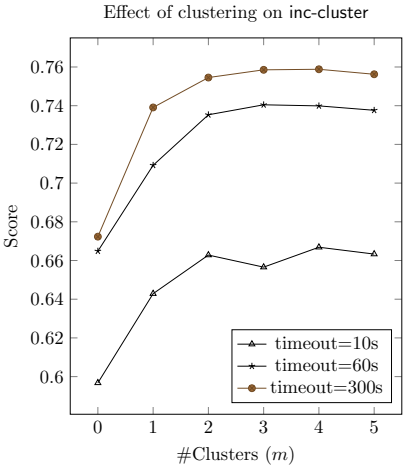
It is possible for either inc-cluster or inc-bmo to terminate before the timeout is reached. Since these algorithms are not guaranteed to find an optimal solution to the original MaxSAT problem, they can terminate with a suboptimal solution before the time limit is reached. Tab. 1 shows how often inc-cluster and inc-bmo terminate before reaching the time limit. We can observe that inc-cluster rarely terminates before the time limit and therefore is unlikely to be further improved by extending it with a different MaxSAT algo-



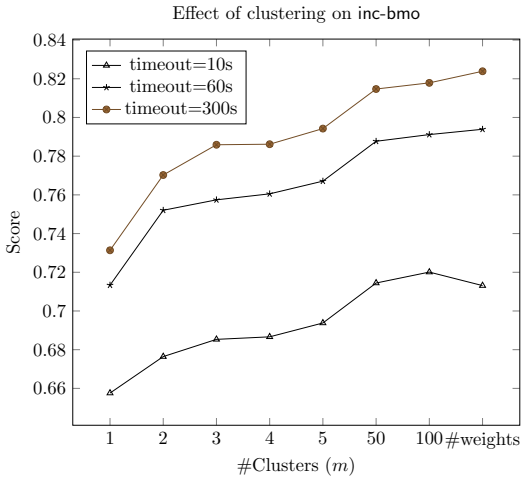
(a) inc-cluster Cluster v/s Score MSE2017



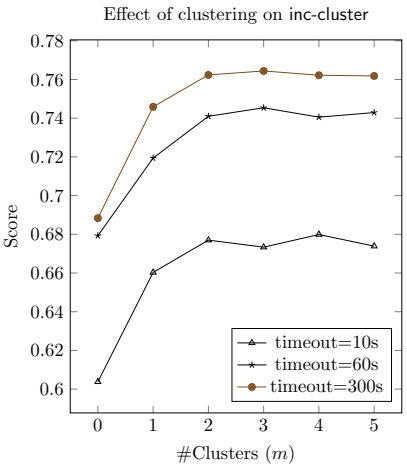
(b) inc-bmo Cluster v/s Score MSE2017



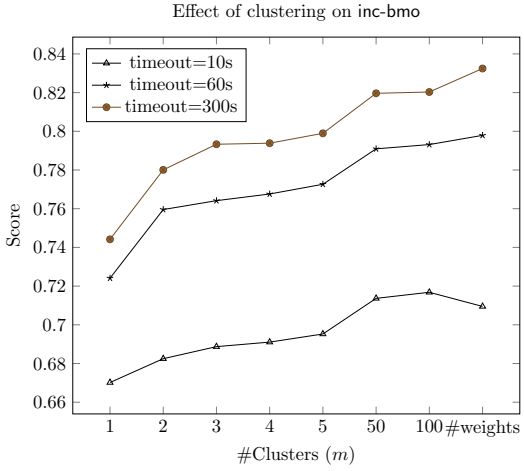
(c) inc-cluster Cluster v/s Score MSE2018



(d) inc-bmo Cluster v/s Score MSE2018



(e) inc-cluster Cluster v/s Score MSE2017 ∪ MSE2018



(f) inc-bmo Cluster v/s Score MSE2017 ∪ MSE2018

Figure 1: Impact of clustering and analysis of accuracy over time

Table 1: Number of benchmarks where inc-cluster and inc-bmo terminated early

Benchmark	#	Solver					
		inc-cluster			inc-bmo		
		10s	60s	300s	10s	60s	300s
MSE2017	156	4	4	5	54	74	94
MSE2018	172	20	22	23	82	102	115
All	240	21	23	25	108	136	157

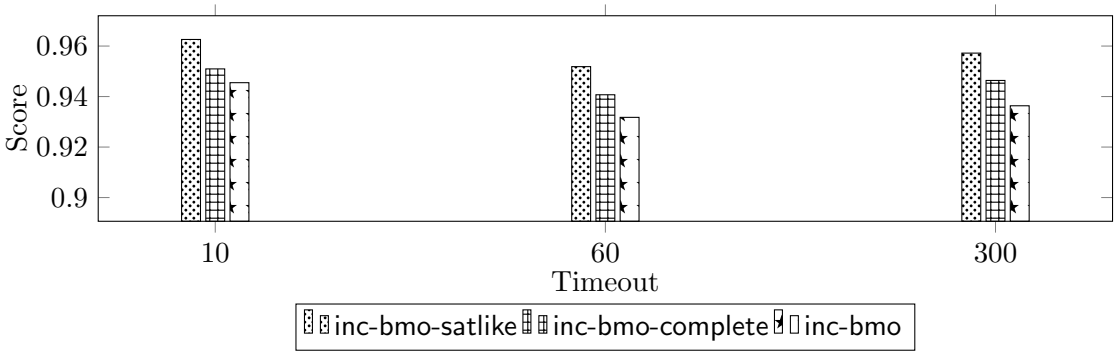


Figure 2: Impact of inc-bmo-complete and inc-bmo-satlike on benchmarks where inc-bmo terminates, considered separately for each time limit

rithm. However, this is not the case for inc-bmo where it reaches early termination on 157 out of 240 benchmarks when using a time limit of 300 seconds. For those benchmarks, we could change the configuration of inc-bmo or dynamically change to a different algorithm that could attempt to further improve the best solution found so far.

Complete MaxSAT algorithms. One possible extension to the inc-bmo algorithm, is to change to a complete MaxSAT algorithm that can find intermediate solutions such as the one presented in Algorithm 2. This was the approach taken by Open-WBO-Inc in the MSE2018. Once inc-bmo terminates, it dynamically changes to the linear Sat-Unsat algorithm presented in Alg. 2. This combination of inc-bmo switching to a complete algorithm will be referred to as inc-bmo-complete. This transformation is done incrementally (i.e., without destroying the SAT solver) and preserving the best known upper bound. Open-WBO-Inc uses the GTE encoding [25] by default unless the number of auxiliary clauses created by the GTE encoding exceeds 3,000,000. In this case, Open-WBO-Inc uses the non-arc consistent Adder encoding [40] that has a linear complexity with respect to the number of auxiliary variables and clauses and reduces the chance of Open-WBO-Inc reaching the memory limit.

Stochastic MaxSAT algorithms. Another alternative is to change to a stochastic MaxSAT algorithm that may be able to improve the best known upper bound since it does not rely on SAT solvers and uses a very different approach. To this end, we modified the stochastic MaxSAT solver SATLike to use the best known solution from inc-bmo as its starting point. Henceforth, we will refer to this combination of switching from inc-bmo to SATLike as inc-bmo-satlike.

Fig. 2 shows the impact of changing algorithms once inc-bmo terminates early on the combined set of benchmarks from the MSE2017 and MSE2018. Note that for the benchmarks where inc-bmo did not terminate early the performance remains the same since the algorithm never changed. It is worth noting that for the 10 seconds plot in Fig. 2, only those subset of benchmarks are considered for which inc-bmo terminated earlier than 10 seconds, and similarly for the other two time limits. Therefore, the subset of benchmarks considered for each time limit is different, which changes the denominator in Eqn. (6). This is the reason for higher scores. Also, the score for 10 seconds for a solver configuration may be higher than the score for 60 seconds purely because of the increase in the denominator in Eqn. (6).

Fig. 2 shows that after inc-bmo terminates, it is more advantageous to switch to a stochastic solving strategy as compared to switching to a complete algorithm. inc-bmo-complete improves the average score of inc-bmo from 0.9363 to 0.9464 whereas inc-bmo-satlike improves the average score of inc-bmo from 0.9363 to 0.9572 for 300 seconds time limit. We observe similar trends across other time limits as well.

Note that even though we showcase that the accuracy of inc-bmo could be further improved by changing algorithms, the approach is modular and is not restricted to the ones presented in this section. Any solver that can take advantage of an upper bound would benefit from this approach and inc-bmo could even be used as a preprocessing technique to find initial upper bounds for weighted MaxSAT instances.

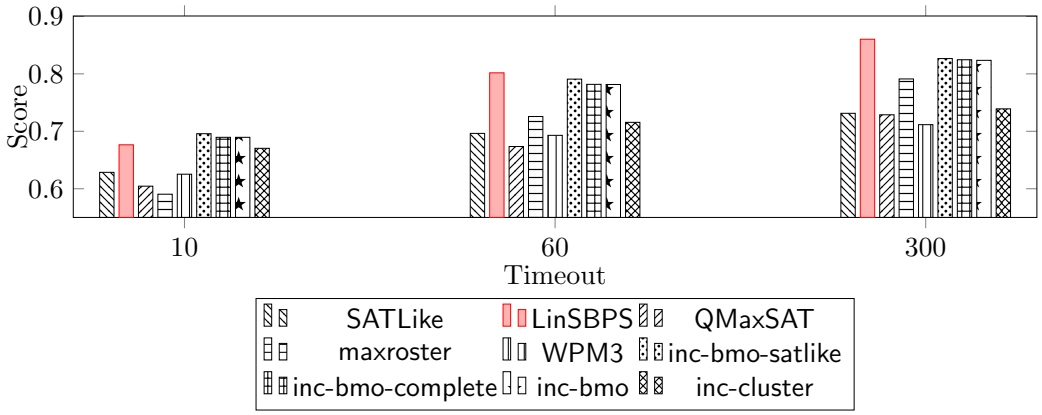
5.3 Comparison against state-of-the-art incomplete MaxSAT solvers

We compared the best version of Open-WBO-Inc for weight-based approximation, inc-cluster with $m = 3$, and subproblem minimization, inc-bmo with $m = \#weights$ in three modes: (i) without switching (inc-bmo) (ii) switching to a complete algorithm (inc-bmo-complete), and (iii) switching to SATLike (inc-bmo-satlike), with the best performing incomplete MaxSAT solvers from the MSE2017 and MSE2018. Specifically, maxroster [39], WPM3 [4], QMaxSAT [26], LinSBPS [15] and SATLike [27] were used for comparison. 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 Alg. 2 [3]. LinSBPS was the winner of the incomplete weighted category for 300 seconds and SATLike was the winner of the incomplete unweighted category.

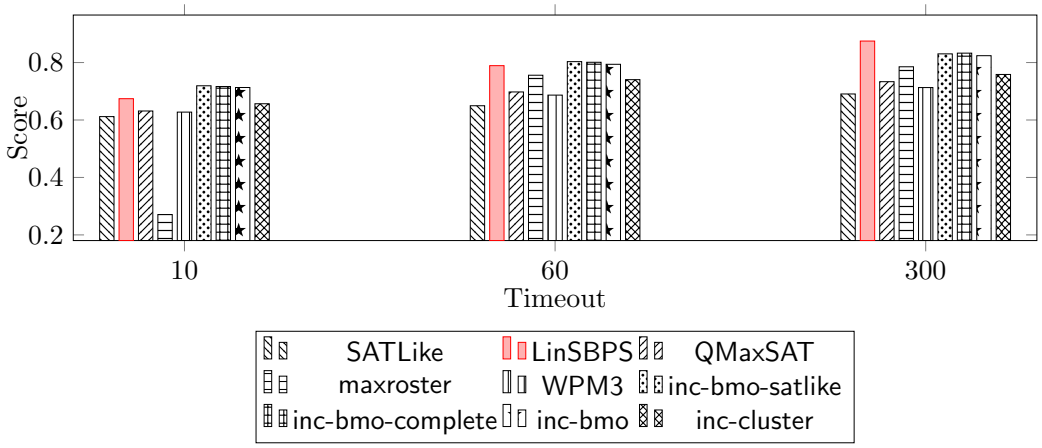
As shown in Fig. 3, for a 10 seconds timeout, all three modes of inc-bmo outperform all other approaches with LinSBPS coming as a close second and inc-cluster placing third. This demonstrates that approximation strategies are quite effective when we want to quickly find a solution which is close to an optimal solution.

As time increases, LinSBPS takes the lead with all three modes of inc-bmo being a close second. LinSBPS performs a similar approximation strategy to inc-cluster where weights are relaxed. However, instead of clustering the weights, LinSBPS divides them by a large value and decreases this value once the simplified problem is solved optimally. This contrasts

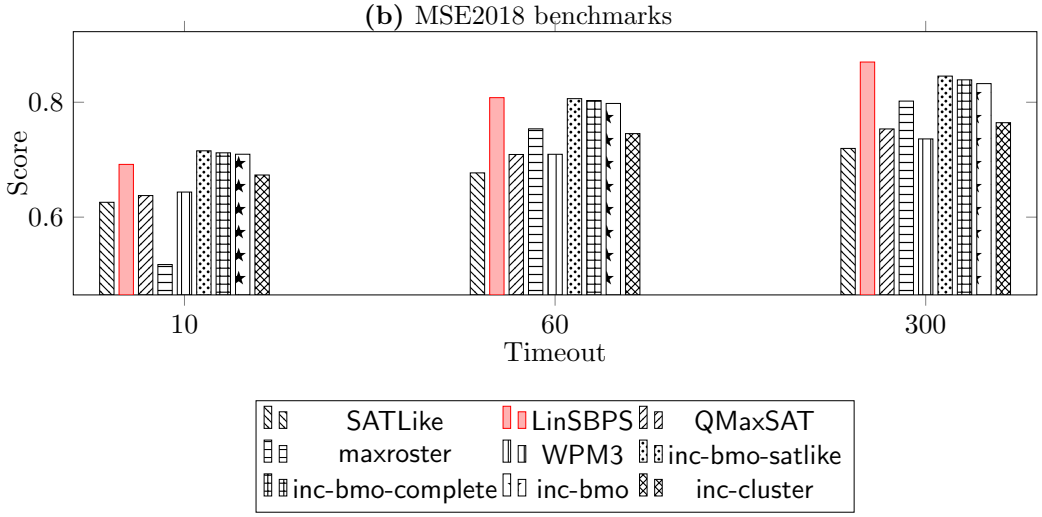
3. Even though MaxHS [13] placed first in the complete weighted category of the MSE2017, its incomplete version is not as competitive [2]. In the MSE2018, Pacose [36] was another solver based on Alg. 2 but its performance is similar to QMaxSAT (only solved more 9 benchmarks than QMaxSAT).



(a) MSE2017 benchmarks



(b) MSE2018 benchmarks



(c) MSE2017 and MSE2018 benchmarks

Figure 3: Comparison against state-of-the-art

Table 2: Number of benchmarks having a given range of unique weights

Range	MSE2017	MSE2018	Combined
1-5	61	63	93
6-10	15	30	37
11-50	6	8	8
51-200	6	24	27
201-500	26	17	28
501-1000	9	9	12
1001-10000	29	20	31
10001-∞	4	1	4
Total	156	172	240

Table 3: Distribution of weights across all benchmarks

	Mean	Std. Dev.	Median
Min	1	0	1
1st Quartile	5.659	2.969	3
Median	237.872	217.095	225
3rd Quartile	30,062.163	8094.713	27410.250
Max	6,779,863,060	54,627,561,614	2,082,335,888

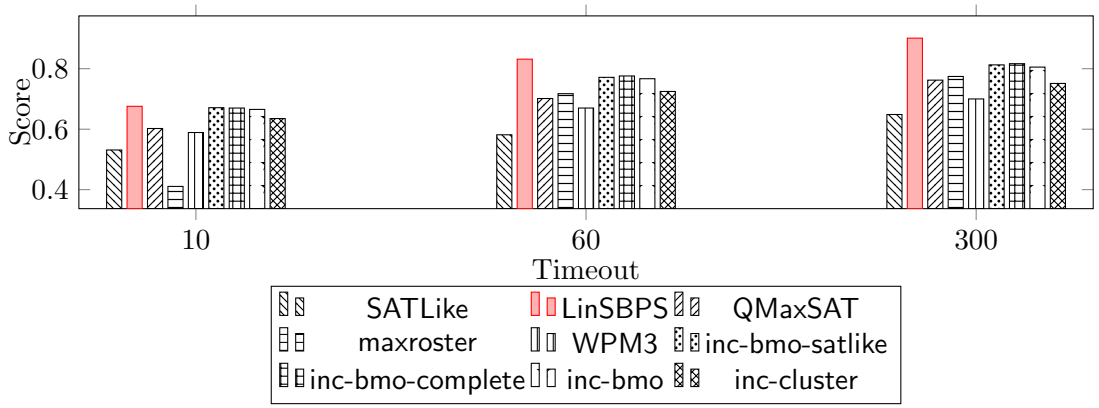
with inc-bmo-complete or inc-bmo-satlike where once we find an optimal solution, we switch to a complete algorithm or we switch to SATLike based local search algorithm respectively.

Even though inc-cluster with $m = 3$ performs worse than QMaxSAT for the 60 seconds and 300 seconds timeout, it outperforms QMaxSAT for the 10 seconds timeout. Overall, inc-cluster is the fourth best solver behind LinSBPS, inc-bmo, and maxroster.

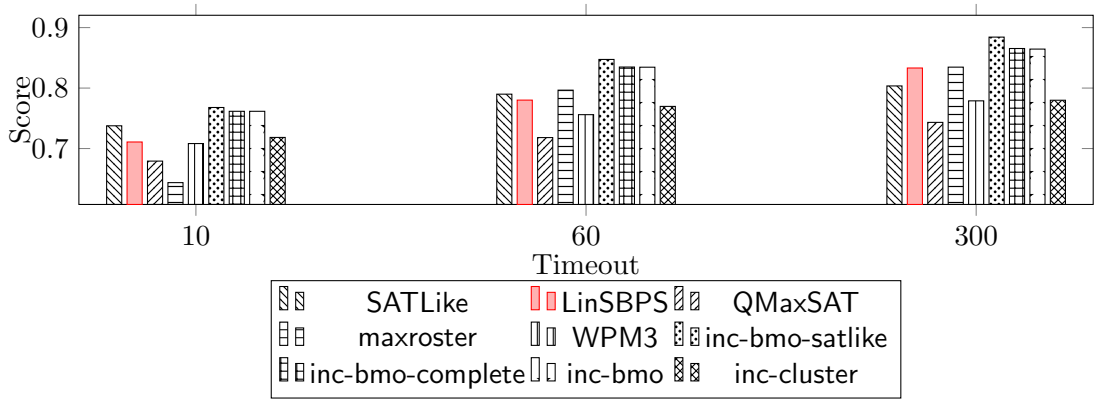
It is worth noting that inc-bmo-satlike and inc-bmo-complete secured third and fourth place respectively in the MaxSAT Evaluation 2019 (MSE2019) [9] ahead of LinSBPS in the incomplete weighted category for 60 seconds timeout. Additionally, in the same category with 300 seconds timeout, inc-bmo-satlike and inc-bmo-complete secured third and fifth place respectively with LinSBPS securing fourth place. These results further validate our claims regarding the performance of inc-bmo-satlike and inc-bmo-complete.

5.4 Impact of benchmark characteristics on solver accuracy

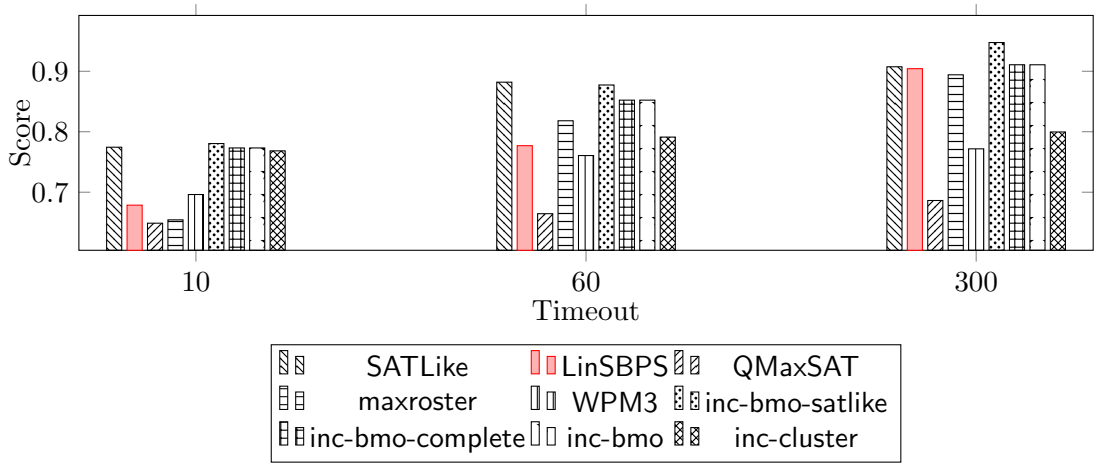
Tab. 2 shows the distribution of soft clauses with unique weights among the benchmarks from the MSE2017 and MSE2018. We can observe a split between benchmarks with a few distinct weights (around 54% with 10 or less distinct weights) and benchmarks with many distinct weights (46% with more than 10 distinct weights, from which around 43% have more than 500 distinct weights). Tab. 3 shows the statistics about the distribution of the weights in the benchmarks. For all the 240 benchmarks, we computed arithmetic mean, median and standard deviation of the weights. An entry (row, col) in the table represents the statistical properties of the weights of the benchmarks. For example, 217.095 in row *Median* and column *Std. Dev.* represents the median of the standard deviations of all 240



(a) Benchmarks from MSE2017 and MSE2018 with ≤ 10 unique weights



(b) Benchmarks from MSE2017 and MSE2018 with > 10 unique weights



(c) Benchmarks from MSE2017 and MSE2018 with > 500 unique weights

Figure 4: Impact of distinct weights on the accuracy of MaxSAT solvers

benchmarks. Tab. 3 shows that though for the most benchmarks the magnitudes of the weights, as well as the spread of weights, is low, there are about 25% benchmarks where the magnitudes are very high (*3rd Quartile,Median*) as well as the spread of the weights is also very high (*3rd Quartile, Std. Dev.*)

To analyze the impact of unique weights on the accuracy of incomplete MaxSAT solvers, we considered three distinct partitions: (i) benchmarks with 10 or less unique weights (130 benchmarks), (ii) benchmarks with more than 10 unique weights and (110 benchmarks), and (iii) benchmarks with more than 500 unique weights (47 benchmarks).

Fig. 4a shows the variation of accuracy on benchmarks with 10 or less unique weights. LinSBPS clearly outperforms all other approaches when solving benchmarks with a few unique weights. However, when analyzing benchmarks with more than 10 unique weights the picture changes drastically and inc-bmo becomes much better than LinSBPS as can be seen in Fig. 4b. We conjecture that the continuous decrease on the weight relaxation performed by LinSBPS performs better when the number of unique weights is smaller. Note that LinSBPS still outperforms inc-cluster approach which clusters the weights into m partitions.

Fig. 4c shows the accuracy on the subset of benchmarks that have more than 500 unique weights. SATLike performs much better when the number of unique weights is large. Since the stochastic algorithm of SATLike does not depend on the number of different unique weights (or on its magnitude), the solver shows a remarkable performance for this kind of benchmarks. However, for 300 seconds we can see that inc-bmo still outperforms SATLike even though the accuracy is comparable. When combining inc-bmo with SATLike, we can observe that inc-bmo-satlike further improves the accuracy of inc-bmo and clearly outperforms all other solvers for 300 seconds when the number of unique weights is large.

5.5 Practical and theoretical deviations

Tab. 4 shows the accuracy of Open-WBO-Inc on benchmarks where the optimal value is known (which corresponds to 80 out of 240 benchmarks) using the scoring metric computed by equation Eqn. (6). The column *Practice* indicates the score observed by running approximation strategies with the best performing parameters. That is, $m = 3$ for inc-cluster and $m = \#weights$ for inc-bmo. Column *Theory* denotes the scores obtained with respect to the theoretical deviation as provided in Eqn. (2) ($m = 1$) for inc-cluster and Eqn. (4) ($m = \#weights$) for inc-bmo.

We can observe that the worst case scenario modeled by the theoretical bounds does not happen in practice. Therefore, the accuracy of inc-cluster and inc-bmo are much higher in practice than their theoretical counterparts.

Notice that the theoretical accuracy for inc-bmo is far worse than the one for inc-cluster. This is due to the theoretical error for inc-bmo is proportional to the entire sum of the weights beneath the first violation, and in practice, such violation may occur very early, therefore, producing a large theoretical error. In inc-cluster, the theoretical accuracy is better than inc-bmo since at most 50% of clauses can contribute to the error, whereas in inc-bmo the percentage of clauses contributing to error can go arbitrarily high.

Table 4: Comparison between theoretical and practical accuracy of inc-bmo and inc-cluster

Practice		Theory	
inc-cluster	inc-bmo	inc-cluster	inc-bmo
0.798	0.867	0.256	0.180

6. Summary

In this paper, we have shown that relatively simple ideas such as clustering and a greedy approach in subproblem minimization result in a significant improvement towards incomplete weighted MaxSAT solving. We analyzed how much deviation can be caused in principle by these approximation strategies under various assumptions. This paper also demonstrates that the approaches proposed here achieve their purpose of finding a good solution in a short time frame and can be quite competitive to the best of the available incomplete solvers. We also show that hybrid approaches such as inc-bmo-satlike can further enhance the accuracy of an incomplete solver.

We believe that such strategies, which are fundamentally incomplete (i.e., may not guarantee optimum even upon termination), need to be investigated further, since for many practical applications, one needs to find a good (but possibly suboptimal) solution very quickly. We have also observed that combinations of various approaches into a hybrid one also offers a lot of potential for improvement in MaxSAT solving and further exploration is very much warranted.

Acknowledgements

This work is partially funded by ECR 2017 grant (ECR/2017/001126) from SERB, DST, India, NSF award #1762363 and CMU/AIR/0022/2017 grant. We thank StarExec [38] for the computing resources.

References

[1] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In *Proc. International Joint Conference on Artificial Intelligence*, pages 2677–2683. AAAI Press, 2015.

[2] Carlos Ansótegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2017. http://maxsat-evaluations.github.io/2017, 2017. [Online; accessed 20-November-2017].

[3] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-Based Weighted MaxSAT Solvers. In *Proc. Principles and Practice of Constraint Programming*, pages 86–101. Springer, 2012.

[4] Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, **250**:37–57, 2017.

[5] Josep Argelich, Daniel Le Berre, Inês Lynce, Joao Marques-Silva, and Pascal Rapicault. Solving Linux Upgradeability Problems Using Boolean Optimization. In *Proc. Workshop on Logics for Component Configuration*, pages 11–22. EPTCS, 2010.

[6] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. MaxSAT Evaluation 2016. http://maxsat.ia.udl.cat/, 2016. [Online; accessed 18-April-2016].

[7] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proc. International Joint Conference on Artificial Intelligence*, pages 399–404. AAAI Press, 2009.

[8] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2018. https://maxsat-evaluations.github.io/2018/, 2017. [Online; accessed 20-November-2018].

[9] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2019. http://maxsat-evaluations.github.io/2019, 2019. [Online; accessed 18-July-2019].

[10] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *JSAT*, **7**(2-3):59–6, 2010.

[11] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring Local Search for Partial MaxSAT. In *Proc. AAAI Conference on Artificial Intelligence*, pages 2623–2629. AAAI Press, 2014.

[12] Shaowei Cai, Chuan Luo, and Haochen Zhang. From Decimation to Local Search and Back: A New Approach to MaxSAT. In *Proc. International Joint Conference on Artificial Intelligence*, pages 571–577. AAAI Press, 2017.

[13] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In *Proc. Principles and Practice of Constraint Programming*, pages 225–239. Springer, 2011.

[14] Emir Demirovic, Geoffrey Chu, and Peter J. Stuckey. Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers. In *Proc. Principles and Practice of Constraint Programming*, pages 99–108. Springer, 2018.

[15] Emir Demirovic and Peter J. Stuckey. LinSBPS. In *Proc. MaxSAT Evaluation 2018 Solver and Benchmark Descriptions*, **B-2018-2**, page 8. University of Helsinki, Department of Computer Science, 2018.

[16] Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, **2**(1-4):1–26, 2006.

[17] Yi Fan, Zongjie Ma, Kaile Su, Abdul Sattar, and Chengqian Li. Ramp: A Local Search Solver based on Make-positive Variables. In *Proc. MaxSAT Evaluation*, 2016.

[18] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability. In *Proc. Network and Distributed System Security Symposium*, 2017.

[19] Alexey Ignatiev, António Morgado, and Joao Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *Proc. International Conference on Theory and Applications of Satisfiability Testing*, pages 428–437. Springer, 2018.

[20] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2. In *Proc. MaxSAT Evaluation 2018 Solver and Benchmark Descriptions*, **B-2018-2**, page 22. University of Helsinki, Department of Computer Science, 2018.

[21] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, **32**(3):241–254, Sep 1967.

[22] Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Proc. Conference on Programming Language Design and Implementation*, pages 437–446. ACM, 2011.

[23] Saurabh Joshi, Prateek Kumar, Ruben Martins, and Sukrut Rao. Approximation Strategies for Incomplete MaxSAT. In *Proc. Principles and Practice of Constraint Programming*, pages 219–228. Springer, 2018.

[24] Saurabh Joshi, Prateek Kumar, Ruben Martins, and Sukrut Rao. Open-Wbo-Inc. http://github.com/sbjoshi/Open-WBO-Inc/, 2018. [Online; accessed 23-November-2018].

[25] Saurabh Joshi, Ruben Martins, and Vasco Manquinho. Generalized Totalizer Encoding for Pseudo-Boolean Constraints. In *Proc. Principles and Practice of Constraint Programming*, pages 200–209. Springer, 2015.

[26] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A Partial Max-SAT Solver. *JSAT*, **8**(1/2):95–100, 2012.

[27] Zhendong Lei and Shaowei Cai. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In *Proc. International Joint Conference on Artificial Intelligence*, pages 1346–1352. AAAI Press, 2018.

[28] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, **243**:26–44, 2017.

[29] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An Efficient Local Search Algorithm for Weighted Maximum Satisfiability. *IEEE Transactions on Computers*, **64**(7):1830–1843, 2015.

[30] Joao Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Mathematics and Artificial Intelligence*, **62**(3-4):317–343, 2011.

[31] Joao Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On Computing Minimal Correction Subsets. In *Proc. International Joint Conference on Artificial Intelligence*, pages 615–622. AAAI Press, 2013.

[32] Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental Cardinality Constraints for MaxSAT. In *Proc. Principles and Practice of Constraint Programming*, pages 531–548. Springer, 2014.

[33] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A Modular MaxSAT Solver,. In *Proc. International Conference on Theory and Applications of Satisfiability Testing*, pages 438–445. Springer, 2014.

[34] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Proc. Principles and Practice of Constraint Programming*, pages 564–573. Springer, 2014.

[35] Alexander Nadel. Solving MaxSAT with Bit-Vector Optimization. In *Proc. International Conference on Theory and Applications of Satisfiability Testing*, pages 54–72. Springer, 2018.

[36] Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic Polynomial Watchdog Encoding for Solving Weighted MaxSAT. In *Proc. International Conference on Theory and Applications of Satisfiability Testing*, pages 37–53. Springer, 2018.

[37] Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP Hybrid MaxSAT Solver. In *Proc. International Conference on Theory and Applications of Satisfiability Testing*, pages 539–546. Springer, 2016.

[38] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A Cross-Community Infrastructure for Logic Solving. In *Proc. International Joint Conference on Automated Reasoning*, pages 367–373. Springer, 2014.

[39] Takayuki Sugawara. MaxRoster: Solver Description. In *Proc. MaxSAT Evaluation 2017: Solver and Benhmark Descriptions*, **B-2017-2**, page 12. University of Helsinki, Department of Computer Science, 2017.

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