





Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search

Emir Demirović¹  and Peter J. Stuckey^{2,3} 

¹ University of Melbourne, Melbourne, Australia
emir.demirovic@unimelb.edu.au

² Monash University, Melbourne, Australia

³ Data61 CSIRO, Melbourne, Australia

Abstract. We present a MaxSAT algorithm designed to find high-quality solutions when faced with a tight time budget, e.g. five minutes. The motivation stems from the fact that, for many practical applications, time resources are limited and thus a ‘good solution’ suffices. We identify three weaknesses of the linear MaxSAT algorithm that prevent it from effectively computing low-violation solutions early in the search and develop a novel approach inspired by local search to address these issues. Our varying resolution method initially considers a rough view of the soft clauses (*low resolution*) and with time refines and adds the remaining constraints until the original problem is solved (*high resolution*). In addition, we combine the technique with solution-guided search. We experimentally evaluate our approach on test bed benchmarks from the MaxSAT Evaluation 2018 and show that improvements can be achieved over the baseline linear MaxSAT algorithm.

Keywords: MaxSAT · Solution-guided search · Incomplete MaxSAT

1 Introduction

Satisfiability (SAT) is a fundamental and well-known problem in computer science. Given a Boolean formula, it is concerned in determining the existence of a satisfying interpretation. Its optimisation variant, Maximum Boolean satisfiability (MaxSAT), deals with computing the interpretation that maximises satisfiability. Given the tremendous improvements in solving technology, MaxSAT has found a wide range of applications in the field of combinatorial optimisation, such as timetabling [2, 13], planning, and scheduling. See [5, 24] for more details.

Substantial research efforts in the MaxSAT community have been directed towards *complete* MaxSAT solving, i.e. developing algorithms that exhaustively explore the search space. In theory, these techniques guarantee to compute the optimum solution. While this is a clear strong point and has proven to be effective

for a number of problems, for large and difficult problems, such as high school timetabling, computing the optimum solution with current technology cannot be done within a reasonable time frame.

As an alternative, *incomplete* algorithms relax the optimality criteria with the aim of providing a suitable trade-off between computational time and solution quality. It is not uncommon for complete MaxSAT algorithms to provide intermediary solutions, playing the role of both complete and incomplete approaches. However, the main focus is laid on *proving* optimality *later* rather than computing *good* solutions *early* in the search.

There has been growing interest in incomplete algorithms in recent years, with a surge of new methods at the recent MaxSAT Evaluation 2018. It has been observed that better anytime performance can be achieved when algorithmic design decisions are centred around finding high-quality solutions quickly. The algorithm presented in this paper follows this line of work.

The first step towards designing an efficient algorithm is to understand the underlying issues and limitations that are preventing current incomplete algorithms from effectively computing good solutions early in the search. We focus on the linear MaxSAT algorithm, an upper-bounding method which repeatedly calls a Satisfiability (SAT) solver, each time imposing constraints to find a solution better than previously found. We identified three core problems with the linear MaxSAT algorithm: scalability, lack of guidance towards good solutions, and a tendency to focus on poor regions of the search space.

We designed an algorithm that aims to address these issues. There are two key components to our approach: (1) a novel *varying resolution* technique and (2) directed search around the currently best-known solution. The former simplifies the formula to roughly approximate the original instance (*low resolution*) and with time refines its view until the original constraints are rebuilt (*high resolution*). The benefits are two-fold: from a strategic side, it aims to satisfy the high impact constraints early in the search, and from a practical side, it allows the linear MaxSAT algorithm to scale by reducing memory requirements. The second key component directs the solver to provide incremental improvements to the currently best-known solution. While this technique has been used in other works [6,7], we provide a subtle yet impactful variation that provides notable improvements for our purposes of incomplete solving. When the two key techniques are combined, better results are obtained over the baseline linear MaxSAT algorithm on benchmarks from the MaxSAT Evaluation 2018.

To summarise, our contributions are as follows:

- We identify three core issues with the linear MaxSAT algorithm that hinders it in computing high-quality solution early in the search.
- We develop a novel varying resolution approach and combine it with a more effective solution-guided search strategy.
- We experimentally evaluate of our algorithm in the context of *incomplete* MaxSAT solving and study the impact of each individual component. Our results demonstrate that varying resolution and solution-guided search

provide improvements over the baseline. We note that our approach was ranked as the *best* performing solver in the incomplete weighted 300s track of the MaxSAT Evaluation 2018.

2 Preliminaries

SAT and MaxSAT. The Satisfiability problem (SAT) is concerned with deciding whether or not there exists an assignment of truth values to variables such that a given propositional logic formula is satisfied. A *literal* l is a Boolean variable x or its negation $\neg x$. A *clause* c is a disjunction of literals, $c \equiv l_1 \vee l_2 \vee \dots \vee l_n$. A propositional formula is, for our purposes, a set of clauses understood as their conjunction, thus in *conjunctive normal form*. An assignment θ is a mapping from a set of Boolean variables $x \in \text{vars}(\theta)$ to a value *true* or *false*. We extend θ to map negative literals, by defining $\theta(\neg x) = \neg\theta(x)$. An assignment θ satisfies a clause c , written $\theta \models c$, if for some literal l in the clause c , $\theta(l) = \text{true}$. In Partial Weighted MaxSAT, clauses are partitioned into hard H and soft S clauses. Each soft clause c is given a weight $w(c)$. The goal is to find an assignment that satisfies the hard clauses and minimises the weighted sum of the unsatisfied soft clauses. An alternative viewpoint for MaxSAT [9], which we adopt throughout this paper, is to associate an *objective variable* with each soft clause and state the problem as satisfying hard clauses while minimising the weighted sum of objective variables. See [10] for more information on SAT and MaxSAT.

CDCL Solvers for SAT [31]. The state of the art for solving SAT problems is based on conflict driven clause learning. The key components are unit propagation, activity based search, and clause learning. Unit propagation of a set of clauses P and a partial assignment θ , repeatedly finds a clause $c \equiv l_1 \vee l_2 \vee \dots \vee l_n \in P$ where $\theta(l_i) = \text{false}$, $1 \leq i < n$ for all literals and extends θ so that $\theta(l_n) = \text{true}$. The literal has c recorded as its reason for becoming true. If $\theta(l) = \text{false}$ for all literals l_i in c the solver detects unsatisfiability. The SAT solver applies unit propagation to extend an initially empty assignment θ . Afterwards, the solver chooses a literal and extends θ to make the literal *true* (treating it as an assumption) and applies unit propagation again. The choice of literal is usually based on the variables that have been in the most recent failures. On detecting unsatisfiability, the solver performs conflict analysis to create a nogood/learned clause which is added to the set of clauses to be solved. Solving continues until either a satisfying assignment is discovered, or unsatisfiability is proven.

Phase Saving [27, 29]. SAT solvers repeatedly make decisions on both branching variables and values. Variables are chosen based on their recent activity in conflicts (VSIDS scheme [25]). A wide-spread approach for truth value assignment is based on phase saving [27], where the solver selects the most recently used value in the search for the variable. Therefore, after backtracking, the solver aims to return to its previous state as closely as possible. Hence, clauses learnt about the previous region of the search space will still be relevant.

The Generalised Totaliser [19]. The Pseudo-Boolean constraint $\sum w_i \cdot x_i < k$ is converted into propositional logic by encoding a binary tree, where the leaf nodes are the input variable. Each parent node contains weighted variables that represent partial sums of its children. The root contains the variables that represent the total sum of the input variables. The desired constraint is obtained by forcing violating output variables to false. The encoding roughly depends on the number of distinct weights, as this is related to the number of possible partial sums. Thus, the encoding is *pseudo-polynomial*, but does not depend on the magnitude of the weights, which can be seen as a unique advantage.

Algorithm 1: The Linear algorithm for MaxSAT

Input: A set of hard H clauses and objective variables X . Each $x_i \in X$ is associated with a weight $w(x_i)$.

Output: An optimal solution θ^* minimising $\sum_{x_i \in X} w(x_i) \cdot x_i$

```

1 begin
2    $\theta^* \leftarrow \emptyset$ 
3    $P \leftarrow H$ 
4   while  $\exists \theta, \forall c \in P. \theta \models c$  do
5      $\theta^* \leftarrow \theta$ 
6      $k \leftarrow \text{cost}(\theta, X)$ 
7      $P \leftarrow P \cup (\sum_{x_i \in X} w(x_i) \cdot x_i < k)$ 
8   return  $\theta^*$ 

```

Example 1. Consider the encoding of the pseudo-Boolean constraint $8x_1 + 5x_2 + 3x_3 + x_4 < 9$. We create a node n representing the sum $n = 8x_1 + 5x_2$ defined by Booleans $\llbracket n \geq 5 \rrbracket$ and $\llbracket n \geq 8 \rrbracket$ and clauses $x_1 \rightarrow \llbracket n \geq 8 \rrbracket$, $x_2 \rightarrow \llbracket n \geq 5 \rrbracket$ and $x_1 \wedge x_2 \rightarrow \text{false}$. The last clause encodes the fact that the partial sum is already too big. Similarly we create a node m representing the sum $m = 3x_3 + x_4$ using Booleans $\llbracket n \geq 1 \rrbracket$, $\llbracket n \geq 3 \rrbracket$, $\llbracket n \geq 4 \rrbracket$ and clauses $x_3 \rightarrow \llbracket n \geq 3 \rrbracket$, $x_4 \rightarrow \llbracket n \geq 1 \rrbracket$ and $x_3 \wedge x_4 \rightarrow \llbracket n \geq 4 \rrbracket$. The root node s encoding the entire sum is encoded using Booleans $\llbracket s \geq 1 \rrbracket$, $\llbracket s \geq 3 \rrbracket$, $\llbracket s \geq 4 \rrbracket$, $\llbracket s \geq 5 \rrbracket$, $\llbracket s \geq 8 \rrbracket$ and the clauses $\llbracket m \geq 1 \rrbracket \rightarrow \llbracket s \geq 1 \rrbracket$, $\llbracket m \geq 3 \rrbracket \rightarrow \llbracket s \geq 3 \rrbracket$, $\llbracket m \geq 4 \rrbracket \rightarrow \llbracket s \geq 4 \rrbracket$, $\llbracket n \geq 5 \rrbracket \rightarrow \llbracket s \geq 5 \rrbracket$, $\llbracket n \geq 5 \rrbracket \wedge \llbracket m \geq 1 \rrbracket \rightarrow \llbracket s \geq 6 \rrbracket$, $\llbracket n \geq 8 \rrbracket \rightarrow \llbracket s \geq 8 \rrbracket$, $\llbracket n \geq 5 \rrbracket \wedge \llbracket n \geq 3 \rrbracket \rightarrow \llbracket s \geq 8 \rrbracket$, $\llbracket n \geq 5 \rrbracket \wedge \llbracket m \geq 4 \rrbracket \rightarrow \text{false}$, $\llbracket n \geq 8 \rrbracket \wedge \llbracket m \geq 1 \rrbracket \rightarrow \text{false}$, $\llbracket n \geq 8 \rrbracket \wedge \llbracket m \geq 3 \rrbracket \rightarrow \text{false}$, and $\llbracket n \geq 8 \rrbracket \wedge \llbracket m \geq 4 \rrbracket \rightarrow \text{false}$. In fact the s literals are not needed, they are included to show the general process of building a node from two children. We only need to keep the clauses encoding incompatible combinations of n and m .

Note that we encode the constraint $800x_1 + 500x_2 + 300x_3 + 100x_4 < 900$ identically. \square

The Linear MaxSAT Algorithm [14,20,21,23]. The optimal solution to a MaxSAT instance can be obtained by solving a series of SAT problems. This is depicted in Algorithm 1. It makes repeated calls to a SAT solver. After each call, it adds a pseudo-Boolean constraint to the formula that enforces the formula to

only admit solutions that have a cost strictly lower than the current best solution. In Sect. 5, we discuss different encodings for the pseudo-Boolean constraint and in the rest of the paper focus our attention on the generalised totaliser encoding (see above). The algorithm iterates until it proves unsatisfiability, in which case the optimal solution was computed in the previous iteration.

3 Algorithm

Our algorithm is designed for short run times, e.g. five minutes. The assumption is that proving optimality within the given time frame is infeasible. Thus, the aim is to find ‘good solutions’ early during the search. The main challenge is to determine a strategy which can identify where the ‘good’ solutions reside in the search space and ensure scalability across a wide range of benchmarks.

Our approach is based on the linear MaxSAT algorithm. This method was chosen as it was the best performing solvers in the *incomplete unweighted 60s* track of the MaxSAT Evaluation 2017. In addition, it has shown competitive resulting for certain applications, e.g. high school timetabling [13]. Two techniques play a key role in our algorithm: (1) a novel *varying resolution* approach and (2) directed search around the currently best-known solution.

3.1 Issues with the Linear MaxSAT Algorithm

To obtain a better understanding of our algorithm, it is important to note the core issues with the linear MaxSAT algorithm. We identified three main issues in the context of incomplete saving: (1) scalability and sensitivity to the values of the weights of the objective variables, (2) lack of a strategy to guide the search towards solutions with low objective value, and (3) proneness to falling in “local optima”, i.e. excessively spending efforts proving unsatisfiability in a certain region of the search space rather than exploring a different part of the search space. The varying resolution approach aims to address the first two points, while the directed search tackles the third point and partially the second. These issues are described in greater detail below.

Issue #1: Scalability and Weight-Value Sensitivity. The linear MaxSAT algorithm encodes a single large pseudo-Boolean constraint, which is directly dependent on the values of the weights of the objective variables.

The generalised totaliser pseudo-Boolean encoding [19], used in this work, roughly depends on the number of unique values of the weights. Therefore, when faced with a large number of diverse weights, the number of clauses and auxiliary variables required to encode the pseudo-Boolean can be prohibitively high. As a result, the pseudo-Boolean encoding can dominate the algorithmic performance and become a bottleneck. Other encodings suffer from related issues.

We note that MaxSAT algorithms that do not require explicitly encoding the pseudo-Boolean constraints are largely unaffected by the variety in weights, e.g. core-guided approaches. On a related note, WPM3 [6] uses the splitting rule

to allow using an encoding where the weights are equal, i.e. cardinality constraints. However, these algorithms focus on increasing the lower bound rather than computing good solutions early in the search.

Issue #2: Lack of Guidance Towards Good Solutions. In each iteration of the linear algorithm, the SAT solver merely seeks to find a satisfying assignment and not necessarily a solution with low cost. Therefore, there is no guidance towards good solutions, which might lead the algorithm to spend excessive time searching in areas that potentially fine-tune small improvements to the objective even though the crucial soft constraints are left unattended.

Issue #3: Tendency to Focus on Poor Regions of the Search Space. This issue is linked to the underlying value-selection heuristic of the SAT algorithm: phase saving. While phase saving is known to be effective for pure satisfiability problems, it can introduce undesired behaviour when used in the linear MaxSAT algorithm. The problem stems from the fact that upon conflict detection and backtracking, phase saving aims to drive the search back into a similar region of the search space as before. This is systematically done through value-assignments for variables: once a new variable is selected, the value most recently used for that variable will be assigned to it. As a result, once the algorithm reaches a region of the search space where there are no better solutions, it will effectively spend its efforts in proving unsatisfiability. Unfortunately, this can be time-consuming and does not lead to finding good solutions quickly.

3.2 Our Approach

There are two key components in our algorithm: the varying resolution approach and solution-guided search, complementary techniques that aim to address the identified issues of the linear MaxSAT algorithm. The former ensures scalability (Issue #1) and guides the search towards good solutions on a high level (Issue #2), while the latter provides incremental improvements to the current best solution (Issues #2 and #3). These components are built into the linear MaxSAT algorithm and exhibit a high degree of synergy, resulting in a better algorithm than the baseline linear MaxSAT algorithm.

Key Component #1: Varying Resolution Approach. The aim of this part is to address Issue #1 and #2. It starts by viewing the MaxSAT formula in *low resolution* by decreasing the weights for all constraints. The weights reduced to zero are removed. After the resulting problem is solved, the weight values are increased (*increase the resolution*), a portion of the previously ignored constraints are added, and the problem is resolved. This process iterates until the problem is viewed in *high resolution*, i.e. the original formula is restored and solved. Weight adjustment results in a heuristic that approximates the formula and reduces the memory requirements, which in turns offers speed-ups. In theory, the procedure preserves completeness, i.e. does not remove any optimal solution, but in practice, only a few iterations of the algorithm are executed within the allocated time resources. We discuss related approaches, namely stratification for core-guided approaches [4] and weight-clustering [18], in Sect. 5.

Algorithm 2: Compute the initial cutoff value

Input: A set of objective variables X , a mapping $w : X \rightarrow \mathbf{N}$, and the threshold coefficient $\beta \in [0, 1]$.

Output: Initial cutoff value d

```

1 begin
2    $S \leftarrow \sum_{x \in X} w(x)$ 
3    $k \leftarrow \max_{x \in X} \{dec\_digits(w(x))\}$ 
4   for  $i = 1..k$  do
5      $frac\_sum[i] \leftarrow \sum_{x \in X \wedge dec\_digits(x)=i} w(x)$ 
6    $d \leftarrow k$ 
7   for  $i = 1..k$  do
8     if  $frac\_sum[i] \div S \geq \beta$  then
9        $d \leftarrow i$ 
10      break
11  return  $10^{(d-1)}$ 

```

To explain our algorithm in detail, we first discuss the initial cutoff value computation, present the varying resolution approach, and lastly describe our modification to the linear MaxSAT algorithm.

Initial Cutoff Value Computation. Algorithm 2 describes the procedure. The goal is to determine a cutoff threshold that partitions the objective variables into low- and high-weighted variables. It first computes: S - the sum of the weights, k - the number of decimal digits used to represent the largest weight, and $frac_sum$ - the array where $frac_sum[i]$ represents the sum of weights with exactly i decimal digits. Note that the number of digits is computed as $dec_digits(x) = \lfloor \log_{10}(x) \rfloor + 1$. Afterwards, the cutoff value is chosen based on the total contribution of weights with precisely d digits with respect to the overall MaxSAT problem. The smallest value d that meets the specified threshold β is selected, or the default value k if no such value exists. The cutoff value is returned as $10^{(d-1)}$. The intuition is that the cutoff point discriminates weights between those that contribute significantly towards the objective and those that do not. The parameter $\beta \in [0, 1]$ regulates the sensitivity of the division: lower/higher values for β lead to lower/higher cutoff values.

Example 2. Consider the formula with $X = \{x_i : i \in \{0, 1, \dots, 8\}\}$ and $w = \{w_0 \mapsto 1200, w_1 \mapsto 800, w_2 \mapsto 700, w_3 \mapsto 500, w_4 \mapsto 50, w_5 \mapsto 15, w_6 \mapsto 9, w_7 \mapsto 8, w_8 \mapsto 2\}$ and parameter $\beta = 0.20$. The sum of weights is 3284 and $frac_sum = \{1 \mapsto 19, 2 \mapsto 65, 3 \mapsto 2000, 4 \mapsto 1200\}$. The inner *if* condition is not satisfied for $i \in \{1, 2\}$, as neither $\frac{19}{3284} \geq 0.20$ nor $\frac{65}{3284} \geq 0.20$, but will trigger for $i = 3$ since $\frac{2000}{3284} \geq 0.20$. Therefore, $d = 3$ and the returned cutoff is 100. \square

Varying Resolution. Algorithm 3 gives an overview. The algorithm starts by computing the initial *cutoff value* (Algorithm 2). Iteratively, a new MaxSAT formula is built, where the hard constraints are as in the original formula, and the weights

of objective variables are divided by the cutoff value (rounded down). The new formulation, along with the best solution found so far θ^* and the original formula, are used to initialise the linear MaxSAT algorithm. The best solution is used by the solution-guided search component, while the original formula is required due to our previously discussed modification of the linear MaxSAT algorithm (see next subsections for both points). After the resulting formula is solved, the cutoff value is decreased and the process is repeated until the original formula is solved. Note that if the sum of the weights is lower than a given parameter α , the varying resolution approach is deemed unnecessary, i.e. the cutoff is set to one and the algorithm proceeds as a linear MaxSAT algorithm with solution-guided search (see component #2). The procedure can be viewed as a search by exponentially decreasing steps, where the approximate objective function is refined at an exponential rate each iteration until the original objective is restored.

Example 3. (continued) Let $\alpha = 1000$. As $\sum_{x_i} w(x_i) \geq \alpha$, the cutoff d is set to 100 (see Example 2). Therefore, $w' = \{x_0 \mapsto 12, x_1 \mapsto 8, x_2 \mapsto 7, x_3 \mapsto 5, x_4 \mapsto 0, x_5 \mapsto 0, x_6 \mapsto 0, x_7 \mapsto 0, x_8 \mapsto 0\}$, and $X' = \{x_0, x_1, x_2, x_3\}$. After the simplified formula is solved, d is decreased to 10 and the process is repeated. \square

Learned clauses are kept as usual during the search within each individual iteration, but the SAT solver is rebuilt at the beginning of each iteration (Algorithm 3, line 11). Learned clauses are not shared in between iterations of varying resolution, as learned clauses in one iteration might refer to auxiliary variables in the pseudo-Boolean encoding that are no longer present in the next iteration.

The approximate objective function requires fewer auxiliary variables and clauses than the original pseudo-Boolean constraint. Recall that the size of the generalised totaliser encoding [19] is related to the number of unique weight values, e.g. the smallest encoding is obtained if all weights are the same value. Dividing the weights by the cutoff results in fewer unique weights, leading to a smaller encoding, which in turn reduces the memory requirements. Note that varying resolution is designed for shorter run times and thus are not particularly suitable for longer runtimes, i.e. the last iteration of varying resolution is the standard linear MaxSAT algorithm.

Observation 1. *Given an initial constraint $I \equiv \sum_{x \in X} w(x) \cdot x \leq ub - 1$, the r rounded version of the constraint is given by $I_r \equiv \sum_{x \in X} \lfloor \frac{w(x)}{r} \rfloor \cdot x \leq \lfloor \frac{ub-1}{r} \rfloor$. It follows that $I \models I_r$, i.e. all solutions of I are also solutions of I_r , since I_r is a Gomory cut [16] derived from I . Hence, adding this constraint does not exclude any optimal solution to the original problem.*

Observation 2. *The varying resolution algorithm is complete regardless of the choice for parameters α and β .*

Observation 3. *The varying resolution is anytime, i.e. it provides intermediary results during its execution.*

Linear MaxSAT Modification. The standard linear MaxSAT algorithm is modified as follows. It additionally stores the original MaxSAT formula and the best

Algorithm 3: The Varying Resolution Approach

Input: A set of hard H clauses and objective variables X , a mapping $w : X \rightarrow \mathbf{N}$, and threshold coefficients $\alpha \in \mathbf{N}$ and $\beta \in [0, 1]$

Output: An optimised solution θ^*

```

1 begin
2    $\theta^* \leftarrow \emptyset$ 
3   if  $\sum_{x \in X} w(x) \geq \alpha$  then
4      $d \leftarrow \text{compute\_initial\_cutoff}(X, \beta)$ 
5   else
6      $d \leftarrow 1$ 
7    $\text{cutoff} \leftarrow 10^{d-1}$ 
8   while  $\text{cutoff} \geq 1$  do
9      $w'(x) = \lfloor \frac{w(x)}{\text{cutoff}} \rfloor$ 
10     $X' = \{x : x \in X \wedge w'(x) > 0\}$ 
11     $\text{solver} \leftarrow \text{initialiseMaxSAT}(H, X, w, X', w')$ 
12     $\text{solver.setInitialSolution}(\theta^*)$ 
13     $\theta^* \leftarrow \text{solver.solve}()$ 
14     $\text{cutoff} \leftarrow \lfloor \frac{\text{cutoff}}{10} \rfloor$ 
15  return  $\theta^*$ 

```

solutions with respect to the current and the original MaxSAT formula. Note that an assignment with a lower objective value for the simplified problem in the varying resolution approach does not necessarily lead to a better solution for the original problem. Therefore, once a new assignment is computed, its cost is computed with respect to the original formula, and it is kept as the globally best solution if its cost is lower than the previous best solution. Regardless of the outcome, the algorithm proceeds as usual, i.e. adds the upper bound with respect to the newly found locally best solution. Thus, it optimises its current problem, but only updates the global solution if it is better with respect to the original MaxSAT formula.

Example 4. Considering the formula within the varying resolution approach with $d = 10$: $w' = \{x_0 \mapsto 120, x_1 \mapsto 80, x_2 \mapsto 70, x_3 \mapsto 50, x_4 \mapsto 5, x_5 \mapsto 1, x_6 \mapsto 0, x_7 \mapsto 0, x_8 \mapsto 0\}$. The linear MaxSAT algorithm is called and assume it finds the solution θ that only violates x_5 . The objective value of θ is 1 locally and 15 globally. Both values are kept as these are the best values found in their respective categories. The pseudo-Boolean constraint $\sum w'(x) < 1$ is added to the MaxSAT formula and the SAT solver is called again. Now assume the solver finds a new solution that violates x_6 and x_7 . The solution is kept as the best local solution ($\sum w'(x) = 0$), but the best global is not updated ($\sum w(x) = 17 \geq 15$). As locally no further improvements can be made, the linear MaxSAT algorithm stops, leading to a new iteration of the varying resolution approach with $d = 1$ where θ is passed as the initial solution. \square

Key Component #2: Solution-Guided Search. In local search, *intensification* aims to provide improvements to the solution by searching through a neighbourhood of solutions *close* to the current solution. Thus, a better solution is found by iteratively performing small incremental changes to the currently considered solution, driving the solution into a (local) minima. The essence of this idea can be captured in a complete search algorithm by using the following value-selection heuristic: once a branching variable has been chosen, assign the value to the variable that it assumes in the best-known solution. Hence, the search progresses *close* to the best-known solution, resembling local search. In our algorithm, we use this value-selection heuristic, as it partially addresses issues #2 and #3 of the linear MaxSAT algorithm.

Similar techniques were used under various names, e.g. solution-based phase saving [1, 6, 12], solution-guided search [7], and large neighbourhood search [30]. The phase saving strategy used in WPM3 [6] is the closest to our work. The difference is subtle yet impactful: we apply solution-guided search to *all* variables in the MaxSAT formula, including auxiliary variables introduced by the pseudo-Boolean encoding, as opposed to only considering variables that appear in the original MaxSAT formula as in WPM3 [6]. For our experimental setting, our strategy proved to be more effective, but we note that WPM3 considered a different setting for their phase saving, i.e. it was considered for solving subproblems generated during the search with the aim of increasing the lower bound.

4 Experimental Results

We performed a detailed computational study to empirically evaluate the effect of varying resolution and solution-guided search.

4.1 Setting

Our setting is the same as in incomplete track of the MaxSAT Evaluation 2018. Thus, we consider unweighted and weighted benchmarks with 60 and 300 s time-outs, for a total of four separate settings. The evaluation uses industrial and application benchmarks. The comparisons are performed on a total of 153 and 172 unweighted and weighted benchmarks, respectively. The experiments were performed on the StarExec cluster, allocating 32 GB of RAM per benchmark.

Scoring. The scoring of a solver for the incomplete track is the sum of scores s_i for each instance. For instance i , a solver finding a solution with objective o_i is awarded score $s_i = bo_i/o_i$ where bo_i is the best objective found by any solver on that instance during the 60 and 300 s runs. If a solver finds no solution for an instance i , the corresponding score is $s_i = 0$. The best solution is taken from the 300 s track.

Our Solver: LinSBPS. We implemented varying resolution and solution-guided search in Open-WBO [23], an open-source MaxSAT solver.

Other Solvers. The remaining solvers used in the evaluation are discussed in more detail in the Sect. 5.

4.2 Results and Discussions

We provide experiments that support our previous claims. The same timeouts and benchmarks are used across experiments. Note that the *score* metric is relative to the solvers considered, i.e. the score for a particular benchmark depends on the best solution computed by the considered solvers. Hence, the score values may differ in different experiments.

Effect of Varying Resolution and Solution-Guided Search. In Table 1a we compare the performance of our techniques compared to the baseline linear algorithm. We consider four variants, depending on whether varying resolution and solution-guided search is used. Note that varying resolution is only used for weighted benchmarks which are deemed as *large enough*, as detailed in Algorithm 3. For the considered benchmark set, varying resolution was used on 89 out of 172 benchmarks (51%).

Each component, varying resolution and solution-guided search, improves the baseline. The best approach is obtained by combining both techniques.

Number of Unique Weights Produced. Varying resolution reduces the number of unique weights in the benchmarks, thus leading to more compact encodings with the GTE. On average, the number of distinct weights drops from 1342 to 29 in the first iteration of varying resolution.

Number of Objective Variables Considered. The underlying MaxSAT formula is simplified with varying resolution. Nevertheless, most of the objective variables are still taken into account, even in the first iteration. On average, the algorithm considers 87% percent of the total number of objective variables in the first iteration of varying resolution.

Number of Iterations Performed. For the benchmarks that use varying resolution, on average, 1.19 iterations were executed with 6.04 iterations needed to restore the original formula.

GTE vs. Adder Pseudo-Boolean Encoding. One of the benefits of varying resolution comes from its ability to produce a smaller pseudo-Boolean encoding. As an alternative, in Table 1b we consider the adder encoding, which represents numbers in binary form and encodes binary adders. This allows for a significant reduction in the encoding size, at the expense of arc consistency.

Our comparison was done only on the benchmarks that use varying resolution. However, while effective for complete solving [20], the adder encoding shows weaker performance for incomplete solving. We believe the loss of arc consistency for the adder encodings forces the solver to spend more time in search, which is detrimental given the tight time budget.

Solution-Guided Search. In Table 1c we compare our variant of solution-guided search with the phase saving strategy used in WPM3 [6]. The difference in the techniques is subtle yet impactful. Nevertheless, regardless of the variant chosen, incrementally improving an existing solution proved to be beneficial for incomplete MaxSAT solving. Compared to WPM3 [6], our variant considers all

variables and not only the original variables. This proved to be advantageous for our setting. As discussed previously in Sect. 3.2, our experimental setting differs from the one considered in WPM3, and hence it must be emphasised that our claims only hold for our particular case of incomplete solving with the linear algorithm. The auxiliary variables in the formula are implied by the original variables in the pseudo-Boolean constraint. Thus, following the idea of remaining *close* to the best solution, all variables must be considered. Setting a different value to an auxiliary variable reflects on the original variables, which was undesirable in our setting.

Parameter Choice. The parameter α defines the size of the benchmark required to activate varying resolution, while β is used to discriminate between more and less important weights. The final values chosen in the solver are $\alpha = 5 \cdot 10^5$ and $\beta = 0.05$. Note that no parameter tuning was performed. The parameter choice discussion that follows is presented as a post-analysis.

Varying resolution is activated when the sum of weights in a benchmark exceeds the threshold α . To study other possible choices for α , we sort the considered benchmarks by the sum of their weights. The 83rd smallest value is $489 \cdot 10^3$. However, the 73rd and 93rd are $71 \cdot 10^3$ and $1603 \cdot 10^3$, exhibiting substantial differences in values. Thus, α can be varied significantly with little effect. Therefore, we selected $\alpha = 5 \cdot 10^5$ in an *ad hoc* manner and decided not to fine-tune the parameter on the previous competition benchmark set, as doing so would likely lead to overfitting.

Parameter β was kept low since our intention was to discard low-valued weights that increase the encoding size but do not provide a significant difference in the objective. Thus, we selected $\beta = 0.05$, i.e. we stop discarding weights with d digits if their contribution is at least 5%.

Comparison with the MaxSAT Evaluation 2018 Solvers. In Table 2 we show the results from the MaxSAT Evaluation 2018 as a comparison with other state-of-the-art incomplete MaxSAT solvers. Our solver, LinSBPS, uses the techniques described in this paper. Our approach can be further improved using *core-boosting* [8] as a preprocessing step, but the main aim of these experiments is to demonstrate the effectiveness of the techniques presented in this paper.

Weighted Track. Our algorithm achieved the *best* rank in the 300s category. A detailed view of these results is given in Fig. 1 (top), which shows the distribution of scores per instance. For each solver, the scores for every instance is computed, the resulting array is sorted, and then plotted as a curve. We can see that our approach provides highly competitive results for the majority of the benchmarks, with only a handful of cases where the score is below 0.8. This illustrates the robustness of our technique when handling a diverse set of benchmarks. For the 60s track, our approach is ranked second.

Our approach takes into account most of the objective variables. Open-WBO-Inc-BMO, as a solver with comparable performance, in contrast, initially aggressively optimises the most important constraints. This seems to provide better

results for 60s runs. However, as more time is allocated, our approach is able to exploit a broader view of the problem, while the other approach keeps optimising a rough approximation.

Unweighted Track. Our approach solely relies on solution-guided search to provide improvements over the baseline for the unweighted track. Nevertheless, our method ranked second and third in the 60 and 300s track, respectively. From Fig. 1 (bottom), we can see that there is a higher deviation in solver performance depending on the benchmark. While for the weighted benchmarks our approach achieved consistently good results when compared with others, for the unweighted track there are no robust solvers: the score distributions are scattered across the interval [0.1, 1] for each solver. We believe this is because it is harder to identify the key constraints for unweighted compared to the weighted instances, and thus there is a higher fluctuation between the results. The best performing solver in the unweighted track, SATLike, is a local search solver specialised in exploring different areas of the search space quickly rather than using sophisticated reasoning technique such as CDCL, which could explain its effectiveness for these benchmarks.

Table 1. Comparison of different variants of our approach. 300 s. (a) The effect of each individual component; (b) Comparison with the adder encoding; (c) Comparison with solution-guided search used in WPM3: SGS(OV).

(a)		(b)		(c)	
Solver	Score	Solver	Score	Solver	Score
VR+SGS	162.00	VR+SGS	161.55	VR+SGS	161.03
SGS	144.46	VR	140.05	VR+SGS(OV)	148.73
VR	140.47	Adder+SGS	129.87	SGS	143.81
Baseline	128.8	Baseline+Adder	125.26	SGS(OV)	132.43

Table 2. Results from the MaxSAT Evaluation 2018. The *score* listed for solvers

(a) Weighted 60 s		(b) Weighted 300 s		(c) Unweighted 60 s		(c) Unweighted 300 s	
Solver	Score	Solver	Score	Solver	Score	Solver	Score
Open-WBO-Inc-BMO	0.810	LinSBPS	0.900	SATLike-c	0.735	SATLike-c	0.854
LinSBPS	0.799	Open-WBO-Inc-BMO	0.842	LinSBPS	0.705	maxroster	0.829
maxroster	0.773	maxroster	0.804	SATLike	0.675	LinSBPS	0.782
Open-WBO-Inc-Cluster	0.743	Open-WBO-Inc-Cluster	0.762	Open-WBO-Inc-OBV	0.654	SATLike	0.702
SATLike-c	0.696	SATLike-c	0.747	Open-WBO-Inc-MCS	0.631	Open-WBO-Inc-OBV	0.842
Open-WBO-Gluc	0.669	SATLike	0.702	Open-WBO-Gluc	0.612	Open-WBO-Inc-MCS	0.762
SATLike	0.661	Open-WBO-Gluc	0.68	Open-WBO-Riss	0.564	Open-WBO-Gluc	0.68
Open-WBO-Riss	0.638	Open-WBO-Riss	0.663	maxroster	0.541	Open-WBO-Riss	0.663

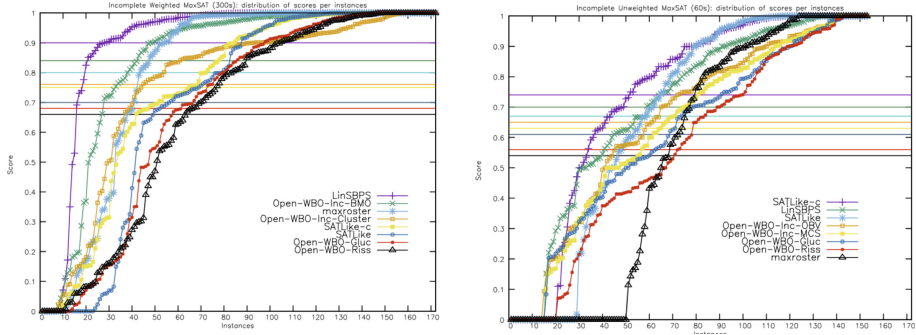


Fig. 1. Detailed results for the weighted 300s (left) and unweighted 60s (right) track. Image courtesy of the MaxSAT Evaluation 2018.

5 Related Work

Local Search. These methods share a common pattern: start by generating a random assignment and iteratively select a variable from an unsatisfied clause and flip its assignment. Complex reasoning mechanisms are typically not employed as in complete solvers, e.g. CDCL. Rather, the success of the methods comes from their ability to rapidly explore a large number of solutions through the use of specialised data structures, careful implementation, and heuristics.

In the recent MaxSAT Evaluation, *SATLike* [22] won the unweighted incomplete track and demonstrated good performance in the weighted track of the recent MaxSAT Evaluation 2018. Its key component is a novel weighting scheme that dynamically changes the weights of clauses during the search.

Complete Techniques for Incomplete Solving. In some cases, complete algorithms report intermediate solutions and thus can take the role of incomplete methods. The linear MaxSAT [14] solvers fall into this category and approaches differ in the way the upper bounding constraint is handled: *SAT4J* [21] uses linear propagators, a technique from constraint programming to avoid explicitly encoding the constraint into Boolean formula, while *QMaxSAT* [20] and *Open-WBO-Gluc* use the adder and GTE pseudo-Boolean encoding, respectively. The winner of the weighted incomplete track in 2017, *maxroster* [32], uses a stochastic solver to produce an initial solution before applying complete-based techniques.

Core-guided approaches [3, 15, 17, 26] consider an initial SAT formula where the soft clauses are treated as hard clauses. Iteratively, a SAT solver is used to compute either a satisfying assignment, which would represent the optimal solution, or an *unsatisfiable core*, i.e. a subset of clauses that cannot simultaneously be satisfied. The core is used to rewrite the formula, e.g. relax the formula by allowing at most one of the clauses from the core to be unsatisfied. *Hitting-set* approaches [11, 28] utilise unsatisfiable cores to separate MaxSAT solving into a SAT and an integer programming component. These approaches are inherently lower bounding and in their pure form do not produce any solution

other than the optimum, rendering them inapplicable to incomplete scenarios. However, when combined with other techniques, unsatisfiable cores can be used for anytime algorithms. For example, WPM3 [6] is a core-guided solver with a stratified approach [4], where initially only a subset of the soft clauses with the highest weights are considered, and after satisfiability is detected, a portion of the remaining soft clauses are added and the process is repeated. Obtaining cores with high weights contribute towards faster lower bounds, and during this process upper bounds are additionally computed. This was the best incomplete solver in the 2016 evaluation. At a high level, our strategy resembles the stratified method, but the underlying solving process and the reasoning behind the techniques make a clear distinction between the approaches.

Core-boosting [8] has recently been proposed to improve linear MaxSAT algorithms. The main idea is to run the linear MaxSAT algorithm after performing core-based rewriting for a limited time. The resulting formula has fewer soft clauses, which simplifies the pseudo-Boolean constraint required in the linear algorithm. Core-boosting can be seen as a form of preprocessing and can be combined with other linear algorithms, such as the one presented in this paper.

Incomplete Weight-Relaxation. Inc-BMO and Inc-Cluster [18] from the MaxSAT Evaluation bear the most similarity to our approach.

These methods cluster the objective variables. Each variable in a cluster is reassigned a *representative weight* as follows: the array of weights is sorted and the difference between adjacent elements is computed. The top $k - 1$ indices with the highest differences are selected, effectively partitioning the weights into k clusters. Each variable within a cluster is reassigned the arithmetic mean of the weights in the group. As a result, there are at most k different weights values.

The two approaches, Inc-BMO and Inc-Cluster, differ in the next step. In Inc-Cluster, a linear MaxSAT algorithm is applied to the new formula. In Inc-BMO, the resulting formula is solved as a lexicographical optimisation problem: the problem is solved considering only the variables with the highest weight, the sum of their violations is fixed to the computed value, and the process is repeated with the second-highest weighted variables, and so forth.

There are two main reasons for the success of these methods: (1) reducing the number of distinct weights results in more compact pseudo-Boolean encodings, increasing performance, and (2) Inc-BMO aggressively optimises the most important constraints. Note that, once the problem is simplified, the effects are irreversible. Therefore, each clustering choice plays an important role. The methods excel for problems where the clustering can be done effectively.

To further illustrate the difference with our approach, consider the MaxSAT problem from Example 2. For $k = 2$, the clustering algorithms partitions the weights into $P_1 = \{w_0, w_1, w_2, w_3\}$ and $P_2 = X - P_1$. Thus, violations within each cluster are treated equally. This is not an issue at the start of the algorithm. However, as the algorithm progresses, some form of refinement is necessary to provide better results. Varying resolution with a cutoff of 100 initially considers only P_1 as BMO-INC, but is able to differentiate between violations, i.e. $w' = \{12, 8, 7, 5\}$. Note that each iteration thereafter refines the formula.

6 Conclusion

We developed a novel approach to incomplete MaxSAT solving consisting of two key components: our *varying resolution* approach, and solution-guided search. The former initially views the problem in low-resolution and with time refines the constraints until the formula is solved in high resolution, i.e. the original problem. Solution-guided search provides incremental improvements by searching close to the current best solution. Overall, our algorithm has proven to be highly effective for short runtimes, placing first in the incomplete weighted 300s track of the MaxSAT Evaluation 2018.

Acknowledgements. We would like to thank the anonymous reviewers for their valuable feedback in preparing the final version of this paper.

References

1. Abío, I., Deters, M., Nieuwenhuis, R., Stuckey, P.J.: Reducing chaos in SAT-like search: finding solutions close to a given one. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 273–286. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21581-0_22
2. Achá, R.A., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and MaxSAT. *Ann. Oper. Res.* **218**(1), 71–91 (2014)
3. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT algorithm using cardinality constraints of bounded size. In: Proceedings of IJCAI 2015 (2015)
4. 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
5. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Proceedings of SAT 2009, pp. 427–440 (2009)
6. Ansótegui, C., Gabàs, J.: WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artif. Intell. J.* **250**, 37–57 (2017)
7. Beck, J.C.: Solution-guided multi-point constructive search for job shop scheduling. *J. Artif. Intell. Res.* **29**, 49–77 (2007)
8. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 39–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_3
9. Berg, J., Järvisalo, M.: Unifying reasoning and core-guided search for maximum satisfiability. In: Calimeri, F., Leone, N., Manna, M. (eds.) JELIA 2019. LNCS (LNAI), vol. 11468, pp. 287–303. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19570-0_19
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (2009)
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. Demirović, E., Chu, G., Stuckey, P.J.: Solution-based phase saving for CP: a value-selection heuristic to simulate local search behavior in complete solvers. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 99–108. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_7
13. Demirović, E., Musliu, N.: MaxSAT-based large neighborhood search for high school timetabling. *Comput. Oper. Res.* **78**, 172–180 (2017)
14. Eén, N., Sorensson, N.: Translating pseudo-boolean constraints into SAT. *J. Satisfiability Boolean Model. Comput.* **2**, 1–26 (2006)
15. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_25
16. Gomory, R.E.: Outline of an algorithm for integer solutions of linear programs. *Bull. Am. Math. Soc.* **64**, 275–278 (1958)
17. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of AAAI 2011 (2011)
18. Joshi, S., Kumar, P., Martins, R., Rao, S.: Approximation strategies for incomplete MaxSAT. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 219–228. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_15
19. 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
20. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: a partial max-sat solver. *J. Satisfiability Boolean Model. Comput.* **8**, 95–100 (2012)
21. Le Berre, D., Parrain, A.: The SAT4J library, release 2.2 system description. *J. Satisfiability, Boolean Model. Comput.* **7**, 59–64 (2010)
22. Lei, Z., Cai, S.: Solving (weighted) partial MaxSAT by dynamic local search for SAT. In: Proceedings of IJCAI, pp. 1346–1352 (2018)
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., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: a survey and assessment. *Constraints* **18**(4), 478–534 (2013)
25. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of DAC 2001, pp. 530–535 (2001)
26. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of AAAI 2014 (2014)
27. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_28
28. 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
29. Sellmann, M.: Disco-Novo-GoGo: integrating local search and complete search with restarts. In: Proceedings of AAAI 2006 (2006)

30. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49481-2_30
31. Marques Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, et al. [10], pp. 131–153
32. Sugawara, T.: Maxroster: solver description. In: MaxSAT evaluation 2017, p. 12 (2017)