Constraint-Based Techniques in Stochastic Local Search MaxSAT Solving

Andreia P. Guerreiro[1], Miguel Terra-Neves[1,2], Inês Lynce[1], José Rui Figueira[3], and Vasco Manquinho[1](✉)

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
{andreia,ines,vmm}@sat.inesc-id.pt

[2] OutSystems, Lisbon, Portugal
miguel.neves@outsystems.com

[3] CEG-IST, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
figueira@tecnico.ulisboa.pt

Abstract. The recent improvements in solving Maximum Satisfiability (MaxSAT) problems has allowed the usage of MaxSAT in several application domains. However, it has been observed that finding an optimal solution in a reasonable amount of time remains a challenge. Moreover, in many applications it is enough to provide a good approximation of the optimum. Recently, new local search algorithms have been shown to be successful in approximating the optimum in MaxSAT problems. Nevertheless, these local search algorithms fail in finding feasible solutions to highly constrained instances. In this paper, we propose two constraint-based techniques for improving local search MaxSAT solvers. Firstly, an unsatisfiability-based algorithm is used to guide the local search solver into the feasible region of the search space. Secondly, given a partial assignment, we perform Minimal Correction Subsets (MCS) enumeration in order to improve upon the best solution found by the local search solver. Experimental results using a large set of instances from the MaxSAT evaluation 2018 show the effectiveness of our approach.

Keywords: Maximum Satisfiability · Local search · Incomplete algorithms

1 Introduction

Over the last decade, a new generation of algorithms for Maximum Satisfiability (MaxSAT) problems has been proposed [1,13,40,42,44]. These new MaxSAT algorithms are usually based on iterative calls to a highly efficient Propositional Satisfiability (SAT) solver. For many industrial benchmarks, the current state-of-the-art MaxSAT algorithms are several orders of magnitude faster than branch-and-bound MaxSAT algorithms [35]. As a result, MaxSAT has been used extensively in many application domains, such as timetabling [4], fault localization in `C` programs [25], and design debugging [45], among others [20].

Despite the success of the new generation of MaxSAT solvers, there is still a wide range of large-scale applications where such solvers fail to prove optimality within a reasonable amount of time. In fact, in some applications, time is so crucial that it suffices to quickly find a good approximation to the optimum [48]. As a result, several incomplete MaxSAT algorithms [3,12,33,36,47] have been proposed with the aim of finding good solutions within a limited time window.

It is well-known that stochastic local search (SLS) solvers are known to be competitive when the problem instances are easy to satisfy. On the other hand, SAT-based algorithms are more effective in problem instances with more constraints, while having more difficulties to deal with the optimization problem.

In the context of SAT solving, there is a large literature on combining SAT-based procedures and SLS techniques (e.g. [5,6,18,34,51]). Moreover, the integration of SAT-based complete algorithms and SLS algorithms has already been proposed for MaxSAT [28]. However, the proposed approaches for MaxSAT are mostly similar to a portfolio of solvers running in parallel or having some predefined criteria where either an SLS or a SAT solver are used. As a result, the integration of SLS and SAT-based techniques is limited.

In this paper, we propose an effective integration of SAT-based techniques in a SLS solver for MaxSAT. In our solver, the control of the solving process changes from SAT-based procedures to stochastic procedures and vice-versa. At each step, each procedure tries to build upon the information received from the other, instead of being independent procedures. The main contributions of the paper are as follows: (1) a new unsatisfiability-based algorithm to correct the SLS current assignment into a feasible solution, (2) a new improvement procedure based on Minimal Correction Subset (MCS) enumeration limited to the context of the SLS solver, and (3) an extensive experimental evaluation that shows the effectiveness of the newly proposed ideas.

The paper is organized as follows. Section 2 introduces the SAT and MaxSAT problems, as well as the notion of unsatisfiable cores and MCSes. Next, Sect. 3 reviews state of the art approximation algorithms for MaxSAT based on complete algorithms and SLS solvers. In Sect. 4, the new unsatisfiability-based procedure for assignment correction is presented, as well as a description of the assignment improvement procedure. Section 5 presents a comparison of our new solver with the top performing solvers on the incomplete track of the 2018 MaxSAT Evaluation. Finally, the paper concludes in Sect. 6.

2 Preliminaries

This section describes the Maximum Satisfiability (MaxSAT) problem, as well as the notions of unsatisfiable core and Minimal Correction Subsets (MCSes). Additional background information and definitions are also provided.

2.1 Maximum Satisfiability

A propositional formula in Conjunctive Normal Form (CNF), defined over a set $X = \{x_1, x_2, \ldots, x_n\}$ of n Boolean variables, is a conjunction of clauses, where a clause is a disjunction of literals. A literal is either a variable x_i or its complement $\bar{x}_i$. A complete assignment is a function $\nu : X \rightarrow \{0, 1\}$ that associates each variable in X with a Boolean value. Given an assignment ν, a literal x_i (respectively $\bar{x}_i$) is said to be satisfied if $\nu(x_i) = 1$ (respectively $\nu(x_i) = 0$). A clause is said to be satisfied by ν if any of its literals is satisfied. Otherwise, it is said to be unsatisfied. A formula ϕ is satisfied by ν if all its clauses are satisfied. On the other hand, if any of the clauses in ϕ is unsatisfied by ν, then ϕ is unsatisfied. Given a CNF formula ϕ, the Propositional Satisfiability (SAT) problem consists of finding a truth assignment ν such that ϕ is satisfied, or prove that no assignment exist that satisfies ϕ.

The Maximum Satisfiability (MaxSAT) problem is an optimization version of the SAT problem and several versions of MaxSAT can be used [35]. In the context of this paper, we focus on the partial MaxSAT problem where clauses in a CNF formula $\phi = \phi_h \cup \phi_s$ are labeled as hard (ϕ_h) or soft (ϕ_s). The goal of partial MaxSAT problems is to find an assignment ν that satisfies all hard clauses in ϕ_h, while minimizing the number of unsatisfied soft clauses in ϕ_s. In weighted partial MaxSAT problems, a positive integer weight is associated with each soft clause and the goal is to satisfy all hard clauses, while minimizing the total weight of unsatisfied soft clauses.

If an assignment ν satisfies all hard clauses, then we say that ν is a feasible assignment. Otherwise, we say that ν is infeasible. In this paper, it is assumed that the set of hard clauses ϕ_h can be satisfied, i.e. there is always a feasible assignment for a given MaxSAT problem instance. Otherwise, the MaxSAT formula would be unsatisfiable.

Throughout the paper, the set notation is used for clauses and CNF formulas. In particular, a CNF formula is seen as a set of clauses and a clause as a set of literals. Finally, we extend the notation of satisfiability of a clause and a set of clauses by an assignment ν. If c_i is a clause satisfied by ν, then $\nu(c_i) = 1$, otherwise $\nu(c_i) = 0$. Let ϕ denote a set of clauses. If assignment ν satisfies ϕ, then $\nu(\phi) = 1$, otherwise $\nu(\phi) = 0$.

Example 1. Consider the following weighted partial MaxSAT formula $\phi = \phi_h \cup \phi_s$ where $\phi_h = \{(x_1 \vee x_2 \vee \bar{x}_3), (x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)\}$ and $\phi_s = \{((\bar{x}_1), 1), ((\bar{x}_2), 3), ((\bar{x}_3), 1)\}$. Note that the positive weight associated with each soft clause denotes the cost of not satisfying the clause. In this case, the assignment $\nu = \{x_1 = 1, x_2 = 0, x_3 = 1\}$ is an optimal solution with a cost of 2, since soft clauses $(\bar{x}_1)$ and $(\bar{x}_3)$ are not satisfied by ν.

2.2 Unsatisfiable Cores and Minimal Correction Subsets

Let ϕ be an unsatisfiable formula. A subset $\phi_C \subseteq \phi$ is an unsatisfiable core of ϕ if and only if ϕ_C is also unsatisfiable. Several techniques exist in the literature for computing unsatisfiable cores (e.g. [15,22]) and current state of the art SAT solvers are able to identify an unsatisfiable core of ϕ.

Example 2. Consider the following CNF formula $\phi = \{(x_1 \vee x_2 \vee \bar{x}_3), (x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3), (\bar{x}_1), (\bar{x}_2), (\bar{x}_3)\}$. One unsatisfiable core of ϕ would be $\phi_C = \{(x_2 \vee x_3), (\bar{x}_2), (\bar{x}_3)\}$, since this subset of clauses of ϕ is unsatisfiable.

Let ϕ_h and ϕ_s be the sets of hard and soft clauses, respectively, such that ϕ_h is satisfiable and $\phi_h \cup \phi_s$ is unsatisfiable. A subset $C \subseteq \phi_s$ is a Minimal Correction Subset (MCS) if and only if $\phi_h \cup (\phi_s \setminus C)$ is satisfiable and $\phi_h \cup (\phi_s \setminus C) \cup \{c\}$ is unsatisfiable for all $c \in C$.

Observe that MCS algorithms [8,19,39,41] easily provide an approximation to the optimal solution of a MaxSAT instance. An MCS algorithm provides an assignment ν that satisfies $\phi_h \cup (\phi_s \setminus C)$. Let $f(C)$ denote the sum of the weights of the clauses in C. Since ν satisfies all hard clauses, its cost will be $f(C)$, thus providing an approximation to the optimum of the MaxSAT instance. In fact, solving a MaxSAT instance can be reduced to finding an MCS with minimum value of $f(C)$ [9].

Example 3. Consider again the weighted partial MaxSAT formula from Example 1, where $\phi_h = \{(x_1 \vee x_2 \vee \bar{x}_3), (x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)\}$ and $\phi_s = \{((\bar{x}_1), 1), ((\bar{x}_2), 3), ((\bar{x}_3), 1)\}$. This formula has two MCSs: $C_1 = \{(\bar{x}_1), (\bar{x}_3)\}$ and $C_2 = \{(\bar{x}_2)\}$. Observe that the cost of C_1 is 2, while the cost of C_2 is 3. Actually, an assignment that satisfies $\phi_h \cup (\phi_s \setminus C_1)$ is an optimal assignment of ϕ since C_1 is the lowest cost MCS. On the other hand, an assignment that satisfies $\phi_h \cup (\phi_s \setminus C_2)$ is an approximation on the optimum of ϕ.

3 Algorithms to Approximate MaxSAT

This section briefly reviews algorithms that can approximate the optimal solution of MaxSAT instances. First, we refer to complete SAT-based algorithms for MaxSAT that can be adapted to provide an approximate solution. Next, stochastic approaches are presented with focus on stochastic local search algorithms.

3.1 SAT-Based Algorithms

Current state-of-the-art complete algorithms for MaxSAT rely on iterative calls to a SAT solver. One possible approach is to use the linear Sat-Unsat algorithm that performs a linear search on the total weight of unsatisfied soft clauses. These algorithms start by solving the hard clauses using a SAT solver. Next, whenever a solution is found, a new pseudo-Boolean constraint[1] is added, such that solutions with a higher or equal cost are excluded. The algorithm stops when the SAT solver returns unsatisfiable. Hence, the last solution found is an optimal solution to the MaxSAT formula.

In large instances, the performance of these algorithms starts to degrade due to large weights in soft clauses, or when the number of soft clauses is very large. Recently, incomplete algorithms have been proposed where only a subset of soft

[1] In the case of partial MaxSAT instances, a cardinality constraint is used.

clauses is considered at each iteration, or the weights are approximated [14,27] to allow a more effective encoding of the Pseudo-Boolean or cardinality constraints.

While linear Sat-Unsat algorithms perform the search refining an upper bound on the optimal solutions, linear Unsat-Sat MaxSAT algorithms iteratively refine a lower bound [2,21,38]. In unsatisfiability-based MaxSAT algorithms, the lower bound is refined by iteratively finding unsatisfiable cores and the first satisfiable SAT call returns an optimal solution to the MaxSAT instance.

Unsatisfiability-based MaxSAT algorithms can also provide upper bounds [1,3,43] by applying a stratified approach, i.e. only a subset of soft clauses with higher weights are considered. The remaining soft clauses are added iteratively to the solver, after the subproblem considering higher weights has been solved. Observe that any MaxSAT algorithm that maintains an upper bound on the optimum can provide an approximate solution. Nevertheless, in many problem instances, it is hard to quickly find a good quality approximation to the optimum.

3.2 Stochastic Algorithms

Stochastic local search (SLS) algorithms for SAT and MaxSAT have been developed in the past [23,46,50]. These algorithms are inherently incomplete, since they are unable to prove unsatisfiability of SAT problems or prove that an assignment is an optimal solution to a MaxSAT instance. Nevertheless, for randomly generated instances, SLS algorithms have been shown to be very effective at finding very good approximations to the optimal solution. In fact, SLS algorithms have been used to quickly find a tight upper bound to MaxSAT instances so that a subsequent branch and bound algorithm could be more effective in pruning the search space [29].

Given a MaxSAT instance $\phi = \phi_h \cup \phi_s$, SLS algorithms start by defining a random assignment ν to all problem variables. While ν does not satisfy all hard clauses ϕ_h, an unsatisfied hard clause $c_i \in \phi_h$ is selected and ν is updated by flipping the value of a variable in c_i. Hence, c_i becomes satisfied by ν. Next, if ν satisfies all hard clauses, then the algorithm focus on minimizing the weight of unsatisfied soft clauses ϕ_s by flipping assignments in ν. There is a plethora of heuristics to implement this generic SLS approach. Recently, new SLS algorithms and techniques have been proposed such as `CCLS` [37], `CCEHC` [36], `Ramp` [17], and `maxroster` [47], among others [11,12,33]. In the MaxSAT Evaluation 2018, `SATLike` [33] was one of the best performing solvers in the incomplete solver track. This was particularly surprising, since no randomly generated instances were selected for the MaxSAT evaluation [7].

Algorithm 1 presents the pseudo-code for `SATLike` [33]. This algorithm maintains a weight associated to each hard clause in ϕ_h and each soft clause in ϕ_s. Initially, hard clauses have weight 1 and soft clauses are associated with its weight in the MaxSAT instance (line 1). After using a procedure based on unit propagation to compute an initial assignment to ν (line 2), the algorithm performs several iterations until a given cutoff limit is reached. At each iteration, if ν satisfies all hard clauses and improves upon the best previous assignment, then ν is saved (lines 5–6). Let $score(x_i)$ denote the weight increase in satisfied

Algorithm 1: SATLike Algorithm

```
   Input: $\phi = \phi_h \cup \phi_s$, $cutoff$
   Output: satisfying assignment to $\phi$
1  InitializeClauseWeights($\phi_h, \phi_s$)
2  $\nu \leftarrow$ InitializeAssignment($Vars(\phi)$)
3  $\nu_{best} \leftarrow \emptyset$
4  while ($\#iterations < cutoff$) do
5      if (($\nu(\phi_h) = 1$) $\wedge$ (Cost($\phi_s, \nu$) $<$ Cost($\phi_s, \nu_{best}$))) then
6          $\nu_{best} \leftarrow \nu$                          // A better solution is found
7      $D \leftarrow \{x_i \in Vars(\phi) | score(x_i) > 0\}$
8      if ($D \neq \emptyset$)) then
9          $x_s \leftarrow$ SelectBMS($D$)
10     else
11         UpdateClauseWeights($\phi_h, \phi_s, \nu$)
12         if ($\nu(\phi_h) = 1$) then
13             $c_s \leftarrow$ RandomSelect($\{c_i : c_i \in \phi_s \wedge \nu(c_i) = 0\}$)
14         else
15             $c_s \leftarrow$ RandomSelect($\{c_i : c_i \in \phi_h \wedge \nu(c_i) = 0\}$)
16         $x_s \leftarrow$ SelectMaxScore($Vars(c_s)$)
17     $\nu \leftarrow$ Flip($\nu, x_s$)                        // Flip value of $x_s$ in $\nu$
18 return ($\nu_{best}$)                      // Returns the best assignment found
```

clauses resulting from flipping x_i. If there are variables that would improve ν with respect to the current clause weight (i.e. variables with positive score), then a variable is selected to be flipped according to a *best from multiple selections (BMS)* strategy (line 9)[2]. Otherwise, the algorithm is at a local minima and the current clause weights are updated according to the strategy defined in [33] (line 11). Next, if ν satisfies all the hard clauses, then an unsatisfied soft clause is selected (line 13). Otherwise, an unsatisfied hard clause is selected instead (line 15). The variable to be flipped is the one with the highest score in the selected clause (line 16). Finally, the best solution found is returned when the cutoff limit is reached (line 18). The cutoff limit depends on a predefined maximum number of iterations without improvement. That is, the algorithm ends when it is unable to find a satisfiable assignment, or when it fails to improve upon the best feasible solution found, within a given number of iterations (in the implementation the limit is of 10^7 iterations). For this purpose, the iteration counter is set to zero whenever ν_{best} is updated.

4 Using SAT Techniques in Local Search

The idea of integrating SAT techniques in SLS algorithms for MaxSAT is not new. For example, solver MiniWalk [28] used SAT solver MiniSat [16] to guide the SLS algorithm WalkSAT [46]. In this case, the SLS algorithm and the SAT solver

[2] We refer to the literature for further details [10,33].

were run in parallel using a shared memory array such that the SLS algorithm would not flip a variable x_i if it would result in a complement assignment to the assignment in the SAT solver. The goal is to use the SAT solver to deal with the hard clauses, so that the SLS algorithm can focus on the optimization of the soft clauses.

Nevertheless, despite some exchange of information in MiniWalk, the SLS algorithm and the SAT solver are run in parallel and mostly in an independent fashion, similar to a parallel portfolio solver. In this paper, the goal is to have a SLS algorithm where SAT-based techniques are effectively used to correct and improve the current assignment in the SLS algorithm. Although correction procedures have already been proposed in evolutionary algorithms for multi-objective optimization [24], this paper proposes a novel procedure where unsatisfiable cores are used to identify sets of variable assignments that need to be changed.

Let $\texttt{SAT}(\phi, \mathcal{A}, \mathit{budget})$ denote a call to a SAT solver where ϕ is a CNF formula, $\mathcal{A}$ is a set of literals considered as assumptions, and *budget* is a positive value. A SAT solver call returns a triple (st, ϕ_C, ν) where st denotes the solver return status ($\texttt{SAT}, \texttt{UNSAT}$ or $\texttt{UNRES}$). If the solver returns `SAT`, then ν contains a satisfiable assignment to ϕ. On the other hand, if the solver returns `UNSAT`, then ϕ_C contains an unsatisfiable core. Note that ϕ might be satisfiable, but the solver might still return `UNSAT` due to the set of assumptions $\mathcal{A}$. This occurs when there are no models of ϕ where all assumption literals in $\mathcal{A}$ are satisfied. Therefore, ϕ_C might contain a subset of clauses from ϕ and literals from $\mathcal{A}$. Finally, the solver returns `UNRES` if during the SAT call the number of conflicts reaches the defined *budget*. Observe that if *budget* is set to $+\infty$, then the SAT call does not return `UNRES`. However, in our context, a conflict limit will be set to avoid the solver to take too much time in a SAT call.

One of the shortcomings of SLS algorithms is that these solvers have difficulties in dealing with highly constrained formulas. Therefore, it might be the case that the SLS algorithm is unable to satisfy ϕ_h or gets stuck in some local minima. In these cases, using SAT-based techniques to find a satisfiable assignment to ϕ_h would be beneficial.

4.1 Assignment Correction

Consider the case when the SLS algorithm is unable to change from an unsatisfiable assignment ν into a better assignment. Algorithm 2 describes our unsatisfiability-based algorithm which performs a correction to ν in order to guide the SLS algorithm to the feasible region of the search space.

First, we start by building a set of assumption literals $\mathcal{A}$ corresponding to the assignment ν (lines 1–3). Next, a SAT call on the set of hard clauses ϕ_h is made (line 4). Clearly, if ν is not feasible, then this call returns `UNSAT` and ϕ_C contains an unsatisfiable core. Therefore, while a satisfiable assignment is not found, the assumption literals that occur in ϕ_C are deemed responsible for the `UNSAT` status, removed from $\mathcal{A}$ (line 7) and a new SAT call is made (line 8). Observe that a conflict limit is defined for the correction procedure. Hence, after

Algorithm 2: Assignment Correction Algorithm

```
   Input: $\phi = \phi_h \cup \phi_s$, $\nu$, $confLimit$
   Output: satisfying assignment to $\phi$
1  $\mathcal{A} \leftarrow \emptyset$
2  foreach ($x_i \in Vars(\phi)$) do
3      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\nu(x_i) = 1\ ?\ x_i : \bar{x}_i)\}$
4  $(\mathsf{st}, \phi_C, \nu_{new}) \leftarrow \mathtt{SAT}(\phi_h, \mathcal{A}, confLimit)$
5  $confBudget \leftarrow confLimit - satSolverConflicts$
6  while ($\mathsf{st} \neq SAT \wedge confBudget > 0$) do
7      $\mathcal{A} \leftarrow \{l_j : l_j \in \mathcal{A} \wedge l_j \notin \phi_C\}$        // remove literals in unsat core
8      $(\mathsf{st}, \phi_C, \nu_{new}) \leftarrow \mathtt{SAT}(\phi_h, \mathcal{A}, confBudget)$
9      $confBudget \leftarrow confBudget - satSolverConflicts$
10 if ($\mathsf{st} \neq SAT$) then
11     $confBudget \leftarrow confLimit/10$
12     while ($\mathsf{st} \neq SAT \wedge |\mathcal{A}| > 0$) do
13         $\mathcal{A} \leftarrow \mathtt{ChooseRandom}(\mathcal{A}, 0.5)$
14         $(\mathsf{st}, \phi_C, \nu_{new}) \leftarrow \mathtt{SAT}(\phi_h, \mathcal{A}, confBudget)$
15 $\nu_{new} \leftarrow \mathtt{Improve}(\phi, \mathcal{A}, \nu_{new}, confLimit)$            // MCS Enumeration
16 return ($\nu_{new}$)                              // Returns the best assignment found
```

each SAT call, the conflict budget is reduced by the number of conflicts in the last SAT call.

Note that if ϕ_h is satisfiable, then a satisfiable assignment is eventually found. However, since the number of conflicts is limited at each SAT call, it is possible that the conflict budget is not enough to find a satisfiable assignment. If this is the case, then our algorithm applies a similar procedure with a more aggressive strategy (lines 12–14) where at each iteration 50% of the literals in $\mathcal{A}$ are removed (line 13). Since the correction procedure only depends on the hard clauses, there is no guarantee regarding its quality. As a result, we also apply a SAT-based improvement procedure (line 15) detailed in Algorithm 3.

4.2 Assignment Improvement

Let $\phi = \phi_h \cup \phi_s$ be a MaxSAT formula and $\mathtt{MCS}(\phi_h, \phi_s, budget)$ denote a call to an MCS algorithm where *budget* is a positive value. An MCS solver call returns a pair (st, ν) where *st* denotes the return status. If the return status *st* is `SAT`, then ν denotes an assignment that satisfies $\phi_h \cup (\phi_s \setminus C)$ where C is an MCS of ϕ. Therefore, ν provides an approximation to the optimal solution of ϕ (see Sect. 2.2). Otherwise, either *st* is `UNSAT` if ϕ_h is not satisfiable or *st* is `UNRES` if the budget conflict limit is reached.

Algorithm 3 describes our improvement algorithm. Given a MaxSAT instance ϕ, a set of assumptions $\mathcal{A}$, a satisfiable assignment ν, and the conflict budget *ConfBudget*, the goal of this algorithm is to find a better quality solution for ϕ through an MCS enumeration procedure.

Algorithm 3: Assignment Improvement Algorithm using MCS enumeration

```
   Input: $\phi = \phi_h \cup \phi_s$, $\mathcal{A}$, $\nu$, $confBudget$
   Output: satisfying assignment to $\phi$
 1 $\nu_{new} \leftarrow \nu$
 2 $\phi_w \leftarrow \phi_h \cup \{(l_j) : l_j \in \mathcal{A}\}$
 3 while $(confBudget > 0)$ do
 4     $(\mathsf{st}, \nu) \leftarrow$ MCS($\phi_w, \phi_s, confBudget$)
 5     if $((\mathsf{st} = SAT) \land (\texttt{Cost}(\phi_s, \nu) < \texttt{Cost}(\phi_s, \nu_{new})))$ then
 6         $\nu_{new} \leftarrow \nu$
 7     if $(\mathsf{st} = UNSAT)$ then
 8         return $(\nu_{new})$                                  // All MCSs found
 9     $\phi_w \leftarrow \phi_w \cup$ BlockingClause($\phi_w, \phi_s, \nu$)
10     $confBudget \leftarrow confBudget - mcsSolverConflicts$
11 return $(\nu_{new})$                          // Returns the best assignment found
```

Algorithm 4: Assignment Improvement Algorithm using Linear Sat-Unsat

```
   Input: $\phi = \phi_h \cup \phi_s$, $\mathcal{A}$, $\nu$, $confBudget$
   Output: satisfying assignment to $\phi$
 1 $\phi_w \leftarrow \phi_h \cup \{(l_j) : l_j \in \mathcal{A}\}$
 2 $(\mathsf{st}, \nu_{new}) \leftarrow$ LinearSat-Unsat($\phi_w, \phi_s, confBudget$)
 3 if $((\mathsf{st} = SAT) \land (\texttt{Cost}(\phi_s, \nu) < \texttt{Cost}(\phi_s, \nu_{new})))$ then
 4     return $\nu_{new}$                // Linear Sat-Unsat found a better solution
 5 else
 6     return $\nu$
```

The algorithm starts by building a working formula ϕ_w from the set of hard clauses ϕ_h and the set of assumptions $\mathcal{A}$ (line 2). Next, the algorithm iterates over all MCSes of ϕ, constrained to the set of assumptions $\mathcal{A}$ and returns the best assignment found (lines 3–10). Each time a new MCS is found, a new clause is added to ϕ_w to prevent the enumeration of the same MCS later on. This new clause (also known as blocking clause) forces at least one of the current variable assignments to have the opposite value (line 9). Finally, observe that, at each iteration, the conflict budget decreases depending on the number of conflicts used in the MCS algorithm.

Notice that the set of literals $\mathcal{A}$ restricts the MCS enumeration procedure. As a result, Algorithm 3 performs a localized MCS enumeration. Moreover, there is no guarantee that the MCSes found by this procedure are MCSes of the original MaxSAT formula ϕ, since the literals in $\mathcal{A}$ must all be satisfied in each MCS call. The main idea is to quickly perform a localized improvement in order to find a better solution than ν.

Many different improvement procedures can be devised, including the usage of complete methods. Algorithm 4 is an alternative to the improvement algorithm where the MCS enumeration is replaced with a call to a Linear Sat-Unsat algorithm (LSU). Observe that the call to the LSU algorithm is limited to a number of conflicts (line 2). Additionally, all literals in $\mathcal{A}$ are forced to be satisfied. Hence, the LSU call is also restricted to a localized region of the search space. If the LSU algorithm finds a feasible assignment, then *st* equals `SAT`. In that case, we check whether the assignment found by the LSU algorithm improves upon the previous solution ν and the best solution is returned. Finally, we note that any complete MaxSAT algorithm that is able to produce an approximation to the optimal solution (see Sect. 3.1) could be used instead of LSU.

4.3 Solvers

Two new solvers were developed: `sls-mcs` and `sls-lsu`. In `sls-mcs`, the `SATLike` solver (Algorithm 1)[3] is extended with the assignment correction algorithm (Algorithm 2) and the assignment improvement algorithm based on MCS enumeration (Algorithm 3). The difference from `sls-mcs` to `sls-lsu` is on the assignment improvement algorithm. In `sls-lsu`, the linear sat-unsat assignment improvement algorithm (Algorithm 4) is used.

Both `sls-mcs` and `sls-lsu` use the Glucose SAT solver (version 4.1) on the assignment correction procedure. Moreover, the CLD [39] algorithm is used as the MCS algorithm in `sls-mcs`. However, for weighted instances, the stratified CLD algorithm [49] is used. In `sls-lsu`, the linear sat-unsat algorithm is the one available at the `open-wbo` open source MaxSAT solver. In both `sls-mcs` and `sls-lsu`, the assignment correction/improvement algorithm is called just before line 5 in Algorithm 1 if SATlike has reached half of the maximum number of iterations without improvement. In such a case, the correction algorithm is called if the current assignment ν does not satisfy all hard clauses, otherwise the improvement algorithm is directly called with approximately half of the literals in the current assignment ν as assumptions. These assumption literals are randomly chosen from ν using the same procedure as in line 13 in Algorithm 2. Note that the iteration counter in Algorithm 1 is set to zero if ν_{best} is updated after the call to the correction/improvement algorithm.

5 Experimental Results

This section evaluates the effectiveness of the ideas proposed in the paper. The `SATLike` solver serves as our baseline solver. Nevertheless, we also compare `sls-mcs` and `sls-lsu` against the best performing solvers at the incomplete track of the last MaxSAT evaluation. No complete solver is included in this comparison because our preliminary results show that running LSU or enumerating MCSes can be hard for several of the instances used, which led to a poor performance. The solvers used in our experimental evaluation are as follows:

- `SATLike`: Stochastic local search solver described in Algorithm 1 [33].

[3] The source code of `SATLike` is publicly available at the 2018 MaxSAT evaluation https://maxsat-evaluations.github.io/2018/descriptions.html.

- `SATLike-c`: Version of `SATLike` submitted to the 2018 MaxSAT Evaluation. Initially, the `SATLike` algorithm is applied. If during the first 50 s, `SATLike` does not find a feasible solution, then the Linear Sat-Unsat complete algorithm from the `open-wbo` solver is used [32].
- `LinSBPS`: Linear sat-unsat algorithm with solution phase saving. In weighted instances, the algorithm starts by building a MaxSAT formula where all soft clause weights are divided by a large constant β. After finding an optimal solution for this formula, a new formula is build where the weights are divided by a new constant β' such that $\beta' < \beta$. The process is repeated until the original formula is solved ($\beta = 1$) [14].
- `maxroster`: This solver starts by applying the stochastic local search solver `Ramp` [17] for 6 s. Next, a complete MaxSAT solver is applied. MSU3 is used for partial MaxSAT, while OLL is used for weighted instances [47].
- `Open-WBO-Inc`: Another two-stage solver that starts by applying an incomplete algorithm, followed by the complete linear sat-unsat procedure of the `open-wbo` solver. For unweighted instances, the incomplete solver can be based on MCSes (`Open-WBO-Inc-MCS`) or based on bit vector optimization (`Open-WBO-Inc-OBV`). For weighted instances, the incomplete solver can be based on modifications on the weights of soft clauses and clustering (`Open-WBO-Inc-Cluster`) or partitioning of soft clauses (`Open-WBO-Inc-BMO`) [26].

All experimental results were obtained on a server with processor Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 64GB of memory. The benchmark set corresponds to the one used in the 2018 MaxSAT evaluation for the incomplete track[4]. The benchmark set contains 153 partial MaxSAT problem instances, and 172 weighted partial MaxSAT problem instances. As in the 2018 MaxSAT competition, two time limits were considered: 60 s and 300 s. For each time limit, each solver was executed 7 times with each instance. Whenever `satlike-c`, `satlike`, `sls-mcs`, and `sls-slu` algorithms reach the *cutoff* stopping criteria before the time limit runs out, the algorithm is called again, and the best solution found among all calls is returned. The conflict limits of the correction and the improvement algorithms in `sls-mcs` and `sls-lsu` were set to 10^5.

5.1 Partial MaxSAT

Table 1 shows the number of instances for which the final solution of `sls-mcs` and `sls-lsu` solvers were produced by the local search part, and how many were produced by the correction and the improvement part. Tables 2 and 3 summarize the pairwise comparisons between solvers for the 60 and 300-s time limit scenarios, respectively. Two variants of the MCS-based local search solver are considered, one as described in Sect. 4.3 (`sls-mcs`), and another (`sls-mcs2`) that does not consider the assumptions $\mathcal{A}$ as hard clauses in the the MCS enumeration procedure, i.e., it implements Algorithm 3 with line 2 replaced by $\phi_w \leftarrow \phi_h$. Each one of the new solvers (`sls-mcs`, `sls-mcs2` and `sls-lsu`) is compared against

[4] Instances available at https://maxsat-evaluations.github.io/2018/.

Table 1. Number of instances for which the best solution was produced by the local search (`sls`), and by the correction/improvement algorithm (`mcs`/`lsu`).

Time limit	`sls-mcs`		`sls-lsu`	
	`sls`	`mcs`	`sls`	`lsu`
60 s	62	53	65	49
300 s	48	86	50	84

Table 2. Partial MaxSAT. Versus table (row wins, ties, column wins). Time limit 60 s.

	`sls-mcs`	`sls-mcs2`	`sls-lsu`
`satlike`	(19,71,**63**)	(24,88,**41**)	(23,78,**52**)
`satlike-c`	(40,70,**43**)	(**52**,75,26)	(**50**,75,28)
`LinSBPS`	(52,23,**78**)	(57,23,**73**)	(54,24,**75**)
`maxroster`	(39,36,**78**)	(44,36,**73**)	(40,37,**76**)
`Open-WBO-Inc-mcs`	(43,20,**90**)	(45,22,**86**)	(47,19,**87**)
`Open-WBO-Inc-obv`	(50,19,**84**)	(51,20,**82**)	(51,21,**81**)
`sls-mcs`	–	(**42**,90,21)	(**29**,113,11)
`sls-mcs2`	(21,90,**42**)	–	(23,97,**33**)
`sls-lsu`	(11,113,**29**)	(**33**,97,23)	–

every other solver considering the median value obtained for each instance. Each table cell contains a triple, (b, e, w), that represents the number of instances for which the solver in that row found a better (b), equal (e), or worse (w) quality solution than the solver in that column. Note that when both solvers are unable to find a feasible assignment, that fact is counted as a tie (e).

Table 1 shows that the correction and the improvement algorithms contribute with almost the same amount of final solutions as the local search part in the 60-s scenario, and almost twice as much in the 300-s scenario. Compared to one another, the `mcs` and the `slu`-based improvement algorithms contribute with nearly the same amount of final solutions to `sls-mcs` and `sls-lsu` solvers, respectively. Tables 2 and 3 show that these solvers found equally good solutions for about two thirds of the instances, while for most of the remaining ones, `sls-mcs` found better solutions than `sls-lsu`. In comparison to the other solvers, the number of times `sls-mcs` outperformed the other solvers was always higher than the number of times `sls-lsu` did. However, compared to most solvers in the 60-s scenario, the difference is very small - it is of only 1 to 3 instances. This means that, for most of the instances for which `sls-mcs` finds a better solution than `sls-lsu`, either `sls-lsu` provides a solution that is also better than the

Table 3. Partial MaxSAT. Versus table (row wins, ties, column wins). Time limit 300 s.

	sls-mcs	sls-mcs2	sls-lsu
satlike	(23,69,**61**)	(29,64,**60**)	(30,69,**54**)
satlike-c	(**48**,68,37)	(**50**,66,37)	(**62**,69,22)
LinSBPS	(53,23,**77**)	(46,34,**73**)	(56,25,**72**)
maxroster	(58,29,**66**)	(**60**,34,59)	(**66**,30,57)
Open-WBO-Inc-mcs	(31,17,**105**)	(27,21,**105**)	(42,18,**93**)
Open-WBO-Inc-obv	(35,17,**101**)	(33,21,**99**)	(44,20,**89**)
sls-mcs	–	(**43**,76,34)	(**40**,106,7)
sls-mcs2	(34,76,**43**)	–	(**51**,79,23)
sls-lsu	(7,106,**40**)	(23,79,**51**)	–

solution found by other solvers, or the solution found by sls-mcs is still not good enough.

The sls-mcs solver clearly improves upon satlike, as it obtained better quality solutions in about 60 instances in both 60 and 300-s scenarios and was worse than satlike in less than 24 instances. Of those 60 instances, satlike was unable to find a feasible solution in about 20 and 30 of them, for the 60 and the 300-s scenario, respectively. This means that not only the correction algorithm was able to help the local search algorithm reach the feasible region, but also the improvement algorithm helped finding better feasible solutions. Compared to satlike-c, sls-mcs is competitive in the 60-s scenario, and is slightly worse in the 300-s scenario. This is not surprising as in the cases where satlike cannot find a feasible solution in the first 48 s, the additional 4 min are fully used by the complete solver. Comparing the two MCS-based local search solvers, sls-mcs had a better performance than sls-mcs2.

Comparing to any other solver in the 60-s scenario, sls-mcs and sls-lsu outperformed all of them. Apart from maxroster, that becomes more competitive, and satlike-c, this remains true for the 300-s scenario.

Overall, the results show that both the correction and the improvement algorithms are useful to the local search algorithm. The former plays an important role for highly constrained problems, for which a feasible solution is difficult to find through local search, and the latter can improve even further upon the solution found.

5.2 Weighted Partial MaxSAT

As the MCS-based local search solver achieved better results for the partial MaxSAT problem than the one based on LSU, only the former was tested for the weighted partial MaxSAT problem. In this scenario, a stratified MCS algorithm is used to enumerate MCSes, where the step of partitioning the set of soft clauses

Table 4. Number of instances of WPMS for which the best solution was produced by the local search (`sls`), and by the correction/improvement algorithm (`mcs`/`mcs2`).

Time limit	`sls-mcs`		`sls-mcs2`	
	`sls`	`mcs`	`sls`	`mcs2`
60 s	72	70	54	88
300 s	59	93	44	110

Table 5. Weighted partial MaxSAT. Versus table (row wins, ties, column wins). Time limit 60 s.

	`sls-mcs`	`sls-mcs2`
`satlike`	(31,91,**50**)	(27,66,**79**)
`satlike-c`	(**52**,75,45)	(40,52,**80**)
`LinSBPS`	(**108**,9,55)	(**104**,13,55)
`maxroster`	(**91**,23,58)	(**77**,23,72)
`Open-WBO-Inc-BMO`	(**99**,10,63)	(**89**,17,66)
`Open-WBO-Inc-cluster`	(70,10,**92**)	(41,15,**116**)
`sls-mcs`	–	(27,79,**66**)
`sls-mcs2`	(**66**,79,27)	–

is performed only once at the beginning. Tables 4, 5 and 6, are analogous to Tables 1, 2 and 3, respectively.

As in the partial MaxSAT problem, the correction and improvement algorithms contribute directly to solutions reported by the solver(s). They are responsible for at least half, and up to two thirds, of the solutions reported by the solvers (see Table 4). Solvers `sls-mcs` and `sls-mcs2` had similar performance in about half of the instances in the 60-s scenario, and in about one third of the instances in the 300-s scenario (see Tables 5 and 6). Moreover, `sls-mcs2` found better solutions than `sls-mcs` in about two thirds of the remaining instances in both scenario. The correction/improvement algorithms in `sls-mcs` work in a more restricted search space than in `sls-mcs2` because they start with an already defined partial assignment (through the assumptions). This is advantageous when SATlike's current assignment is reasonably good, but when SATlike does not perform so well (in the weighted scenario), it seems more advantageous not to consider its current assignment (as in `sls-mcs2`).

Compared to `satlike`, and despite `sls-mcs` performing better in the 60-s scenario, contrary to what was expected its performance decayed in the 300-s scenario. Conversely, `sls-mcs2` performed much better than `satlike`, and even

Table 6. Weighted partial MaxSAT. Versus table (row wins, ties, column wins). Time limit 300 s.

	`sls-mcs`	`sls-mcs2`
`satlike`	(**70**,40,62)	(51,32,**89**)
`satlike-c`	(**90**,32,50)	(62,24,**86**)
`LinSBPS`	(**111**,12,49)	(**98**,19,55)
`maxroster`	(**97**,19,56)	(**81**,28,63)
`Open-WBO-Inc-BMO`	(**102**,8,62)	(**85**,16,71)
`Open-WBO-Inc-cluster`	(68,9,**95**)	(37,16,**119**)
`sls-mcs`	–	(36,46,**90**)
`sls-mcs2`	(**90**,46,36)	–

better than `satlike-c`. Compared to the other solvers in the two time-limit scenarios, `sls-mcs` showed a weaker performance, except when compared to `Open-WBO-Inc-cluster`. On the other hand, `sls-mcs2` had, in general, a better performance but still only outperforms `Open-WBO-Inc-cluster`.

Overall, `sls-mcs` has a poor performance, particularly against `satlike`. On the other hand, `sls-mcs2` showed a performance superior to `sls-mcs`, and was more competitive to the remaining solvers. This contrast may be indicative that the local search part is being too biased towards some regions of the search space, and that may be restricting too much the search space of the improvement algorithm after the assignment correction step. The inferior performance of `satlike` and `satlike-c` in the 2018 MaxSAT evaluation for weighted MaxSAT problem instances reinforces the conjecture. Moreover, `sls-mcs` does not make full use of stratification, as it does not take into account that, by forcing the assumption to be satisfied, some of the soft clauses are also satisfied. Thus, considering only the remaining soft clauses in the stratification process should lead to better results.

6 Conclusions and Future Work

In this paper, we propose the integration of SAT-based algorithms into a state of the art SLS solver for MaxSAT, where the solving process changes iteratively between the SLS and SAT-based procedures. A novel algorithm based on the identification of unsatisfiable cores is used for assignment correction of the SLS procedure. As a result, the SLS solver is guided into the feasible area of the search space, thus improving the search process in the SLS solver. Moreover, assignment improvement procedures are also devised and integrated into the SLS solver. Experimental results show the effectiveness of our approach, as the new incomplete MaxSAT solver is able to quickly find better approximations for a larger number of problem instances than other state of the art incomplete MaxSAT solvers.

For future work, we plan to extend the usage of unsatisfiable cores in SLS solvers, since other procedures can be devised where the unsatisfiable cores would guide the SLS algorithm. Furthermore, a more dynamic interaction between the SLS procedure and the SAT-based procedure should be tried. Finally, current results show that SLS algorithms for weighted MaxSAT can be greatly improved. Currently, SLS solvers still spend many iterations trying to satisfy the hard clauses. However, a tighter integration of SAT-based procedures would enable the SLS algorithm to focus on the optimization part of the problem.

Acknowledgments. This work was supported by national funds through FCT with references UID/CEC/50021/2019, PTDC/CCI-COM/31198/2017 and DSAIPA/AI/0044/2018.

References

1. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving WPM2 for (Weighted) partial MaxSAT. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 117–132. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40627-0_12
2. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (Weighted) partial MaxSAT through satisfiability testing. In: Kullmann [30], pp. 427–440
3. Ansótegui, C., Gabàs, J.: WPM3: an (in)complete algorithm for weighted partial maxsat. Artif. Intell. **250**, 37–57 (2017)
4. Asín, R., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and MaxSAT. Ann. Oper. Res. **218**(1), 71–91 (2014)
5. Audemard, G., Lagniez, J.-M., Mazure, B., Saïs, L.: Boosting local search thanks to CDCL. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 474–488. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_34
6. Audemard, G., Simon, L.: GUNSAT: a greedy local search algorithm for unsatisfiability. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 2256–2261 (2007)
7. Bacchus, F., Järvisalo, M.J., Martins, R., et al.: MaxSAT evaluation 2018 (2018)
8. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2005. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30557-6_14
9. Birnbaum, E., Lozinskii, E.: Consistent subsets of inconsistent systems: structure and behaviour. J. Exp. Theor. Artif. Intell. **15**(1), 25–46 (2003)
10. Cai, S.: Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
11. Cai, S., Luo, C., Thornton, J., Su, K.: Tailoring local search for partial maxsat. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada, 27–31 July 2014, pp. 2623–2629. AAAI Press (2014)
12. Cai, S., Luo, C., Zhang, H.: From decimation to local search and back: a new approach to maxsat. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 571–577 (2017). ijcai.org

13. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39071-5_13
14. Demirovic, E., Stuckey, P.J.: LinSBPS. MaxSAT Evaluation 2018: Solver and Benchmark Descriptions, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki, pp. 8–9 (2018)
15. Dershowitz, N., Hanna, Z., Nadel, A.: A scalable algorithm for minimal unsatisfiable core extraction. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 36–41. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_5
16. Een, N.: MiniSat: a sat solver with conflict-clause minimization. In: Proceedings SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing, pp. 502–518 (2005)
17. Fan, Y., Ma, Z., Su, K., Sattar, A., Li, C.: Ramp: a local search solver based on make-positive variables. MaxSAT Evaluation (2016)
18. Fang, L., Hsiao, M.S.: A new hybrid solution to boost SAT solver performance. In: Design, Automation and Test in Europe Conference, pp. 1307–1313 (2007)
19. Felfernig, A., Schubert, M., Zehentner, C.: An efficient diagnosis algorithm for inconsistent constraint sets. Artif. Intell. Eng. Des. Anal. Manuf. **26**(1), 53–62 (2012)
20. Feng, Y., Bastani, O., Martins, R., Dillig, I., Anand, S.: Automated synthesis of semantic malware signatures using maximum satisfiability. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, 26 February–1 March 2017. The Internet Society (2017)
21. 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
22. Goldberg, E.I., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Conference and Exposition on Design, Automation and Test in Europe, pp. 10886–10891 (2003)
23. Gu, J.: Efficient local search for very large-scale satisfiability problems. ACM SIGART Bull. **3**(1), 8–12 (1992)
24. Henard, C., Papadakis, M., Harman, M., Traon, Y.L.: Combining multi-objective search and constraint solving for configuring large software product lines. In: International Conference on Software Engineering, pp. 517–528 (2015)
25. Jose, M., Majumdar, R.: Cause clue clauses: error localization using maximum satisfiability. In: Programming Language Design and Implementation, pp. 437–446. ACM (2011)
26. Joshi, S., Kumar, P., Manquinho, V., Martins, R., Nadel, A., Rao, S.: Open-WBO-Inc in MaxSAT evaluation 2018. MaxSAT Evaluation 2018: Solver and Benchmark Descriptions, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki, pp. 16–17 (2018)
27. 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
28. Kroc, L., Sabharwal, A., Gomes, C.P., Selman, B.: Integrating systematic and local search paradigms: a new strategy for MaxSAT. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, 11–17 July 2009, pp. 544–551 (2009)
29. Kugel, A.: akmaxsat and akmaxsat_ls solver description. Technical report, MaxSAT Evaluation 2012 Solver Descriptions (2012)

30. Kullmann, O. (ed.): International Conference on Theory and Applications ofSatisfiability Testing, LNCS, vol. 5584. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2
31. Lang, J. (ed.): Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, 13–19 July 2018, Stockholm, Sweden (2018). ijcai.org
32. Lei, Z., Cai, S.: SATlike-c. MaxSAT Evaluation 2018: Solver and Benchmark Descriptions, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki, pp. 24–25 (2018)
33. Lei, Z., Cai, S.: Solving (weighted) partial maxsat by dynamic local search for SAT. In: Lang [33], pp. 1346–1352
34. Letombe, F., Marques-Silva, J.: Hybrid incremental algorithms for booleansatisfiability. Int. J. Artif. Intell. Tools **21**(6) (2012). https://doi.org/10.1142/S021821301250025X
35. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, pp. 613–631. IOS Press (2009)
36. Luo, C., Cai, S., Su, K., Huang, W.: CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artif. Intell. **243**, 26–44 (2017)
37. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Computers **64**(7), 1830–1843 (2015)
38. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for Weighted Boolean Optimization. In: Kullmann [30], pp. 495–508
39. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On Computing Minimal Correction Subsets. In: International Joint Conference on Artificial Intelligence, pp. 615–622 (2013)
40. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_39
41. Mencía, C., Previti, A., Marques-Silva, J.: Literal-based MCS extraction. In: International Joint Conference on Artificial Intelligence, pp. 1973–1979 (2015)
42. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_41
43. Morgado, A., Ignatiev, A., Marques-Silva, J.: MSCG: robust core-guided maxsat solving. JSAT **9**, 129–134 (2014)
44. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: AAAI Conference on Artificial Intelligence, pp. 2717–2723. AAAI Press (2014)
45. Safarpour, S., Mangassarian, H., Veneris, A.G., Liffiton, M.H., Sakallah, K.A.: Improved design debugging using maximum satisfiability. In: Formal Methods in Computer-Aided Design, pp. 13–19. IEEE Computer Society (2007)
46. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In: Johnson, D.S., Trick, M.A. (eds.) Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, 11–13 October 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 521–532. DIMACS/AMS (1993)
47. Sugawara, T.: Maxroster: solver description. MaxSAT Eval. **2017**, 12 (2017)
48. Terra-Neves, M., Machado, N., Lynce, I., Manquinho, V.: Concurrency debugging with maxSMT. In: AAAI Conference on Artificial Intelligence. AAAI Press (2019)

49. Terra-Neves, M., Lynce, I., Manquinho, V.M.: Stratification for constraint-based multi-objective combinatorial optimization. In: Lang [31], pp. 1376–1382
50. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: The Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, 10–13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004)
51. Zhang, J., Zhang, H.: Combining local search and backtracking techniques for constraint satisfaction. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, pp. 369–374 (1996)