

Structure-guided Local Improvement for Maximum Satisfiability

André Schidler^[0000-0001-6790-7158] and Stefan Szeider^[0000-0001-8994-1656]

TU Wien, Vienna, Austria
`{aschidler,sz}@ac.tuwien.ac.at`

Abstract. The enhanced performance of today’s MaxSAT solvers has elevated their appeal for many large-scale applications, notably in software analysis and computer-aided design. Our research delves into refining anytime MaxSAT solving by repeatedly identifying and solving smaller subinstances that are chosen based on the graphical structure of the instance with a complete solver. We investigate various strategies to pinpoint these subinstances. This structure-guided selection of subinstances provides a complete solver with a high potential for improving the current solution. Our exhaustive experimental analyses contrast our methodology as instantiated in our tool MaxSLIM with previous studies and benchmark it against leading-edge MaxSAT solvers.

1 Introduction

MaxSAT solvers (solvers for the partial weighted maximum satisfiability problem) have proven to be indispensable tools with an expansive range of applications, including problems that arise in software analysis [33], post-silicon fault localization [34], the identification of concurrency bugs and suggestions for fixes [12], and malware detection in smartphone apps [8]. Additional applications and case studies [18, 20] highlight MaxSAT’s versatility in computer-aided design and related areas. While the focus in the past was mainly on creating superior exact MaxSAT solvers tailored for identifying optimal solutions, there has been a noticeable shift towards the significance of anytime MaxSAT solvers in recent times. Unlike exact solvers that seek optimal results, anytime solvers prioritize finding commendable solutions in a shorter time frame and, when interrupted, output the best solution found so far. Bacchus and Hickey [10] introduced a technique using Large Neighborhood Search [23, 32] that integrates the capabilities of both exact and anytime solvers, harnessing the advantages of both.

In our paper, we further develop the methodology of combining exact and anytime MaxSAT solvers, leveraging the local characteristics of the MaxSAT instance’s graphical structure to direct the search and dissection of the problem. We embody our innovative approach in the tool MaxSLIM, which supports an array of strategies for choosing local substances grounded on the incidence graph’s graphical structure of the MaxSAT instance. Each strategy starts from a small number of variables and strategically extends the set of variables by tracing out a subgraph in the graphical model until a predefined budget is reached.

These selection strategies use a preference metric to select the starting variable and to decide which variable to add next. We consider six different metrics in our framework. MaxSLIM uses a new MaxSAT solver specifically designed for solving local instances, or, alternatively, can use any of the currently available MaxSAT solvers for this purpose. Another compelling feature of MaxSLIM is that if chosen appropriately, several local instances can be solved in parallel.

Our comprehensive experimental evaluation on the instances from the 2023 MaxSAT Evaluation¹ showcases the efficacy of considering the graphical structure in the MaxSAT instances. The insights derived from a wide-ranging set of experiments conclusively indicate that integrating the graphical structure not only elevates the performance but does so in a significant and robust manner. The consistent and notable performance enhancements highlight the value and effectiveness of this integrated approach.

1.1 Related Work

As mentioned above, Hickey and Bacchus [10] proposed an anytime MaxSAT solver based on the Large Neighborhood Search (LNS) metaheuristic [23, 32] called MaxSAT-LNS. Similar to MaxSLIM, MaxSAT-LNS also first computes an initial (possibly suboptimal) solution and then tries to improve this solution using LNS. In each round of LNS, a random subset of variables is selected, and their assignment is fixed according to the initial solution, after which a separate solver is run to find an improved assignment for the remaining variables. Already fixing a few assignments can imply many other assignments via unit propagation, making the problem of completing the assignment comparatively easy. The main difference to MaxSLIM is that MaxSAT-LNS selects subinstances without utilizing the graphical structure of the entire instance.

SAT-Based Local Improvement (SLIM) is a specific type of LNS [23, 32] tailored for the use with (Max)SAT solvers. As such, SLIM is an anytime metaheuristic that embeds (Max)SAT encodings into heuristic algorithms. In the past, SLIM has been used for a variety of problems. Some of these instantiations used SAT solvers for solving local instances, while others used MaxSAT solvers or QBF solvers. We list some instantiations below along with the respective solvers used:

- graph decomposition problems [9, 14, 15, 24] (SAT),
- Bayesian Network structure learning [25–27] (MaxSAT),
- decision tree induction [30] (SAT),
- graph coloring [29, 31] (SAT), and
- circuit minimization [13, 28] (QBF).

The common aspect between all these SLIM instantiations is that the initial solution (to be improved by iteratively solving local instances) was too large to be computed directly by a SAT-based solver. Hence initial solutions were

¹ <https://maxsat-evaluations.github.io/2023/>

computed by other, often greedy heuristic methods. MaxSLIM uses MaxSAT solvers for both tasks—finding the initial solution and solving local instances.

Local Search with SAT Oracle (LSSO) [5] is another LNS metaheuristic that uses a MaxSAT solver for finding improvements. Here, the user supplies problem specific neighborhood generators and the remaining algorithm is fixed, apart from hyperparameters that control the search. LSSO has been successfully used for cell placement.

LSSO follows the same approach as many other LNS instantiations, particularly using (Max)SAT solvers such as timetabling [7]: the neighborhood is defined based on global properties of the instance. The search is then performed over the whole instance, while restricting how much the solution can change. In contrast, SLIM instantiations look at the structure of the instance and local properties. The search space for the local instances is then unrestricted, apart from ensuring consistency with the global solution. While SLIM performs generally well, some instance’s do not allow for local improvements. Hence, SLIM is often combined with other heuristic methods as they often complement each other. One big advantage of the SLIM approach is that it can be applied to instances that are too large to tackle as a whole, as it is not necessary to look at the whole instance.

In exact MaxSAT solving, a different subdivision based on the graphical representation is used [22]. Here, the soft clauses, but no other variables and clauses, are partitioned based on a graphical representation of the instance. In MaxSLIM we do not partition the instance, i.e., subinstances can overlap. Further, due to the large size of the instances used with MaxSLIM, we also have to restrict variables that do not occur in soft clauses.

2 Preliminaries

2.1 SAT and MaxSAT

A *propositional formula in conjunctive normal form (CNF formula)* is a set of clauses, each *clause* is a set of literals, each *literal* is a propositional variable or a negated propositional variable. We consider a CNF formula as the conjunction of its clauses and each clause as a disjunction of its literals. For a literal $\ell \in \{x, \neg x\}$ we define $\text{var}(\ell) = x$, for a clause C we define $\text{var}(C) = \{\text{var}(\ell) : \ell \in C\}$ and for a CNF formula F we define $\text{var}(F) = \bigcup_{C \in F} \text{var}(C)$. An *assignment* is a mapping $\tau : X \rightarrow \{0, 1\}$ defined on a set X of variables; we write $\text{var}(\tau) = X$. We extend τ to literals by setting $\tau(\neg x) = 1 - \tau(x)$. We implicitly use the equivalency $\neg \neg v = v$. For an assignment τ , we put $\text{lit}(\tau) = \{x : x \in \text{var}(\tau), \tau(x) = 1\} \cup \{\neg x : x \in \text{var}(\tau), \tau(x) = 0\}$. An assignment τ is *total* for a CNF formula F if $\text{var}(\tau) = \text{var}(F)$. All future references to assignments address assignments which may or may not be total, unless explicitly specified. For two assignments τ_1, τ_2 with $\text{var}(\tau_1) \cap \text{var}(\tau_2) = \emptyset$ we define $\tau_1 \cup \tau_2$ to be the assignment with $\text{var}(\tau_1 \cup \tau_2) = \text{var}(\tau_1) \cup \text{var}(\tau_2)$ and $(\tau_1 \cup \tau_2)(x) = \tau_i(x)$ for $x \in \text{var}(\tau_i)$.

An assignment τ *satisfies* a clause C if it sets at least one literal of C to 1. An assignment satisfies a CNF formula if it satisfies all its clauses. For a clause

C and an assignment τ , we write $C[\tau] = \{\ell \in C : \text{var}(\ell) \notin \text{var}(\tau)\}$. For a CNF formula F and an assignment τ , $F[\tau]$ denotes the CNF formula obtained from F by removing all clauses that are satisfied by τ and removing from the remaining clauses all literals that τ sets to 0, that is $F[\tau] = \{C[\tau] : C \in F, \tau \text{ does not satisfy } C\}$. Thus τ satisfies F if and only if $F[\tau] = \emptyset$.

We write $\text{var}(\mathcal{F}) = \text{var}(F_h) \cup \text{var}(F_s)$ and $\mathcal{F}_\tau = (F_h[\tau], F_s[\tau], w')$ where w' is defined for $C \in F_s[\tau]$ by $w'(C) = \sum_{C' \in F_s \text{ with } C = C'[\tau]} w(C')$.

For an assignment τ and a CNF formula F , $\text{UP}_F(\tau)$ denotes the assignment obtained from τ by unit propagation over F . This means, we iteratively extend τ to literals ℓ that are forced because there exists a clause $C \in F$ where $C[\tau] = \{\ell\}$.

An instance of the Maximum Satisfiability problem, or *MaxSAT instance* \mathcal{F} is a triple consisting of two CNF formulas F_h and F_s and a weight function $w : C \in F_s \rightarrow \mathbb{N}$. The clauses of F_h are *hard*, the clauses of F_s *soft*. A *solution* to \mathcal{F} is an assignment τ that satisfies F_h . The *cost* of a solution τ , denoted $\text{cost}(\mathcal{F}, \tau)$, is the sum of weights of the soft clauses not satisfied by τ . An *optimal solution* is one with minimum cost over all solutions, thereby maximizing the sum of weights of the satisfied soft clauses, therefore the name MaxSAT.

We distinguish between two types of MaxSAT solvers, *exact* solvers which provide optimal solutions and *anytime* solvers which aim at providing good solutions within a given time bound and thus are not concerned with the optimality of the solutions. Anytime algorithms can be interrupted at any point of time, upon which they immediately output the best solution found so far and terminate. MaxSAT solvers can be both exact and anytime: a solver's classification expresses if a solver's focus is on proving optimality or finding good solutions, as no solver is good at both aspects. A subtype of exact solvers are exact *incremental solvers*. These solvers are run multiple times for the same MaxSAT instance. In between each run, the MaxSAT instance can be modified. Further, for each run, the user can specify a temporary variable assignment using *assumptions* that the solver uses only for this one run.

3 MaxSAT-SLIM

In this section, we describe *MaxSLIM*, our variant of SLIM for MaxSAT. MaxSLIM expects a MaxSAT instance—the *global instance*—as input. Then, either a global solution τ is provided, or MaxSLIM computes one using a heuristic. MaxSLIM improves this global solution by repeatedly extracting *local instances*, as discussed in the next section, and solving them using a *local solver*. Whenever the local solver finds a solution with lower cost, we found an improvement for the global solution. The local solver is subject to a local timeout and is stopped whenever this timeout elapses.

Alongside our structural approach, we also discuss the details of MaxSAT-LNS's neighborhood definition in subsequent sections.

3.1 Local Instances

Given a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ and weight function w , we iteratively construct a set $L \subseteq \text{var}(\mathcal{F})$ of *candidate variables*.

Given a total assignment τ of F , we define $\tau|_{\text{var}(\mathcal{F}) \setminus L}$ as the restriction of τ to $\text{var}(\mathcal{F}) \setminus L$, and let $\tau_{\bar{L}} = \text{UP}_{F_h}(\tau|_{\text{var}(\mathcal{F}) \setminus L})$. Then L induces the *local instance*

$$\mathcal{F}_L = \mathcal{F}[\tau_{\bar{L}}].$$

\mathcal{F}_L is expected to be much smaller than \mathcal{F} . We call the variables in $\text{var}(\mathcal{F}_L)$ *free variables*. Given a solution π for \mathcal{F}_L , we obtain a new global solution by completing $\tau_{\bar{L}}$ to a total assignment using π :

$$\tau \leftarrow \tau_{\bar{L}} \cup \pi.$$

The updated τ is indeed a solution for \mathcal{F} : all hard clauses not part of \mathcal{F}_L are by definition satisfied by $\tau_{\bar{L}}$ and the hard clauses in \mathcal{F}_L are satisfied by π . A similar argument holds for the soft clauses. The global cost decreases exactly by the value the local cost decreases. Therefore, any improvement for the local instance directly provides improvements for the global solution.

We use the number of free variables for our *budget*. Hence, we want to choose L such that $|\text{var}(\mathcal{F}_L)| \leq b$ for some budget b . This ensures that the local instances do not take too long to solve. Although unit propagation requires additional runtime, this runtime investment is justified by the extra reduction in local instance size and better runtime estimation of using $|\text{var}(\mathcal{F}_L)|$ instead of $|L|$. MaxSAT-LNS uses a similar method: it incrementally fixes the values of some variables, until the number of unassigned variables is below a specific threshold. Our focus on the free variables as opposed to the fixed variables, allows us to follow the instance's structure, as is our next topic.

3.2 Local Instance Selection

The goal of local instance selection (i.e., selection of the candidate variables) is that we would like to reach many unsatisfied soft clauses that can be satisfied by changing the assignment of the free variables. This poses two challenges. The first challenge is identifying the right soft clauses: improvements usually require that some satisfied soft clauses become unsatisfied in exchange for satisfying some previously unsatisfied soft clauses of higher total weight. Once the soft clauses are identified, we know which variables need to become free. The second challenge is identifying the other variables required to free the soft clauses' variables. This gives us the candidate variables for our local instance. This task is hard to perform efficiently, as unit propagation behaviour is very instance specific and hard to predict.

We address these challenges by using the *incidence graph* (also known as the clause-variable incidence graph) for our local instance selection. The incidence graph $G_{\mathcal{F}}$ is the graph with the set of vertices $V(G_{\mathcal{F}}) = \text{var}(\mathcal{F}) \cup F_h \cup F_s$ and the set of edges $E(G_{\mathcal{F}}) = \{\{u, C\} \in E(G_{\mathcal{F}}) : C \in F_h \cup F_s, u \in \text{var}(C)\}$.

Hence, the incidence graph is a bipartite graph that connects the clauses with the variables they contain, negated or unnegated. Given a global solution τ for \mathcal{F} and a set $L \subseteq \text{var}(\mathcal{F})$, we define the restriction of the incidence graph $G_{\mathcal{F},L}$ to unsatisfied clauses. For this definition, we assume that the assignment to the candidate variables changed and use

$$\tau'(x) = \begin{cases} 1 - \tau(x) & \text{if } x \in L \\ \tau(x) & \text{otherwise} \end{cases}$$

Then, $V(G_{\mathcal{F},L}) = \text{var}(\mathcal{F}) \cup \{C \in F_s \cup F_h : C \text{ not satisfied by } \tau'\}$ and $E(G_{\mathcal{F},L}) = E(G_{\mathcal{F}}) \cap (V(G_{\mathcal{F},L}) \times V(G_{\mathcal{F},L}))$.

Local instance selection searches for connected subgraphs of the incidence graph that allow for improvements. We focus on connected subgraphs as after changing an assignment to a variable x , unit propagation can only affect variables within the same connected subgraph as x . The restricted incidence graph focuses this search even further.

MaxSLIM constructs local instances using *strategies* for exploring $G_{\mathcal{F},L}$. Each strategy is a different greedy algorithm that picks variables by maximizing a *metric*. Each metric $s(\cdot)$ defines a *score* for each variable and soft clause x , denoted by $s(x)$.

Algorithm 1 shows the general approach. In each iteration, we initially start from a single unsatisfied soft clause $C \in F_s$. The soft clause is chosen according to the metric and we avoid repeatedly choosing the same soft clause by keeping track of our previous choices in D . Hence, we start with initial set of candidate variables $L_0 = \text{var}(C)$. We then extend this set to $L_{i+1} = L_i \cup S$ —where S depends on the strategy used—until $|\text{var}(\mathcal{F}_{L_{i+1}})|$ exceeds our budget or we have added all variables. Then $L := L_{i+1}$. Whenever we tried all soft clauses or found an improvement, we reset D .

3.3 Strategies

Given a global instance $\mathcal{F} = (F_h, F_s, w)$, a global solution τ for \mathcal{F} , and a metric $s : \text{var}(\mathcal{F}) \cup F_s \rightarrow \mathbb{R}$ (which can depend on τ), we use one of the following strategies for extending L_i to L_{i+1} , where ties are always broken arbitrarily:

- *Variable Strategy*: Let $N_c = \{C : u \in L_i, \{u, C\} \in E(G_{\mathcal{F},L_i})\}$ and $N_v = \{u : C \in N_c, \{u, C\} \in E(G_{\mathcal{F},L_i})\} \setminus L_i$, i.e., N_v is the set of variables which occur in some clause together with at least one variable in L_i . This strategy sets $L_{i+1} = L_i \cup \{\arg \max_{u \in N_v} s(u)\}$. In other words, this strategy adds as many high scoring variables to the local instance as possible.
- *Adjacency Strategy*: This strategy picks a variable $v = \arg \max_{v \in L_i} s(v)$ and then extends L_i to L_{i+1} by adding all variables of distance 2 from v in $G_{\mathcal{F},L_i}$. The idea behind this strategy is that high scoring candidate variables are only useful if they become free. Adding all variables that occur together in a clause with the high scoring variables, increases the chances of the high scoring candidate variables becoming free.

Algorithm 1: MaxSLIM

Data: A MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, a metric $s(\cdot)$, a strategy σ for selecting local instances, and a budget b .

Result: A solution τ .

```

1  $\tau \leftarrow \text{solve}(\mathcal{F})$  // Alternatively,  $\tau$  can be passed as a parameter.
2  $D \leftarrow \emptyset$ 
3 while within global timeout do
4    $i \leftarrow 0$ 
5    $C_s \leftarrow \arg \max_{C \in F_s \setminus D, C \cap \tau = \emptyset} s(C)$ 
6    $L_0 \leftarrow \text{var}(C_s)$ 
7    $D \leftarrow D \cup \{C_s\}$ 
8   while  $\text{var}(L_i) \neq \text{var}(\mathcal{F})$  and  $|\text{var}(\mathcal{F}_{L_i})| < b$  do
9     Extend  $L_i$  to  $L_{i+1}$  using the strategy  $\sigma$ .
10     $i \leftarrow i + 1$ 
11  end
12   $\tau_{L_{i+1}} \leftarrow \text{solve}(\mathcal{F}_{L_{i+1}})$ 
13  if  $\text{cost}(\mathcal{F}_{L_{i+1}}, \tau_{L_{i+1}}) < \text{cost}(\mathcal{F}_{L_{i+1}}, \tau)$  then
14     $\tau \leftarrow \tau|_{\text{var}(\mathcal{F}) \setminus \text{var}(\mathcal{F}_{L_{i+1}})} \cup \tau_{L_{i+1}}$ 
15    Update metric.
16     $D \leftarrow \emptyset$ 
17  end
18  if  $D = \{C \in F_s : C \cap \text{lit}(\tau) = \emptyset\}$  then
19     $D \leftarrow \emptyset$ 
20  end
21 end
22 return  $\tau$ 

```

- *k-Adjacency Strategy*: Similar to the previous strategy, with the key difference that instead of adding all variables of a clause, this strategy only adds the k best variables from each clause. This causes wider exploration of the incidence graph compared to the unbounded version. The drawback is that this strategy is slower.
- *Clause Strategy*: This strategy picks an arbitrary clause $C \in V(G_{\mathcal{F}, L_i})$ with $\text{var}(C) \cap L_i \neq \emptyset$ and then sets $L_{i+1} = L_i \cup \{\arg \max_{v \in \text{var}(C) \setminus L_i} s(v)\}$. The idea here is that should we change the assignment to any candidate variable, some clauses may become unsatisfied. This strategy follows the chain of necessary assignment changes to ensure that all clauses are satisfied.
- *Fast Strategy*: This strategy does not use a metric. Let $v \in L_i$ be an arbitrary vertex and N_v be defined as in the Variable Strategy, then $L_{i+1} = L_i \cup N_v$. Hence, all variables occurring with any variable in L_i are added. This strategy tries to maximize the speed with which local instances are constructed.
- *Global Strategy*: This strategy emulates MaxSAT-LNS’s neighborhood definition and does not follow Algorithm 1: instead of starting from a soft clause, we use $L_0 = \text{var}(\mathcal{F})$ and we stop once we are within our budget. In

each iteration $L_{i+1} = L_i \setminus \{\arg \min_{v \in L_i} s(v)\}$. This strategy ensures that the assignment of high scoring variables is preserved.

Different strategies explore the incidence graph differently and lead to different local instances. Further, different strategies have different runtime complexities. The two factors that influence runtime the most are sorting the variables to pick the variable of highest score and how often we need to run unit propagation to calculate the number of free variables. Hence, the Fast Strategy is indeed the fastest, followed by the Global Strategy, as the latter requires sorting all variables only once. The runtime of the other strategies varies strongly from instance to instance: whenever changes to the assignment cause few propagations, the strategies are fast, as the number of free variables closely matches the number of candidate variables. The Variable Strategy has to maintain a priority queue and is usually the slowest, followed by the Clause Strategy.

Next, we will discuss the different metrics used by the strategies.

3.4 Metrics

The metrics try to identify variables and soft clauses that have a high probability of contributing to an improvement. We heavily use the concept of *units*: variable v is a unit of clause C in respect to an assignment τ if $\{v\} = \text{var}(\text{lit}(\tau) \cap C)$. Hence, if a clause has a unit, changing the unit's assignment will make the clause unsatisfied. We define $\text{unit}(v) = \{C \in F_s \cup F_h : \{v\} = \text{var}(\text{lit}(\tau) \cap C)\}$. Hence, $\text{unit}(v)$ are exactly those clauses that become unsatisfied, if we change the value of v . Particularly for instances with homogenous soft clause weights, many soft clauses end up having the same metric score. For this reason, we use $\arg \min_{v \in C} |\text{unit}(v)|$ —smaller is better—as a tie breaker, whenever the score of two soft clauses is the same.

We consider the following metrics:

- *Random Metric*: Assigns each variable and soft clause a random score. This causes widespread exploration over consecutive local instances, but does not consider which parts of the instance are more promising.
- *Unit Metric*: For each variable $v \in \text{var}(\mathcal{F})$ the score is

$$s(v) = -|\text{unit}(v) \cap F_h| - \sum_{C \in \text{unit}(v) \cap F_s} w(C).$$

Put in words, this metric prefers variables where changing the assigned value would unsatisfy as few clauses as possible. For a clause C the score is $\min_{v \in \text{var}(C)} s(v)$.

- *Counting Metric*: The score of a variable is the negative number of times it was selected as a candidate variable. For a clause C , the score $s(C) = \sum_{v \in \text{var}(C)} s(v)$. This metric encourages exploration of all variables over time.

- *LNS Metric*: The metric used by MaxSAT-LNS [10]. For a variable v , we define

$$s(v) = \sum_{\substack{C \in F_s, \\ v \in \text{var}(C)}} \begin{cases} -w(C), & \text{if } v \in \text{var}(C \cap \text{lit}(\tau)); \\ w(C), & \text{if } C \cap \text{lit}(\tau) = \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

We extend this scoring to a clause C by setting $s(C) = \sum_{v \in \text{var}(C)} s(v)$.

- *Satisfying Metric*:

$$s(v) = \sum_{\substack{C_h \in F_h, C_s \in F_s, \\ v \in \text{var}(F_h), \\ C_h \cap \tau = \{\ell\}, \neg \ell \in C_s}} \begin{cases} 0, & \text{if } \text{lit}(\tau) \cap C_s \neq \emptyset; \\ 0, & \text{if } v = \text{var}(\ell); \\ w(C_s), & \text{otherwise.} \end{cases}$$

This metric identifies variables in unsatisfied soft clauses that cannot be changed to a different value, as they alone satisfy some hard clause. Giving the other variables in these hard clauses a high score and thereby changing their value can enable MaxSLIM to satisfy more soft clauses. Again, for a clause C , $s(C) = \sum_{v \in \text{var}(C)} s(v)$.

- *Ratio Metric*: The score for a variable v is the number of clauses that would be satisfied if v 's assignment were changed divided by the number of clauses that would become unsatisfied by the change. For a clause C the score is $\min_{v \in \text{var}(C)} s(v)$.

Instead of using the metric scores directly as discussed, we use *weighted random sampling*: We use a constant factor c such that $s(x) + c > 0$ for all $x \in F_s \cup \text{var}(\mathcal{F})$ and then, for each $x \in \text{var}(\mathcal{F}) \cup F_s$, we set $s(x) \leftarrow \log u_x \cdot \frac{1}{s(x) + c}$, where u_x is a randomly generated number between 0 and 1. Sampling causes MaxSLIM to explore more different local instances, leading to more improvements over time.

3.5 Local Solvers

MaxSLIM can use any MaxSAT solver as the local solver. We discuss the advantages of different solver types and then discuss our own solver developed for MaxSLIM.

Anytime solvers seem like a straightforward choice, as we only require improved solutions, not necessarily optimal ones. Unfortunately, anytime solvers often struggle to show optimality. Hence, they often run for the full local timeout, where an exact solver can determine within seconds that no improvement is possible. While current implementations of anytime solvers do not offer the option to start from a known solution, the underlying algorithms themselves would support it in principle.

Exact solvers are indeed often faster in showing that no improvement is possible. The main disadvantage when using exact solvers is that they often do not find (good) intermediate solutions, whenever the local run does not finish. Further, exact solvers cannot profit from an already known solution.

Incremental exact solvers allow for encoding the instance once and then perform local runs by specifying the local instance via assumptions. This has two advantages: time is saved on solver instantiations and insight is gained by the solver via cores; learned clauses can be used over several local runs. Unfortunately, local instances often require thousands to hundreds of thousands of assumptions, severely degrading the incremental solver’s performance. This large number of assumptions also means that learned information usually does not carry over from local instance to local instance. Indeed, in our experiments we found that only negligibly many cores from a local instance generalize to the global instance. Hence, there is rarely any advantage of using an incremental solver over a simple exact solver.

MaxSLIM offers several solvers, but the default solver that is also used in our experiments, is our own simple implementation of the OLL algorithm [1, 17] together with some specific adaptations. For brevity, we discuss the main differences to a plain OLL implementation, without giving the details of OLL itself. For this description, we assume that each soft clause consists of only one literal, as this can always be achieved by introducing auxiliary variables.

A simple method that works for all solvers is using the fact that any improved solution has to satisfy at least one additional soft clause. Let $F_{L,s}$ be the set of soft clauses of local instance F_L and $F_{L,u} = \{C \in F_{L,s} : C \cap \text{lit}(\tau) = \emptyset\}$. We can now add a single disjunction stating that at least one soft clause in $F_{L,u}$ has to be satisfied. This has at least one of two effects: any solution the local solver finds is different from the current solution, which can lead to improvements in subsequent local runs, and more often, adding this disjunction causes an increase of the cost of an optimal solution. This in turn speeds up the time our solver needs to determine that no improvement is possible, as we can stop once we reach the best known cost.

The second change is *upper bound search*, where our solver actively tries to find improved non-optimal solutions. In each OLL iteration, we assume that exactly one additional soft clause in $F_{L,u}$ is satisfied. If the subsequent SAT call returns satisfiable, we have reduced our upper bound and found a better solution. Otherwise, we proceed as usual: we extract a core and increase our lower bound. This way, we may find improved solutions, even if the solver does not find an optimal solution.

3.6 Parallel MaxSAT-SLIM

Local instances can be solved in parallel, as long as the local solutions are compatible to each other and the global solution stays a valid solution. Given a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, a set \mathcal{L} of disjoint sets $L_1, \dots, L_n \subset \text{var}(\mathcal{F})$, and a global solution τ , we construct a sequence $(\mathcal{F}_1, \dots, \mathcal{F}_n)$ of local instances that can be solved in parallel, where $\mathcal{F}_i = (F_{h,i}, F_{s,i})$ for $1 \leq i \leq n$.

We use the shorthands $X_i = \bigcup_{L_j \in \mathcal{L}, j < i} L_j$ and $\bar{\tau}_i = (\text{UP}_{F_h}(\tau|_{\text{var}(\mathcal{F}) \setminus L_i}))|_{X_i}$. First, we construct the set of hard clauses. Whenever for a clause $C \in F_h$ (i) $C \cap \bar{\tau}_i = \emptyset$ and (ii) $C \cap \tau|_{L_i} \neq \emptyset$, we add $\{\ell \in C : \text{var}(\ell) \in L_i\}$ to $F_{h,i}$. The first condition states that the clause is not satisfied by any non-free variable that is

not part if local instance prior in the sequence. The second condition reduces the size of the local instance even further. If $\tau|_{L_i}$ and $\bar{\tau}_i$ does not satisfy a clause, then it is satisfied by $\tau|_{X_i}$ and, consequently, the local solutions to the previous local instances ensure that C remains satisfied.

The soft clauses are constructed slightly differently, as they might or might not be satisfied by τ . Whenever for a clause $C \in F_s$ (i) $C \cap \bar{\tau}_i = \emptyset$, and either (iia) $C \cap \tau|_{L_i} \neq \emptyset$ or (iib) $\text{var}(C) \cap X_i = \emptyset$, we add $\{\ell \in C : \text{var}(\ell) \in L_i\}$ to $F_{s,i}$. Conditions (i) and (iia) are the same as for the hard clauses. Condition (iib) addresses soft clauses that are not satisfied by τ : in this case, we have to avoid that the soft clause can be satisfied in two local solution, which would (mis)count the cost reduction twice and might cause an overall increase in the cost.

We say that the local instances $\mathcal{F}_1, \dots, \mathcal{F}_n$ are *independent* of each other and can now formulate our key observation:

Observation 1 *Given a MaxSAT instance \mathcal{F} , a sequence of independent local instances $(\mathcal{F}_1, \dots, \mathcal{F}_n)$, a global solution τ for \mathcal{F} , and a sequence local solutions $\mathcal{T} = (\tau'_1, \dots, \tau'_n)$ for the local instances such that for each $1 \leq i \leq n$ it holds that $\text{cost}(\tau'_i, \mathcal{F}_i) \leq \text{cost}(\tau, \mathcal{F}_i)$. Let $\tau' = \bigcup_{\tau'_i \in \mathcal{T}} \tau'_i$, then $\tau|_{\text{var}(\mathcal{F}) \setminus \text{var}(\tau')} \cup \tau'$ is a solution for \mathcal{F} with cost at most $\text{cost}(\tau, \mathcal{F})$.*

This idea can be implemented in our algorithm in a straightforward manner. we construct our first local instance as usual and for the subsequent $n - 1$ local instances we follow the construction as defined above. Other changes to MaxSLIM are not necessary.

4 Implementation Details

We implemented MaxSLIM and included the anytime solvers *NuWLS*² [4] and *Loandra* [3], which in turn provide the anytime solvers from *TT-Open-WBO-Inc* [21], *Open-WBO-Inc* [11, 19], and *Open-WBO* [16]. This allows us to employ the top anytime solvers within MaxSLIM. These anytime solvers can be used to compute an initial solution, with MaxSLIM using NuWLS as the default, as NuWLS won the 2022 MaxSAT evaluation. The initial solver is run for a specified time, which defaults to one minute. If no solution can be found in that time, the initial solver is run until any solution is found. Alternatively, the initial solution can be provided as an input.

As the local solver, we use our OLL implementation as default. Alternatively, one of the anytime solvers can be used. We also provide an interface that writes the local MaxSAT instance to a file and calls a user-specified MaxSAT solver. Each local solver run is limited by a local timeout which is 55 seconds as a default. The default auf 55 seconds allows local instance selection and solving to complete within one minute.

MaxSLIM uses a specific fraction of $|\text{var}(\mathcal{F})|$ as the initial budget, governed by an upper and a lower bound that insure an initial local instance that is neither

² <https://maxsat-evaluations.github.io/2022/descriptions.html>

too small nor too big. Every five consecutive failures of finding an improvement, we increase the budget by $\frac{|\text{var}(\mathcal{F})|}{10}$. Whenever the budget reaches the total number of variables, we run the local solver without a timeout on the whole instance. The defaults MaxSLIM uses are 10% of the instance’s variables as starting budget, but no more than 25 000 variables and no lower bound.

MaxSLIM also offers the option to use multiple soft clauses to initialize the local instance. The default here is to use a single soft clause.

5 Experimental Evaluation

In our experimental evaluation, we address several research questions:

- Q1:** Is there a benefit of a structured approach compared to an unstructured/random one like MaxSAT-LNS? (Section 5.2)
- Q2:** Is there a benefit of local improvement (structured or random) as compared to just running the initial solver for the entire time? (Section 5.3)
- Q3:** Does parallelization yield better results in practice? (Section 5.4)
- Q4:** What strategies/metrics work best? (Section 5.5)

We first introduce our implementation and experimental setup and then examine the results concerning these questions.

5.1 Experimental Setup

Cluster The experiments were run on servers with two AMD EPYC 7402 CPUs, each with 2 CPUs, each having 24 cores running at 2.8 GHz, and using Ubuntu 18.04. Each run had 64 GB of memory. We used GCC 11 to compile MaxSLIM, NuWLS, and MaxSAT-LNS.

Instances We used the instances from the 2023 MaxSAT evaluation’s anytime track³. We keep the provided separation into weighted and unweighted instances, where unweighted means all soft clauses have weight 1. The set contains 179 unweighted and 160 weighted instances.

Comparison We compare MaxSLIM against MaxSAT-LNS⁴ and NuWLS⁵.

We utilize the same scoring system as the MaxSAT evaluation: let c_{best} be the cost of the best known solution for the given instance, and c_{solver} be the cost of the solution the given solver found, the score is calculated with $\frac{c_{\text{best}}+1}{c_{\text{solver}}+1}$. The solvers finding the best solution get a score of 1. The lowest cost among all our experiments provides the baseline. Hence, values in different tables are comparable.

³ <https://maxsat-evaluations.github.io/2023/benchmarks.html>

⁴ https://github.com/rgh000/MaxSAT_LNS

⁵ <https://maxsat-evaluations.github.io/2023/descriptions.html>

We perform three runs per configuration, using three specific random seeds for reproducibility. The random seeds themselves have been initially randomly generated. We always give the average of the three runs. The randomness of all methods causes a comparatively large variance. Over all instances the variance is, depending on the method, between 0 and 0.003, but the maximum variance observed on a single instance is 0.394. This is caused by hard to find and impactful improvements.

Initially, we ran NuWLS for one hour and extracted the current best solution after one minute, 5 minutes, 30 minutes, and 60 minutes. For our evaluation we use a total runtime of 5, 30 and 60 minutes. For NuWLS the results are exactly the solutions extracted at the 5, 30, and 60 minute marks. For MaxSLIM and MaxSAT-LNS the timeline is different, as they are seeded with NuWLS’s solution. For the 5 minute runs, MaxSLIM and MaxSAT-LNS already start at minute 1 and get NuWLS’s 1-minute solution as input. Hence, they run for 4 minutes. For the 30 and 60 minute runs, MaxSLIM and MaxSAT-LNS start at minute 5 and get NuWLS’s 5-minute solution as input. Hence, they run for 25 and 55 minutes respectively. The 5 minute timelimit follows the rules of the MaxSAT evaluation. We use the same starting solution for the 30 and 60 minute timelimits to better compare the method’s contribution to the improvement.

Configuration If not stated differently, MaxSLIM was run using the default parameters as stated above. We use the Variable Strategy as the default and either the Unit Metric if the instance is unweighted or the Satisfying Metric if the instance is weighted. For comparison, we also use MaxHS⁶ [6] as a local solver.

5.2 Comparison of SLIM and LNS (Q1)

The comparison between MaxSAT-LNS and MaxSLIM in Table 1 shows that MaxSLIM performs overall better than MaxSAT-LNS. Particularly, with higher timeouts, MaxSLIM finds more improvements, while MaxSAT-LNS’s finds fewer and fewer improvements. This suggests that the structured approach searches more exhaustively for improvements.

There is no easily identifiable property of the instance that indicates which of the two methods will be better for a given instance. Neither number of variables, number of soft/hard clauses, number of edges in the incidence graph or the cost are good indicators.

5.3 Comparison of MaxSLIM and NuWLS (Q2)

One crucial question is whether MaxSLIM is better than running the anytime solver for the entire duration. Table 1 shows that for the short 5-minute runs, NuWLS performs on average better. Although it finds improvements for fewer

⁶ <https://github.com/fbacchus/MaxHS>, Commit c8df41a

Table 1: Comparison between MaxSLIM (MSLIM), MaxSAT-LNS (MLNS) and NuWLS. *Improved* shows on how many instances the solver could improve upon the initial solution. *Better* shows on how many instances one solver achieved a better result than the other solver.

	Unweighted		Weighted		Unweighted		Weighted	
	MLNS	MSLIM	MLNS	MSLIM	NuWLS	MSLIM	NuWLS	MSLIM
5-Minutes	164 Instances		154 Instances		164 Instances		154 Instances	
Score	0.885	0.898	0.833	0.837	0.901	0.898	0.885	0.837
Improved	67	75	82	83	62	75	63	83
Better	34	48	41	40	31	46	43	52
30-Minutes	171 Instances		159 Instances		171 Instances		159 Instances	
Score	0.926	0.939	0.914	0.928	0.936	0.939	0.918	0.928
Improved	66	72	82	85	55	72	50	85
Better	28	48	31	48	24	50	29	64
60-Minutes	171 Instances		159 Instances		171 Instances		159 Instances	
Score	0.931	0.943	0.919	0.938	0.937	0.943	0.925	0.938
Improved	68	74	82	87	55	74	52	87
Better	32	48	30	51	22	50	29	64

instances, it performs very well on some instances, lowering the average. This changes for the longer runtimes, where MaxSLIM outperforms NuWLS.

The large gap between MaxSLIM and NuWLS after the 5-minute timeout shows how good NuWLS is at finding improvements quickly. MaxSLIM is considerably slower as establishing the correct budget takes up time in the beginning of the run. The strength of MaxSLIM is highlighted in the longer runtimes, where its structured approach balances its slower performance as it consistently finds improvements.

While there is no clear indicator for which instances one or the other solver performs better, the results suggest that NuWLS performs better for some of the application domains. As this is not correlated to instance size, we expect that the instances where NuWLS performs better require improvements that are not local.

5.4 Parallel MaxSLIM (Q3)

In our evaluation of parallelism, we ran MaxSLIM with an increasing number of parallel threads. The results are in Table 2.

The effectiveness of parallelization is very instance specific and overall it performs worse. Particularly for instances where large local instances are required, it is rarely possible to find more than one local instance that allows for improvements. Further, if the instance contains few unsatisfied soft clauses, they cannot

Table 2: Performance of parallelized MaxSLIM over a 5-minute timelimit using a different number of threads. The single threaded run is used as a baseline. The table also shows on how many instances parallelization performed *Better* or *Worse* than the baseline.

Threads	Unweighted				Weighted			
	Score	Improved	Better	Worse	Score	Improved	Better	Worse
1	0.898	75	-	-	0.838	83	-	-
2	0.878	67	15	56	0.836	85	28	36
4	0.877	66	14	53	0.834	89	25	41
6	0.881	54	20	21	0.833	85	29	40

be split into many subsets that allow for improvements. The extra synchronization effort then significantly slows down MaxSLIM.

5.5 Impact of Metrics and Strategies (Q4)

MaxSLIM offers a large number of hyperparameters. An interesting question is, how do these hyperparameters impact the performance of MaxSLIM. Given the large number of possible configurations, we limit our presentation to the results of using different metrics and strategies. Hence, we use the same configuration as for the other experiments and only vary either the metric or strategy.

Table 3 shows the impact of different metrics. Interestingly, the worst metric for the unweighted instances is the best one for weighted instance, and the opposite is almost true as well. Overall, the metrics have a large impact on the results, as each of them manages to achieve outstanding results on some instances that cannot be achieved using a different metric.

Table 3 shows how the different strategies performed. Here, there is a clear strategy that performed best for weighted and unweighted instances. As with the metrics, each strategy has its unique applications, where it performed better than the other strategies.

Given an oracle that tells us the best strategy and metric for a given instance (Virtual Best), we could achieve 0.929 score on unweighted instances and 0.860 on weighted instances.

We also looked into different local solver choices. For unweighted instances, our solver performed indeed best (0.898), followed by our solver without upper bound search (0.889). Calling an external solver—MaxHS—has a significant penalty (0.875) and the anytime solver performs even worse (0.872). Hence, for unweighted instances, the local solver choice is very clear. For weighted instances, our solver without upper bounded search performs actually better than with upper bound search (0.844 compared to 0.837). This is due to the use of stratification for weighted instances, which can also find non-optimal solutions. For weighted instances, the anytime solver performed comparatively well (0.836) and the external solver performed worst (0.824).

Table 3: Performance of different metrics and strategies. Best Score shows on how many instances MaxSLIM found the best solution using the given metric or strategy. Unique Best shows on how many instances the best solution could only be found using the given metric or strategy. The timelimit was 5 minutes.

Metric	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Unit	0.898	75	25	12	0.837	86	36	11
Ratio	0.897	78	26	10	0.840	83	28	9
Counting	0.896	68	22	11	0.838	86	36	11
Random	0.894	65	17	6	0.836	82	18	4
Satisfying	0.893	70	27	13	0.838	83	38	10
LNS	0.889	78	26	10	0.841	87	45	19
Strategy	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Variable	0.898	75	30	13	0.838	83	34	21
Adjacency	0.892	76	32	19	0.832	88	18	10
Clause	0.892	70	24	10	0.837	92	30	16
5-Adjacency	0.890	75	30	15	0.833	90	21	10
Fast	0.886	54	18	7	0.835	78	28	15
Global	0.882	66	21	8	0.828	87	18	7

6 Conclusion

In this paper, we have proposed MaxSLIM as a structured approach to anytime MaxSAT solving. It tackles the problem of anytime MaxSAT solving by iteratively extracting and solving smaller subinstances whose selection is guided by the graphical structure of the instance. This combines anytime and exact MaxSAT solvers in a novel way. Our experimental evaluation shows the competitiveness of MaxSLIM as compared to state-of-the-art anytime solvers, and other LNS approaches. MaxSLIM’s trajectory of improvements over time is particularly attractive for applications with more than a few minutes of runtime.

Our evaluation uses a default configuration and our results show that choosing the parameters—strategy, metric, local solver, timeouts, initial number of soft clauses—according to the application can significantly improve MaxSLIM’s performance even further. An interesting avenue for further research is adapting other anytime solver to integrate better within MaxSLIM. This would allow us to interleave local improvement phases with additional runs of the global solver, starting from the best solution found so far. Such an interleaved SLIM approach has shown to be surprisingly powerful for circuit minimization [28]. As MaxSLIM can benefit from a good tuning of its parameters, we expect that further efficiency improvements can be obtained through automated algorithm configuration, possibly even adjusting parameters during the run [2]. Finally, further investigations on parallel local improvement methods could be profitable.

References

1. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Dovier, A., Costa, V.S. (eds.) *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012*, September 4-8, 2012, Budapest, Hungary. *LIPIcs*, vol. 17, pp. 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012). <https://doi.org/10.4230/LIPIcs.ICLP.2012.211>
2. Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Reactive dialectic search portfolios for MaxSAT. In: Singh, S., Markovitch, S. (eds.) *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, February 4-9, 2017, San Francisco, California, USA. pp. 765–772. AAAI Press (2017), <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14872>
3. Berg, J., Demirovic, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L., Stergiou, K. (eds.) *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*. *Lecture Notes in Computer Science*, vol. 11494, pp. 39–56. Springer (2019). https://doi.org/10.1007/978-3-030-19212-9_3
4. Chu, Y., Cai, S., Luo, C.: NuWLS: improving local search for (weighted) partial MaxSAT by new weighting techniques. In: Williams, B., Chen, Y., Neville, J. (eds.) *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023*, Washington, DC, USA, February 7-14, 2023. pp. 3915–3923. AAAI Press (2023), <https://ojs.aaai.org/index.php/AAAI/article/view/25505>
5. Cohen, A., Nadel, A., Ryvchin, V.: Local search with a SAT oracle for combinatorial optimization. In: Groote, J.F., Larsen, K.G. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*. *Lecture Notes in Computer Science*, vol. 12652, pp. 87–104. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_5
6. Davies, J., Bacchus, F.: Exploiting the power of mip solvers in maxsat. In: Järvisalo, M., Gelder, A.V. (eds.) *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*. *Lecture Notes in Computer Science*, vol. 7962, pp. 166–181. Springer (2013). https://doi.org/10.1007/978-3-642-39071-5_13
7. Demirovic, E., Musliu, N.: MaxSAT-based large neighborhood search for high school timetabling. *Comput. Oper. Res.* **78**, 172–180 (2017). <https://doi.org/10.1016/j.cor.2016.08.004>
8. 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, February 26 - March 1, 2017*. The Internet Society (2017), <https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-synthesis-semantic-malware-signatures-using-maximum-satisfiability/>
9. Fichte, J.K., Lodha, N., Szeider, S.: SAT-based local improvement for finding tree decompositions of small width. In: Gaspers, S., Walsh, T. (eds.) *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference,*

- Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 401–411. Springer Verlag (2017). https://doi.org/10.1007/978-3-319-66263-3_25
10. Hickey, R., Bacchus, F.: Large neighbourhood search for anytime MaxSAT solving. In: Raedt, L.D. (ed.) Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22. pp. 1818–1824. International Joint Conferences on Artificial Intelligence Organization (7 2022). <https://doi.org/10.24963/ijcai.2022/253>
 11. Joshi, S., Kumar, P., Rao, S., Martins, R.: Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.* **11**(1), 73–97 (2019). <https://doi.org/10.3233/SAT190118>, <https://doi.org/10.3233/SAT190118>
 12. Khoshnood, S., Kusano, M., Wang, C.: ConcBugAssist: constraint solving for diagnosis and repair of concurrency bugs. In: Young, M., Xie, T. (eds.) Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12–17, 2015. pp. 165–176. ACM (2015). <https://doi.org/10.1145/2771783.2771798>
 13. Kulikov, A.S., Pechenev, D., Slezkin, N.: SAT-based circuit local improvement. In: Szeider, S., Ganian, R., Silva, A. (eds.) 47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22–26, 2022, Vienna, Austria. LIPIcs, vol. 241, pp. 67:1–67:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). <https://doi.org/10.4230/LIPIcs.MFCS.2022.67>, <https://doi.org/10.4230/LIPIcs.MFCS.2022.67>
 14. Lodha, N., Ordyniak, S., Szeider, S.: A SAT approach to branchwidth. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 179–195. Springer Verlag (2016). https://doi.org/10.1007/978-3-319-40970-2_12
 15. Lodha, N., Ordyniak, S., Szeider, S.: SAT-encodings for special treewidth and pathwidth. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 429–445. Springer Verlag (2017). https://doi.org/10.1007/978-3-319-66263-3_27, <http://www.ac.tuwien.ac.at/files/tr/ac-tr-17-012.pdf>
 16. Martins, R., Manquinho, V.M., Lynce, I.: Open-WBO: A modular MaxSAT solver. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14–17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8561, pp. 438–445. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_33
 17. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O’Sullivan, B. (ed.) Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8656, pp. 564–573. Springer (2014). https://doi.org/10.1007/978-3-319-10428-7_41
 18. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: A survey and assessment. *Constraints An Int. J.* **18**(4), 478–534 (2013). <https://doi.org/10.1007/s10601-013-9146-2>
 19. Nadel, A.: Solving maxsat with bit-vector optimization. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Feder-

- ated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10929, pp. 54–72. Springer (2018). https://doi.org/10.1007/978-3-319-94144-8_4
20. Nadel, A.: Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In: Barrett, C.W., Yang, J. (eds.) 2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019. pp. 193–202. IEEE (2019). <https://doi.org/10.23919/FMCAD.2019.8894273>
 21. Nadel, A.: Polarity and variable selection heuristics for sat-based anytime maxsat. *J. Satisf. Boolean Model. Comput.* **12**(1), 17–22 (2020). <https://doi.org/10.3233/sat-200126>
 22. Neves, M., Martins, R., Janota, M., Lynce, I., Manquinho, V.M.: Exploiting resolution-based representations for MaxSAT solving. In: Heule, M., Weaver, S.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9340, pp. 272–286. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_20
 23. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of Metaheuristics, pp. 399–419. Springer Verlag (2010)
 24. Ramaswamy, V.P., Szeider, S.: MaxSAT-based postprocessing for treedepth. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12333, pp. 478–495. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_28
 25. Ramaswamy, V.P., Szeider, S.: Learning fast-inference bayesian networks. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 17852–17863 (2021), <https://proceedings.neurips.cc/paper/2021/hash/94e70705efae423efda1088614128d0b-Abstract.html>
 26. Ramaswamy, V.P., Szeider, S.: Turbocharging treewidth-bounded bayesian network structure learning. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 3895–3903. AAAI Press (2021). <https://doi.org/10.1609/aaai.v35i5.16508>
 27. Ramaswamy, V.P., Szeider, S.: Learning large Bayesian networks with expert constraints. In: Cussens, J., Zhang, K. (eds.) Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence. Proceedings of Machine Learning Research, vol. 180, pp. 1592–1601. PMLR (01–05 Aug 2022), <https://proceedings.mlr.press/v180/peruvemba-ramaswamy22a.html>
 28. Reichl, F., Slivovsky, F., Szeider, S.: Circuit minimization with QBF-based exact synthesis. In: Williams, B., Chen, Y., Neville, J. (eds.) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023. pp. 4087–4094. AAAI Press (2023), <https://ojs.aaai.org/index.php/AAAI/article/view/25524>

29. Schidler, A.: SAT-based local search for plane subgraph partitions (CG challenge). In: Goaoc, X., Kerber, M. (eds.) 38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany. LIPIcs, vol. 224, pp. 74:1–74:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). <https://doi.org/10.4230/LIPIcs.SoCG.2022.74>
30. Schidler, A., Szeider, S.: SAT-based decision tree learning for large data sets. In: Proceedings of AAAI’21, the Thirty-Fifth AAAI Conference on Artificial Intelligence. AAAI Press (2021). <https://doi.org/10.1609/aaai.v35i5.16509>
31. Schidler, A., Szeider, S.: SAT-boosted tabu search for coloring massive graphs. *ACM J. Exp. Algorithmics* **28** (jul 2023). <https://doi.org/10.1145/3603112>
32. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J. (eds.) Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer (1998). https://doi.org/10.1007/3-540-49481-2_30
33. Si, X., Zhang, X., Grigore, R., Naik, M.: Maximum satisfiability in software analysis: Applications and techniques. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 68–94. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_4
34. Zhu, C.S., Weissenbacher, G., Sethi, D., Malik, S.: Sat-based techniques for determining backbones for post-silicon fault localisation. In: Zilic, Z., Shukla, S.K. (eds.) 2011 IEEE International High Level Design Validation and Test Workshop, HLDVT 2011, Napa Valley, CA, USA, November 9-11, 2011. pp. 84–91. IEEE Computer Society (2011). <https://doi.org/10.1109/HLDVT.2011.6113981>