



# Structure-Guided Local Improvement for Maximum Satisfiability

André Schidler  

Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider  

Algorithms and Complexity Group, TU Wien, Austria

---

## 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 with an exact solver smaller subinstances that are chosen based on the graphical structure of the instance. We investigate various strategies to pinpoint these subinstances. This structure-guided selection of subinstances provides an exact 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.

**2012 ACM Subject Classification** Hardware → Theorem proving and SAT solving; Theory of computation → Discrete optimization; Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming; General and reference → Experimentation

**Keywords and phrases** maximum satisfiability, large neighborhood search (LNS), SAT-based local improvement (SLIM), incomplete MaxSAT, graphical structure, metaheuristic

**Digital Object Identifier** 10.4230/LIPIcs.CP.2024.26

## Supplementary Material

*Software (Source Code/Results):* <https://doi.org/10.5281/zenodo.12516816>

**Funding** Austrian Science Fund (FWF), project 10.55776/P36420.

**Acknowledgements** Part of this work was carried out while taking part in the Dagstuhl Seminar 23261 “SAT Encodings and Beyond,” as well as in the extended reunion of the program “Satisfiability: Theory, Practice, and Beyond” in the spring of 2023 at the Simons Institute for the Theory of Computing at UC Berkeley.

## 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 [37], post-silicon fault localization [38], the identification of concurrency bugs and suggestions for fixes [13], and malware detection in smartphone apps [10]. Additional applications and case studies [20, 22] highlight MaxSAT’s versatility in computer-aided design and related areas. While the focus in the past was 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. Hickey and Bacchus [12] introduced a technique using Large Neighborhood Search [36] that integrates the capabilities of exact and anytime solvers, harnessing the advantages of both.



© André Schidler and Stefan Szeider;

licensed under Creative Commons License CC-BY 4.0

30th International Conference on Principles and Practice of Constraint Programming (CP 2024).

Editor: Paul Shaw; Article No. 26; pp. 26:1–26:23

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 subinstances 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 several different metrics in our framework. MaxSLIM uses a new MaxSAT solver specifically designed for solving local instances, or, alternatively, can use any other MaxSAT solver for this purpose.

Our comprehensive experimental evaluation on the instances from the 2023 MaxSAT Evaluation<sup>1</sup> 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.

### Related Work

As mentioned above, Hickey and Bacchus [12] proposed an anytime MaxSAT solver based on the Large Neighborhood Search (LNS) metaheuristic [36, 25] called MaxSAT-LNS. Similar to MaxSLIM, MaxSAT-LNS also tries to improve a sub-optimal solution using LNS. In each round of LNS, a weighted random subset of variables is selected and their values fixed. Then, a separate solver is run to find an improved assignment for the remaining variables. 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 instance.

SAT-Based Local Improvement (SLIM) is a specific type of LNS tailored for the use with (Max)SAT solvers. As such, SLIM is an anytime meta-heuristic that embeds (Max)SAT encodings into heuristic algorithms. In the past, SLIM has been used for a variety of problems, such as graph decomposition problems [11, 15, 16, 26], Bayesian Network structure learning [27, 28, 29], decision tree induction [33, 35], graph coloring [32, 34], and circuit minimization [14, 30, 31]. 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 computed by other, often greedy heuristic methods.

Other SAT-based LNS approaches exist for timetabling [9] and cell placement [8]. Cell placement follows the metaheuristic *Local Search with SAT Oracle (LSSO)* where the user supplies problem specific neighborhood generators and the remaining algorithm is fixed, apart from hyperparameters that control the search. These LNS instantiations, like most other, define the neighborhood 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

---

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

with the global solution. One big advantage of the SLIM approach is that it can be applied to instances that are too large to encode as a whole, as it is only necessary to encode the local instances.

Given the large volume of research, we focus the discussion of anytime MaxSAT solving on the state-of-the-art solvers used in our experiments. SATLike [6] is a dynamic local search algorithm that is used in several anytime solvers. SATLike uses a dynamic weighting scheme that is used to pick variables that are then flipped in the hope of finding a good feasible solution. NuWLS [7] is based on the SATLike algorithm and proposes a different weighing scheme. Its implementation NuWLS-c uses another anytime MaxSAT solver (TT-Open-WBO-Inc) for the initial solution. NuWLS-c won the MaxSAT Evaluations 2022 and 2023. NoSAT MaxSAT [5] is another anytime solver based on SATLike and most recently NuWLS, but in contrast to other anytime solvers, relies solely on local search without invoking a SAT solver. TT-Open-WBO-Inc [24] uses either Open-WBO-Inc's [18] Boolean multilevel optimization (BMO) [17] for weighted instances, or a SAT-based bit-vector optimization algorithm (BVO) [21] for unweighted instances. Instead of calling a SAT solver, TT-Open-WBO-Inc's BVO and BMO algorithms call a local search solver that searches for solutions close to the best known solution. The local solver achieves this by polarity saving and incremental SAT calls [23]. Loandra [4] uses preprocessing and linear search (SAT-UNSAT). The linear search is sped up by preprocessing the instance using core-guided search: the core-guided search runs for a limited time, after which the reformulated instance is passed to the linear search. These solvers follow, broadly speaking, two approaches: local search (NuWLS, NoSAT, TT-Open-WBO-Inc) and simplifying the whole instance (Open-WBO-Inc, Loandra) in order to make finding a good feasible solution easier. In contrast, MaxSLIM looks at much larger neighborhoods than local search and does not modify the instance itself. Instead it tries to repeatedly extract easier subproblems.

## 2 Preliminaries

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  $\tau$  to literals by setting  $\tau(\neg x) = 1 - \tau(x)$ . We implicitly use the equivalency  $\neg\neg v = v$ . For an assignment  $\tau$ , 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  $\tau$  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  $\tau_1, \tau_2$  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  $\tau$  *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  $\tau$ , we write  $C[\tau] = \{\ell \in C : \text{var}(\ell) \notin \text{var}(\tau)\}$ . For a CNF formula  $F$  and an assignment  $\tau$ ,  $F[\tau]$  denotes the CNF formula obtained from  $F$  by removing all clauses that are satisfied by  $\tau$  and removing from the remaining clauses all literals that  $\tau$  sets to 0, that is  $F[\tau] = \{C[\tau] : C \in F, \tau \text{ does not satisfy } C\}$ . Thus  $\tau$  satisfies  $F$  if and only if  $F[\tau] = \emptyset$ .

For an assignment  $\tau$  and a CNF formula  $F$ ,  $\text{UP}_F(\tau)$  denotes the assignment obtained from  $\tau$  by unit propagation over  $F$ . This means, we iteratively extend  $\tau$  to literals  $\ell$  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  $\tau$  that satisfies  $F_h$ . The *cost* of a solution  $\tau$ , denoted  $\text{cost}(\mathcal{F}, \tau)$ , is the sum of weights of the soft clauses not satisfied by  $\tau$ . 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. In an *unweighted* instance, every soft clause has weight 1.

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  $w'(C) = \sum_{C' \in F_s \text{ with } C = C'[\tau]} w(C')$ .

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 none of the existing solvers is good at both aspects. A subtype of exact solvers are exact *incremental solvers*. These solvers are run multiple times on almost 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*.

### 3 MaxSLIM

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  $\tau$  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.

#### 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* that induce our local instance. The candidate variables are selected using a strategy, the topic of Section 3.3. Given a total assignment  $\tau$  of  $F$ , we define  $\tau|_{\text{var}(F) \setminus L}$  as the restriction of  $\tau$  to  $\text{var}(F) \setminus L$ , and let  $\tau_{\bar{L}} = \text{UP}_{F_h}(\tau|_{\text{var}(F) \setminus L})$ . Hence,  $\tau|_{\text{var}(F) \setminus L}$  is the assignment after fixing the value of all non-candidate variables and performing unit propagation and  $L$  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*, which are a subset of the candidate variables. Given a solution  $\pi$  for  $\mathcal{F}_L$ , we obtain a new global solution by completing  $\tau_{\bar{L}}$  to a total assignment using  $\pi$ :  $\tau \leftarrow \tau_{\bar{L}} \cup \pi$ .

The updated  $\tau$  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  $\pi$ . 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 also improves the global solution.

We specify the *budget* in terms of the number of free variables. 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. 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 considering the *incidence graph* (also known as the clause-variable incidence graph) for our local instance selection. While other graphical models for MaxSAT instances exist (such as the primal graph or the resolution graph), our research shows that the incidence graph is best suited for our method: (i) the incidence graphs contains all the information available in the primal graph and resolution graph, and (ii) the neighborhood of a variable shows which clauses are impacted in case we change the variable's value. 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  $E(G_{\mathcal{F}}) = \{ \{u, C\} : C \in F_h \cup F_s, u \in \text{var}(C) \}$ . Hence, the incidence graph is a bipartite graph that connects the clauses with the variables they contain, negated or unnegated. We annotate the edges with the polarity of the variable in the clause. Given a global solution  $\tau$  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

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

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 by considering only those clauses that become unsatisfied after changing  $x$ .

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 or 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  $L_0 = \text{var}(C)$  of candidate variables. 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

■ **Algorithm 1** MaxSLIM.

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

**Output:** A solution  $\tau$ .

```

1:  $\tau \leftarrow \text{solve}(\mathcal{F})$  //  $\tau$  can also be passed as a parameter.
2:  $D \leftarrow \emptyset$  //  $D$  keeps track of visited soft clauses.
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 while
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 if
18:   if  $D = \{C \in F_s : C \cap \text{lit}(\tau) = \emptyset\}$  then
19:      $D \leftarrow \emptyset$ 
20:   end if
21: end while
22: return  $\tau$ 

```

---

budget or we have added all variables. Then  $L := L_{i+1}$ . Whenever we tried all soft clauses or found an improvement, we reset  $D$ . Hence, MaxSLIM runs either until the global timeout is reached or the budget allows solving the whole instance.

### 3.3 Strategies

Given a global instance  $\mathcal{F} = (F_h, F_s, w)$ , a global solution  $\tau$  for  $\mathcal{F}$ , and a metric  $s : \text{var}(\mathcal{F}) \cup F_s \rightarrow \mathbb{R}$  (which can depend on  $\tau$ ), 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  $L_{i+1} = L_i \cup \{\arg \max_{u \in N_v} s(u)\}$ . I.e., this strategy adds as many high-scoring variables to the local instance as possible.
- *k-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 the  $k$  best 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 variables that occur together in a clause with the high-scoring variables increases the chances of the high scoring candidate variables becoming free.
- *Fast Strategy:* This strategy does not use a metric to avoid sorting and priority queues. 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 together with any variable in  $L_i$  are added.



This strategy tries to maximize the speed with which local instances are constructed. Which strategy is best depends on the structure of the instance, as we will discuss in our experiments. We discuss further strategies and results in Appendix A.

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. For brevity, only some metrics are discussed here and more metrics and results can be found in Appendix B. We heavily use the concept of *units*: variable  $v$  is a unit of clause  $C$  with respect to an assignment  $\tau$  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:

- *Unit Metric*: For each variable  $v \in \text{var}(\mathcal{F})$  the score is

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

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

- *Satisfying Metric*:

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

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. For a soft clause  $C_s$ ,  $s(C_s) = \sum_{v \in \text{var}(C_s)} s(v)$ .

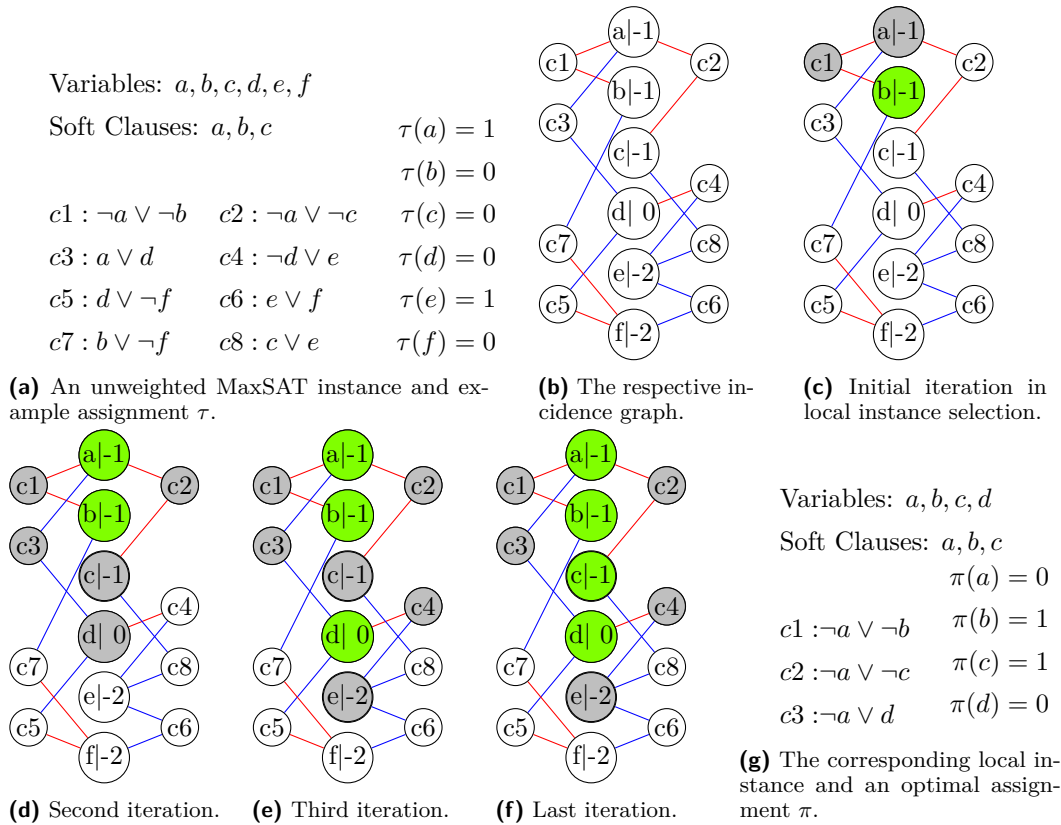
- *NuWLS Metric*: This metric uses the initial weighting scheme of NuWLS [7]. For unweighted instances, each soft clause  $C_s$  has score  $s(C_s) = 1000$ , for weighted instances, with  $w_a$  being the average weight over all soft clauses,  $s(C_s) = w(C_s) \cdot \frac{3000}{w_a}$ .

For the purposes of computing the variable score, let  $s(C) = 1$  for all  $C \in F_h$ . The score for a variable  $v$  is defined as

$$s(v) = \sum_{C_s \in F_s, C_s \cap \text{lit}(\tau) = \emptyset} s(C_s) - \sum_{C \in \text{unit}(v)} s(v).$$

This metric weighs the clauses that will be satisfied by changing the variable's value against the clauses that will become unsatisfied.

Instead of using the metric scores directly as discussed, we use *weighted random sampling*, similar to MaxSAT-LNS: 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 diverse local instances, leading to more improvements over time.



■ **Figure 1** Local instance selection using the Unit Metric, Variable Strategy, and a budget of 4. A red edge indicates that the variable occurs negated in a clause and a blue edge indicates that the variable occurs unnegated.

► **Example 1.** Figure 1 shows an example for local instance selection. Figure 1a shows an unweighted MaxSAT instance with three unary soft clauses and a sub-optimal assignment  $\tau$ . The corresponding incidence graph is shown in Figure 1b. The graph also shows the polarity of the variable using the edge color, blue for positive and red for negative. The example uses the Unit Metric: the weight for the variables is given with the variable name in the graph vertices.  $a$  has weight  $-1$  as it alone satisfied  $c3$  under  $\tau$ , while  $d$  has weight  $0$  because it only satisfies  $c4$ , which is also satisfied by  $e$  under  $\tau$ . Local instance selection starts in Figure 1c using the Variable Strategy, for clarity we do not use weighted random sampling. Initially,  $b$  and  $c$  could be chosen as they occur in unsatisfied soft clauses and have highest weight. Here,  $b$  is chosen arbitrarily as the initial soft clause and variable. The gray vertices have been explored by the strategy and only the variables corresponding to explored vertices are considered as candidate variables in the next iteration. Since  $a$  is the only such variable, it is added next in Figure 1d. In the next two iteration, the variables with the highest weights are selected, until we hit our budget of 4. We now have selected four candidate variables that are also free variables and the corresponding local instance is shown in Figure 1g. The improved assignment to the local instance can then be completed to a global assignment using the original assignments to  $e$  and  $f$ .

### 3.5 Local Solvers

MaxSLIM can use any MaxSAT solver as the local solver, whether it is an anytime or an exact solver. Next, we will 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. There are methods like stratification [2, 17] that find upper bounds for weighted instances during solving, but the upper bounds are often not tight until late in the solving process. Further, exact solvers cannot profit much from an already known solution, apart from hardening [2] which only works for weighted instances. Incremental exact solvers would support encoding the instance only once and reusing learned information across multiple local instances. Unfortunately, our experiments show that the large number of assumptions (up to hundreds of thousands and more) required for expressing the local instances slows down an incremental solver. Further, the large number of assumptions makes it hard for the solver to learn cores or clauses that are useful for another local instance. We could not observe any improvement of using an incremental solver over a non-incremental exact solver.

In MaxSLIM, we use our own exact solver based on the OLL algorithm [1, 19] 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 a 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 increases the optimal cost of the local instance. The increase in optimal cost, in turn, makes it easier and faster to determine whether no improvement is possible.

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.

## 4 Experimental Evaluation

In our experimental evaluation, we address several research questions:

- Q1:** Is there a benefit of a structured approach compared to an unstructured one like MaxSAT-LNS? (Section 4.2)
- Q2:** Is there a benefit of local improvement compared to just running the initial solver for the entire time? (Section 4.3)
- Q3:** How does MaxSLIM compare to other anytime solvers? (Section 4.4)
- Q4:** What strategies/metrics work best? (Section 4.5)

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

## 4.1 Experimental Setup

### 4.1.1 Cluster

The experiments were run on servers with two AMD EPYC 7402 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 all the solvers. We use timeouts of 5, 30 and 60 minutes.

### 4.1.2 Comparison

We compare our implementation of MaxSLIM against MaxSAT-LNS<sup>2</sup>, as well as the MaxSAT Evaluation 2023 solvers NuWLS-c (static), TT-Open-WBO-Inc (Glucose for unweighted, IntelSAT for weighted), Loandra, and NoSAT MaxSAT<sup>3</sup>. We utilize the same scoring system as the MaxSAT Evaluation: let  $c_{\text{best}}$  be the cost of the best known solution for the given instance, and  $c_{\text{solver}}$  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.

We perform three runs per solver and configuration, using three specific random seeds for reproducibility, whenever the solver supports it. The random seeds themselves have been initially randomly generated. If not stated otherwise, we give the average of the three runs.

We generate an initial solution using NuWLS-c, the winner of the MaxSAT Evaluation 2023, and aim for a comparability between MaxSLIM, MaxSAT-LNS, and NuWLS-c using the following setup. Instead of running NuWLS-c separately for 5, 30, and 60 minutes, we only run it for the 60 minute-runs and extract the current best solution after 1, 5, 30, and 60 minutes. These extracted solutions are then used as the initial solution for MaxSLIM and MaxSAT-LNS: for the 5 minute runs, MaxSLIM and MaxSAT-LNS get NuWLS-c’s best solution after one minute as an input; for 30 and 60 minute timeouts we give MaxSLIM and MaxSAT-LNS the NuWLS-c’s best solution after 5 minutes as the initial solution. We compensate for this by running MaxSLIM and MaxSAT-LNS for only 4, 25, and 55 minutes instead of 5, 30 and 60 minutes. This setup means that MaxSLIM, MaxSAT-LNS, and NuWLS-c always start from the same solution. Instances where no initial solution could be computed are omitted from the results.

We note that it is common that anytime solvers use other anytime solvers: TT-Open-WBO-Inc uses NuWLS-c, and NuWLS-c uses TT-Open-WBO-Inc [5]. Further, we did not try to create the best anytime solver, but evaluate how well our structured approach works. Interleaving our approach more with the other solvers would yield better results, but makes it hard to identify how much our approach contributes. Possible improvements are using NuWLS for those instances that do not allow local improvements, or optimizing the timeout for the initial solution.

### 4.1.3 Instances

We used the instances from the 2023 MaxSAT Evaluation’s anytime track<sup>4</sup>. The set contains 179 unweighted and 160 weighted instances. Our experiments show that EvalMaxSAT-SCIP, the winner of the unweighted track and close third-best solver in the weighted track, was able to solve 39 of the unweighted instances and 48 of the weighted instances within one hour.

<sup>2</sup> [https://github.com/rgh000/MaxSAT\\_LNS](https://github.com/rgh000/MaxSAT_LNS)

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

<sup>4</sup> <https://maxsat-evaluations.github.io/2023/benchmarks.html>

We restrict the results to those instances where we could find an initial solution. This avoids giving NuWLS-c a better score on instances where NuWLS-c does not find an initial solution, but later finds a solution within the timelimit. Hence, we avoid lowering MaxSLIM's and MaxSAT-LNS's score in case of NuWLS-c's poor performance.

#### 4.1.4 Configuration

We solve local instances using our OLL solver limited to a local timeout of 55 seconds. This admits local instance selection and solving the local instance to finish within a minute. As the budget, we initially use  $\frac{|\text{var}(\mathcal{F})|}{10}$  many variables, but not more than 25 000. Every five consecutive failures of finding an improvement, we increase the budget by another  $\frac{|\text{var}(\mathcal{F})|}{10}$ . Whenever the budget reaches the total number of variables, we run the local solver without a timeout on the whole instance. We use the Variable Strategy and either the Unit Metric for unweighted, or the NuWLS Metric for weighted instances.

■ **Table 1** Virtual best average scores for different combinations of solvers. Virtual best scores take for each instance the best solution over the specified solvers.

Solvers	Unweighted			Weighted		
	5 m	30 m	60 m	5 m	30 m	60 m
MaxSLIM	0.886	0.920	0.927	0.833	0.911	0.917
MaxSLIM & MaxLNS	0.896	0.930	0.936	0.849	0.927	0.933
MaxSLIM & NuWLS	0.901	0.937	0.940	0.888	0.928	0.933
All Solvers	0.929	0.970	0.980	0.947	0.976	0.981

## 4.2 Comparison of SLIM and LNS (Q1)

The comparison between MaxSAT-LNS and MaxSLIM in Tables 2 and 3 shows that MaxSLIM performs overall better than MaxSAT-LNS for all timeouts and scores. This suggests that the structured approach has an advantage over randomly selecting the local instances. In contrast to all the other solvers, MaxSAT-LNS and MaxSLIM both have a large variance between the runs. This suggests that weighted random sampling introduces a significant diversity among the local instances. The virtual best results in Table 1 show that MaxSAT-LNS and MaxSLIM are not very complementary. Interestingly, the difference between the virtual best and MaxSLIM's score is constant over the timeouts, suggesting that some improvements were only found by MaxSAT-LNS.

## 4.3 Comparison of MaxSLIM and NuWLS-c (Q2)

One crucial question is whether MaxSLIM is better than running the anytime solver used for the initial solution for the entire duration. The results in Table 2 show that MaxSLIM performs better on unweighted instances, where it is better on all metrics for all timeouts.

The results are different for weighted instances as shown in Table 3. Here, MaxSLIM performs worse for the 5-minute runs. The weighted instances contain on average three times more variables and hard clauses, as well as ten times more soft clauses compared to the unweighted instances. This increase in size decreases the performance of the solvers and leads to a poor initial solution. NuWLS-c is then faster at finding improvements than MaxSLIM. This is clearly visible in the much better results on higher timeouts, where MaxSLIM is able

## 26:12 Structure-Guided Local Improvement for Maximum Satisfiability

■ **Table 2** Comparison between MaxSLIM, MaxSAT-LNS (MaxLNS), NuWLS-c (NuWLS), TT-Open-WBO-Inc (TT-OpenWI), Loandra, and NoSAT MaxSAT (NoSAT) for unweighted instances. The score is obtained by taking for each instance the best scoring run, the worst scoring run, and the average over all three runs. *Best* shows on how many instances the solver found the best solution.

	MaxLNS	MaxSLIM	NuWLS	TT-OpenWI	Loandra	NoSAT
5-Minutes	164 Instances					
Score Min	0.859	<b>0.876</b>	<b>0.876</b>	0.872	0.806	0.576
Score Average	0.871	<b>0.887</b>	0.885	0.876	0.820	0.589
Score Max	0.887	<b>0.899</b>	0.895	0.881	0.835	0.602
Best	32	<b>42</b>	33	32	<b>42</b>	15
30-Minutes	171 Instances					
Score Min	0.895	0.902	<b>0.910</b>	0.896	0.884	0.577
Score Average	0.907	<b>0.919</b>	0.917	0.907	0.892	0.591
Score Max	0.928	<b>0.937</b>	0.925	0.917	0.900	0.606
Best	36	39	30	28	<b>41</b>	13
60-Minutes	171 Instances					
Score Min	0.898	0.907	<b>0.911</b>	0.902	0.887	0.587
Score Average	0.912	<b>0.923</b>	0.918	0.913	0.900	0.601
Score Max	0.930	<b>0.940</b>	0.926	0.925	0.912	0.617
Best	39	<b>41</b>	30	29	<b>41</b>	14

to find improvements overlooked by NuWLS-c. Even on the 5-minute timeout, MaxSLIM finds many improvements, overlooked by NuWLS-c, as highlighted by the high virtual best score in Table 1.

The gap between the virtual best and the best score behaves similar to the comparison with MaxSAT-LNS. Except for the 5-minute weighted run, the gap remains almost constant over the timeouts, suggesting that some improvements are not found by MaxSLIM. One type of instance where NuWLS-c performs better are instances, where improvements are only possible on (almost) the full instance, since solving these instances with an exact solver is slow.

### 4.4 Other Solvers (Q3)

Tables 2 and 3 also show the results for the other solvers from the 2023 MaxSAT Evaluation. Striking is the large variance for every solver, showing that luck plays a significant role in the score. MaxSLIM’s variance stays comparatively high even for the longer timeouts, indicating that longer timeouts could result in further significant improvements.

In the unweighted case, MaxSLIM is the best solver over the different timeouts, but it clearly distinguishes itself from NuWLS-c only for the best score, as the average score is only better by a small margin.

The weighted case is similar to the comparison with NuWLS-c for the same reasons. Hence, MaxSLIM performs best for the longer timeouts, where the improvements become harder to find.

■ **Table 3** Comparison between MaxSLIM, MaxSAT-LNS (MaxLNS), NuWLS-c (NuWLS), TT-Open-WBO-Inc (TT-OpenWI), Loandra, and NoSAT MaxSAT (NoSAT) for weighted instances. The score is from taking for each instance the best scoring run, the worst scoring run, and the average over all three runs. *Best* shows on how many instances the solver found the best solution.

	MaxLNS	MaxSLIM	NuWLS	TT-OpenWI	Loandra	NoSAT
5-Minutes	154 Instances					
Score Min	0.797	0.813	0.857	<b>0.862</b>	0.818	0.307
Score Average	0.818	0.833	0.868	<b>0.872</b>	0.843	0.319
Score Max	0.839	0.851	<b>0.882</b>	0.881	0.868	0.331
Best	32	29	31	22	<b>35</b>	0
30-Minutes	159 Instances					
Score Min	0.877	<b>0.895</b>	0.889	0.881	0.882	0.327
Score Average	0.894	<b>0.911</b>	0.889	0.888	0.893	0.338
Score Max	0.914	<b>0.929</b>	0.912	0.895	0.906	0.348
Best	34	<b>42</b>	17	17	33	0
60-Minutes	159 Instances					
Score Min	0.881	<b>0.903</b>	0.896	0.889	0.887	0.330
Score Average	0.899	<b>0.917</b>	0.906	0.897	0.898	0.341
Score Max	0.918	<b>0.933</b>	0.914	0.906	0.910	0.352
Best	35	<b>41</b>	19	20	34	0

Hence, MaxSLIM is generally competitive, but needs more local search for better performance on short weighted runs. The virtual best score over all solvers in Table 1 shows that the approaches are quite complementary, as the virtual best for 5-minutes would outperform the single best after one hour.

#### 4.5 Impact of the Configuration (Q4)

An interesting question is how the hyperparameters impact MaxSLIM's performance. We ran the same configuration used in the previous experiments and varied only a single parameter. The virtual best scores over all the configurations discussed in this section are 0.926 for unweighted and 0.867 for weighted instances. Hence, a good dynamic configuration could severely improve MaxSLIM's performance.

Table 4 shows the results of disabling some of MaxSLIM's features.

The upper bounding search does have a noticeable impact. Nonetheless, disabling it yields better results on some instances. The varying performance comes from the fact that the upper bounding search is slower than normal search whenever the local instance cannot be improved.

Weighted random sampling performs in a similar manner, but improves the performance overall more than upper bounding search. There is no clear indication as to when sampling is beneficial and when it is not.

Local search has the biggest impact on the results. Whenever we take the first solution from the initial solver, instead of letting it run the full minute, the performance degrades significantly. This is particularly impactful on weighted instances. MaxSLIM is comparatively slow, but can find improvements not visible to local search. Local search is complementary,

■ **Table 4** Performance of different disabled features of MaxSLIM over a 5-minute timelimit. *Improved* shows how many input solutions were improved. *Better* and *Worse* show on how many instances the configuration did better or worse than the baseline.

Configuration	Unweighted				Weighted			
	Score	Improved	Better	Worse	Score	Improved	Better	Worse
Baseline	0.887	97	-	-	0.833	105	-	-
No Upper Bounding Search	0.878	94	16	32	0.825	101	16	33
No Weighted Sampling	0.875	88	22	40	0.819	103	15	40
No Initial NuWLS Solution	0.817	-	20	94	0.650	-	10	102

as it is very good at finding improvements fast, but often gets stuck when improvements become hard to find. Hence, not running local search severely degrades the performance for short runtimes.

Next, we discuss results on using different strategies and metrics.

#### 4.5.1 Strategies

Table 5 shows the results of comparing different strategies. The results show that the Variable Strategy is overall the best strategy, while the 5-Adjacency finds many solutions missed by the Variable Strategy. The Fast Strategy performs overall worst, but performs better on some very large instances. In general, different strategies perform complementarily.

■ **Table 5** Performance of different strategies. *Best* Score shows on how many instances MaxSLIM found the best solution using the given strategy. *Unique Best* shows on how many instances the best solution could only be found using the given strategy. The timelimit was 5 minutes.

Strategy	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Variable	<b>0.887</b>	<b>97</b>	44	5	<b>0.833</b>	105	34	14
5-Adjacency	<b>0.887</b>	94	<b>48</b>	<b>13</b>	0.822	<b>107</b>	<b>40</b>	<b>18</b>
Fast	0.873	79	28	5	0.817	99	24	10

In Table 6 are several statistics for each strategy. We can see that the relative time spent on constructing local instances is indeed significantly lower for the fast strategy, and about the same for all other strategies. Interestingly, the number of local instances is much higher for the Variable and 5-Adjacency Strategy. This shows that the fast strategy extracts many local instances that do not lead to an improvement, but require long solving times.

Another interesting observation is that neither the number of local instances, nor the number of improved local instances is a good indicator for performance. According to both results, the Variable Strategy would not be the best strategy.

#### 4.5.2 Metrics

Table 7 shows the results of comparing different metrics. The score between the best and the worst metric generally does not vary much. The Unit Metric is in the absence of weights the best overall metric. Unsurprisingly, for weighted instances, those metrics that use the weights work better.



■ **Table 6** Strategy statistics averaged over all instances: average local instance size relative to the full instance (LI Size), average ratio of free variables to candidate variable per local instance (F/C), average fraction of runtime spent on local instance generation (LI Time Ratio), average fraction of variables changed per improvement (Changes), average ratio of local instances that have been solved optimally (Optimal Ratio), average number of local instances (#LI), average number of local instances that led to an improvement (#Improved).

Strategy	LI Size	F/C	LI Time Ratio	Changes	Optimal Ratio	#LI	#Improved
Unweighted							
Variable	0.40	0.64	0.10	0.06	0.64	49.11	2.01
5-Adjacency	0.38	0.65	0.12	0.07	0.67	55.49	3.29
Fast	0.41	0.69	0.02	0.07	0.50	32.29	1.69
Weighted							
Variable	0.23	0.52	0.21	0.12	0.42	33.33	3.27
5-Adjacency	0.28	0.57	0.24	0.10	0.44	29.71	3.55
Fast	0.35	0.60	0.12	0.12	0.35	23.16	1.54

The statistics in Table 8 show that the different metrics perform very similarly. Together with the strategy statistics in Table 6 there are some interesting details regarding the performance differences between weighted and unweighted instances. The local instances for weighted instances are much smaller than for unweighted instances, while constructing them takes significantly longer. The number of variables changed and optimal ratio also show that it is harder to find the improvements for weighted instances. This further explains the poorer performance on weighted instances compared to unweighted instances.

■ **Table 7** Performance of different metrics. *Best* Score shows on how many instances MaxSLIM found the best solution using the given metric. *Unique Best* shows on how many instances the best solution could only be found using the given metric. The timelimit was 5 minutes.

Metric	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Unit	<b>0.887</b>	<b>97</b>	51	5	0.823	103	<b>47</b>	<b>17</b>
NuWLS	0.885	<b>97</b>	51	8	<b>0.833</b>	<b>105</b>	44	9
Satisfying	0.883	96	<b>66</b>	<b>13</b>	0.824	103	36	7

## 5 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 which have been refined for several years, and other LNS approaches. MaxSLIM's trajectory of improvements over time is particularly attractive for applications with longer runtimes.

■ **Table 8** Metric statistics averaged over all unweighted instances: average local instance size relative to the full instance (LI Size), average ratio of free variables to candidate variable per local instance (F/C), average fraction of runtime spent on local instance generation (LI Time Ratio), average fraction of variables changed per improvement (Changes), average ratio of local instances that have been solved optimally (Optimal Ratio), average number of local instances ( $\#LI$ ), average number of local instances that led to an improvement ( $\#Improved$ ).

Metric	LI Size	F/C	LI Time Ratio	Changes	Optimal Ratio	$\#LI$	$\#Improved$
Unweighted							
Unit	0.40	0.64	0.10	0.06	0.64	49.11	2.01
NuWLS	0.40	0.63	0.10	0.07	0.63	49.78	2.16
Satisfying	0.42	0.66	0.11	0.06	0.57	47.00	1.78
Weighted							
Unit	0.27	0.54	0.20	0.10	0.42	31.72	3.59
NuWLS	0.23	0.52	0.21	0.12	0.42	33.33	3.27
Satisfying	0.27	0.57	0.22	0.10	0.39	30.87	3.40

Our evaluation uses a default configuration and our results show that choosing the parameters – strategy, metric, timeouts – according to the application can significantly improve MaxSLIM’s performance even further.

An interesting avenue for further research is to adapt other anytime solvers to integrate better within MaxSLIM. This would allow us to interleave local improvement phases with additional runs of the initial solver, starting from the best solution found so far. Such an interleaved SLIM approach has shown to be surprisingly powerful for circuit minimization [30, 31]. The large variance when comparing the score of different runs for the same configuration, as well as different configurations, indicates that interleaving MaxSLIM with itself, using different strategies and metrics, may also be beneficial for the result. Although MaxSLIM can be parallelized, results so far show that this is only beneficial on some instances, but further improvements may be possible. As MaxSLIM can benefit from the tuning of its parameters, we expect that further efficiency improvements can be obtained through automated algorithm configuration, possibly even adjusting parameters during the run [3].

---

## References

- 1 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPICs*, pages 211–221. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.ICLP.2012.211.
- 2 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012. doi:10.1007/978-3-642-33558-7\_9.
- 3 Carlos Ansótegui, Josep Pon, Meinolf Sellmann, and Kevin Tierney. Reactive dialectic search portfolios for MaxSAT. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 765–772. AAAI Press, 2017. URL: <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14872>.

- 4 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2019. doi:10.1007/978-3-030-19212-9\_3.
- 5 Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen. Maxsat evaluation 2023: Solver and benchmark descriptions. Technical report, University of Helsinki, 2023.
- 6 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.*, 287:103354, 2020. doi:10.1016/J.ARTINT.2020.103354.
- 7 Yi Chu, Shaowei Cai, and Chuan Luo. NuWLS: improving local search for (weighted) partial MaxSAT by new weighting techniques. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *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*, pages 3915–3923. AAAI Press, 2023. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/25505>.
- 8 Aviad Cohen, Alexander Nadel, and Vadim Ryvchin. Local search with a SAT oracle for combinatorial optimization. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *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*, volume 12652 of *Lecture Notes in Computer Science*, pages 87–104. Springer, 2021. doi:10.1007/978-3-030-72013-1\_5.
- 9 Emir Demirovic and Nysret Musliu. MaxSAT-based large neighborhood search for high school timetabling. *Comput. Oper. Res.*, 78:172–180, 2017. doi:10.1016/j.cor.2016.08.004.
- 10 Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. 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. URL: <https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-synthesis-semantic-malware-signatures-using-maximum-satisfiability/>.
- 11 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. SAT-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411. Springer Verlag, 2017. doi:10.1007/978-3-319-66263-3\_25.
- 12 Randy Hickey and Fahiem Bacchus. Large neighbourhood search for anytime MaxSAT solving. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1818–1824. International Joint Conferences on Artificial Intelligence Organization, July 2022. doi:10.24963/ijcai.2022/253.
- 13 Sepideh Khoshnood, Markus Kusano, and Chao Wang. ConcBugAssist: constraint solving for diagnosis and repair of concurrency bugs. In Michal Young and Tao Xie, editors, *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 165–176. ACM, 2015. doi:10.1145/2771783.2771798.
- 14 Alexander S. Kulikov, Danila Pechenev, and Nikita Slezkin. SAT-based circuit local improvement. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPICs*, pages 67:1–67:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.67.

- 15 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 179–195. Springer Verlag, 2016. doi:10.1007/978-3-319-40970-2\_12.
- 16 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for special treewidth and pathwidth. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 429–445. Springer Verlag, 2017. doi:10.1007/978-3-319-66263-3\_27.
- 17 João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4):317–343, 2011. doi:10.1007/S10472-011-9233-2.
- 18 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In Carsten Sinz and Uwe Egly, editors, *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*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. doi:10.1007/978-3-319-09284-3\_33.
- 19 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O’Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7\_41.
- 20 António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints An Int. J.*, 18(4):478–534, 2013. doi:10.1007/s10601-013-9146-2.
- 21 Alexander Nadel. Solving MaxSAT with bit-vector optimization. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2018. doi:10.1007/978-3-319-94144-8\_4.
- 22 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In Clark W. Barrett and Jin Yang, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 193–202. IEEE, 2019. doi:10.23919/FMCAD.2019.8894273.
- 23 Alexander Nadel. On optimizing a generic function in SAT. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 205–213. IEEE, 2020. doi:10.34727/2020/ISBN.978-3-85448-042-6\_28.
- 24 Alexander Nadel. Polarity and variable selection heuristics for sat-based anytime maxsat. *J. Satisf. Boolean Model. Comput.*, 12(1):17–22, 2020. doi:10.3233/sat-200126.
- 25 David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of Metaheuristics*, pages 399–419. Springer Verlag, 2010.
- 26 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. MaxSAT-based postprocessing for treedepth. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 478–495. Springer, 2020. doi:10.1007/978-3-030-58475-7\_28.
- 27 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Learning fast-inference bayesian networks. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 17852–17863, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/94e70705efae423efda1088614128d0b-Abstract.html>.

- 28 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. 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*, pages 3895–3903. AAAI Press, 2021. doi:10.1609/aaai.v35i5.16508.
- 29 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Learning large Bayesian networks with expert constraints. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1592–1601. PMLR, 01–05 August 2022. URL: <https://proceedings.mlr.press/v180/peruvemba-ramaswamy22a.html>.
- 30 Franz-Xaver Reichl, Friedrich Slivovsky, and Stefan Szeider. Circuit minimization with QBF-based exact synthesis. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *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*, pages 4087–4094. AAAI Press, 2023. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/25524>.
- 31 Franz-Xaver Reichl, Friedrich Slivovsky, and Stefan Szeider. eSLIM: Circuit minimization with SAT based local improvement. In *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India, 2024*. to appear.
- 32 André Schidler. SAT-based local search for plane subgraph partitions (CG challenge). In Xavier Goaoc and Michael Kerber, editors, *38th International Symposium on Computational Geometry, SoCG 2022, June 7-10, 2022, Berlin, Germany*, volume 224 of *LIPICs*, pages 74:1–74:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SoCG.2022.74.
- 33 André Schidler and Stefan Szeider. 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. doi:10.1609/aaai.v35i5.16509.
- 34 André Schidler and Stefan Szeider. SAT-boosted tabu search for coloring massive graphs. *ACM J. Exp. Algorithmics*, 28, July 2023. doi:10.1145/3603112.
- 35 André Schidler and Stefan Szeider. SAT-based decision tree learning for large data sets. *J. Artif. Intell. Res.*, 80:875–918, 2024.
- 36 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael J. Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998. doi:10.1007/3-540-49481-2\_30.
- 37 Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 2017. doi:10.1007/978-3-319-63387-9\_4.
- 38 Charlie Shucheng Zhu, Georg Weissenbacher, Divyot Sethi, and Sharad Malik. Sat-based techniques for determining backbones for post-silicon fault localisation. In Zeljko Zilic and Sandeep K. Shukla, editors, *2011 IEEE International High Level Design Validation and Test Workshop, HLDVT 2011, Napa Valley, CA, USA, November 9-11, 2011*, pages 84–91. IEEE Computer Society, 2011. doi:10.1109/HLDVT.2011.6113981.

## A Strategies

We use the same definitions as in Section 3.3: given a global instance  $\mathcal{F} = (F_h, F_s, w)$ , a global solution  $\tau$  for  $\mathcal{F}$ , and a metric  $s : \text{var}(\mathcal{F}) \cup F_s \rightarrow \mathbb{R}$  (which can depend on  $\tau$ ), we use the following additional strategies for extending  $L_i$  to  $L_{i+1}$ , where ties are always broken arbitrarily:

- *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.
- *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.
- *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. We will further discuss the runtimes in the experimental results in the next section.

### A.1 Strategy Results

■ **Table 9** Performance of different strategies. *Best* Score shows on how many instances MaxSLIM found the best solution using the given strategy. *Unique Best* shows on how many instances the best solution could only be found using the given strategy. The timelimit was 5 minutes.

Strategy	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Global	<b>0.906</b>	92	<b>64</b>	<b>22</b>	0.826	104	35	14
Variable	0.887	<b>97</b>	44	5	<b>0.833</b>	105	34	14
5-Adjacency	0.887	94	48	13	0.822	<b>107</b>	<b>40</b>	<b>18</b>
Clause	0.877	95	36	1	0.816	106	28	8
Adjacency	0.873	92	36	4	0.820	106	29	13
Fast	0.873	79	28	5	0.817	99	24	10

Table 9 shows the results of comparing different strategies. The results show that the Variable Strategy is overall the best strategy, while the Global Strategy performs better for unweighted instances. Nonetheless, every strategy has several instances where it is the only



■ **Table 10** Strategy statistics averaged over all instances: average local instance size relative to the full instance (LI Size), average ratio of free variables to candidate variable per local instance (F/C), average fraction of runtime spent on local instance generation (LI Time Ratio), average fraction of variables changed per improvement (Changes), average ratio of local instances that have been solved optimally (Optimal Ratio), average number of local instances (#LI), average number of local instances that led to an improvement (#Improved).

Strategy	LI Size	F/C	LI Time Ratio	Changes	Optimal Ratio	#LI	#Improved
Unweighted							
Global	0.40	-	0.02	0.07	0.73	77.79	4.25
Variable	0.40	0.64	0.10	0.06	0.64	49.11	2.01
5-Adjacency	0.38	0.65	0.12	0.07	0.67	55.49	3.29
Clause	0.33	0.61	0.11	0.07	0.65	55.35	2.13
Adjacency	0.36	0.66	0.10	0.07	0.69	50.11	2.65
Fast	0.41	0.69	0.02	0.07	0.50	32.29	1.69
Weighted							
Global	0.24	-	0.09	0.11	0.50	46.59	4.30
Variable	0.23	0.52	0.21	0.12	0.42	33.33	3.27
5-Adjacency	0.28	0.57	0.24	0.10	0.44	29.71	3.55
Clause	0.24	0.54	0.23	0.06	0.48	34.06	3.33
Adjacency	0.20	0.57	0.24	0.09	0.45	28.09	3.28
Fast	0.35	0.60	0.12	0.12	0.35	23.16	1.54

one that could find the best solution. The difference in score is significantly larger among unweighted instances compared to weighted instances. Hence, it is easier to give a good default strategy for unweighted instances than for weighted instances.

In Table 10 are several statistics for each strategy. We can see that the relative time spent on constructing local instances is low for the Global Strategy and the Fast Strategy, and about the same for all other strategies. Nonetheless, the number of local instances is much higher for the global strategy than for the fast strategy. This shows that the fast strategy extracts many local instances that do not lead to an improvement, but require long solving times. This also explains the good performance of the global strategy on unweighted instances: the high number of local instances shows that MaxSLIM increases the budget faster than with the other strategies. Further, the global strategy is good at finding many small improvements, which has more overall impact on unweighted instances than on weighted instances.

Generally, the statistics are more homogenous for unweighted instances than for weighted instances, which also explains the results in Table 9. Another interesting observation is that neither the number of local instances, nor the number of improved local instances is a good indicator for performance. According to both results, the Variable Strategy would be among the worst strategies.

## B Metrics

We use the definitions from Section 3.4: a variable  $v$  is a unit of clause  $C$  with respect to an assignment  $\tau$  if  $\{v\} = \text{var}(\text{lit}(\tau) \cap C)$  and  $\text{unit}(v) = \{C \in F_s \cup F_h : \{v\} = \text{var}(\text{lit}(\tau) \cap C)\}$ .

We define the following additional 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.
- *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 [12]. 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  $s(C) = \sum_{v \in \text{var}(C)} s(v)$ .

- *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  $\min_{v \in \text{var}(C)} s(v)$ .

## B.1 Metric Results

■ **Table 11** Performance of different metrics. *Best* Score shows on how many instances MaxSLIM found the best solution using the given metric. *Unique Best* shows on how many instances the best solution could only be found using the given metric. The time limit was 5 minutes.

Metric	Unweighted				Weighted			
	Score	Improved	Best	Unique	Score	Improved	Best	Unique
Unit	<b>0.887</b>	<b>97</b>	51	5	0.823	103	<b>47</b>	<b>17</b>
NuWLS	0.885	<b>97</b>	51	8	<b>0.833</b>	<b>105</b>	44	9
Satisfying	0.883	96	<b>66</b>	<b>13</b>	0.824	103	36	7
LNS	0.882	96	56	12	0.823	103	41	14
Counting	0.881	92	45	2	0.822	104	38	7
Random	0.881	91	39	0	0.823	102	32	5
Ratio	0.879	92	46	3	0.822	103	32	6

Table 11 shows the results of comparing different metrics. The score between the best and the worst metric generally does not vary much. Nonetheless, apart from the Random Metric, every metric achieves several unique best scores. The Unit Metric is in the absence of weights the best overall metric. Unsurprisingly, for weighted instances, those metrics that use the weights work better.

Table 12 shows various statistics for these metrics. In general, most statistics do not vary much between metrics, except for the number of improving local instances.

■ **Table 12** Metric statistics averaged over all unweighted instances: average local instance size relative to the full instance (LI Size), average ratio of free variables to candidate variable per local instance (F/C), average fraction of runtime spent on local instance generation (LI Time Ratio), average fraction of variables changed per improvement (Changes), average ratio of local instances that have been solved optimally (Optimal Ratio), average number of local instances (#LI), average number of local instances that led to an improvement (#Improved).

Metric	LI Size	F/C	LI Time Ratio	Changes	Optimal Ratio	#LI	#Improved
Unweighted							
Unit	0.40	0.64	0.10	0.06	0.64	49.11	2.01
NuWLS	0.40	0.63	0.10	0.07	0.63	49.78	2.16
Satisfying	0.42	0.66	0.11	0.06	0.57	47.00	1.78
LNS	0.43	0.64	0.09	0.06	0.56	46.32	1.79
Counting	0.40	0.63	0.10	0.06	0.64	46.87	1.72
Random	0.40	0.63	0.10	0.06	0.63	46.79	1.76
Ratio	0.40	0.64	0.11	0.06	0.64	49.99	2.07
Weighted							
Unit	0.27	0.54	0.20	0.10	0.42	31.72	3.59
NuWLS	0.23	0.52	0.21	0.12	0.42	33.33	3.27
Satisfying	0.27	0.57	0.22	0.10	0.39	30.87	3.40
LNS	0.28	0.54	0.20	0.09	0.39	31.23	3.46
Counting	0.25	0.53	0.20	0.10	0.40	31.16	3.23
Random	0.25	0.53	0.20	0.08	0.38	30.40	3.15
Ratio	0.25	0.54	0.21	0.08	0.39	30.58	3.52