Enhancing MaxSAT Local Search via a Unified Soft Clause Weighting Scheme

Yi Chu
Institute of Software, Chinese Academy of Sciences, Beijing, China

Chu-Min Li
MIS, UR 4290, Université de Picardie Jules Verne, Amiens, France
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Furong Ye
Key Laboratory of System Software, Chinese Academy of Sciences, Beijing, China
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Shaowei Cai[1]
Key Laboratory of System Software, Chinese Academy of Sciences, Beijing, China
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Abstract

Local search has been widely applied to solve the well-known (weighted) partial MaxSAT problem, significantly influencing many real-world applications. The main difficulty to overcome when designing a local search algorithm is that it can easily fall into local optima. Clause weighting is a beneficial technique that dynamically adjusts the landscape of search space to help the algorithm escape from local optima. Existing works tend to increase the weights of falsified clauses, and such strategies may result in an unpredictable landscape of search space during the optimization process. Therefore, in this paper, we propose a Unified Soft Clause Weighting Scheme called Unified-SW, which increases the weights of all soft clauses in feasible local optima, whether they are satisfied or not, while preserving the hierarchy among them. We implemented Unified-SW in a new local search solver called *USW-LS*. Experimental results demonstrate that *USW-LS*, outperforms the state-of-the-art local search solvers across benchmarks from anytime tracks of recent MaxSAT Evaluations. More promisingly, a hybrid solver combining *USW-LS* and TT-Open-WBO-Inc won all four categories in the anytime track of MaxSAT Evaluation 2023.

2012 ACM Subject Classification Theory of computation → Randomized local search

Keywords and phrases Weighted Partial MaxSAT, Local Search Method, Weighting Scheme

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.8

Supplementary Material *Software (Source Code)*: `https://github.com/filyouzicha/USW-LS`
archived at `swh:1:dir:d24759a3c14cbec511b97300cf55409c9f6cc9bf`

Funding This work was supported by the National Natural Science Foundation of China under Grant 62302492, and the French Agence Nationale de la Recherche, via AI chair ANR-19-CHIA-0013-01 (Massal'IA).

1 Introduction

Given a propositional formula in conjunctive normal form (CNF), the maximum satisfiability problem (MaxSAT) aims to find an assignment that maximizes the number of satisfied clauses. Nowadays, research on MaxSAT mostly focuses on the partial MaxSAT (PMS), in

[1] Corresponding author.

which clauses are divided into hard and soft ones, and a feasible assignment, i.e., solution, requires all hard clauses to be satisfied, and an optimal solution should maximize the number of satisfied soft clauses. PMS can be generalized to weighted PMS (WPMS) by assigning positive integer weights to soft clauses to establish a hierarchy of importance among them, and the objective is to find an assignment that satisfies all hard clauses and maximizes the sum of weights of satisfied soft clauses. Note that when all weights are identical, WPMS becomes PMS. Therefore, we use WPMS to represent both WPMS and PMS. By focusing on WPMS, we are focusing on both WPMS and PMS.

In addition to obtaining much attention in computer science theory, WPMS has also been studied well in practical application domains since it can be applied to solve many real-world optimization problems such as scheduling [20], the maximum clique problem [32], FPGA routing [21], and computational protein design [2]. State-of-the-art WPMS solvers can be either exact or anytime. Exact solvers are guaranteed to find an optimal solution, and anytime solvers can quickly find a solution. With decades of development, Exact and anytime solvers have both achieved success in solving WPMS, and they are highly complementary. Recent annual MaxSAT Evaluations[2] include two main tracks. One is to assess exact solvers, and the other is to assess anytime solvers with the WPMS instances that no exact solver can solve within 1 hour. Exact solvers mainly comprise two categories: SAT-based [4, 19, 25, 5, 44, 35, 36, 6, 8, 9] and Branch-and-Bound (BnB) [31, 22, 27, 26, 1, 16, 30, 33]. Leading anytime solvers are hybrid [17, 43, 48, 29]: they apply LS for preprocessing and then resort to SAT-based methods. This paper focuses on the LS component of solvers for WPMS.

The LS method for WPMS iteratively chooses a variable and flips its value (from *True* to *False*, or from *False* to *True*) following a greedy strategy to satisfy all hard clauses and maximize the sum of weights of satisfied soft clauses. The main difficulty is that it can frequently fall into local optima, which means that flipping any variable cannot increase the number of satisfied hard clauses or the sum of weights of satisfied soft clauses. Many approaches have been introduced to overcome this difficulty, among which perhaps the most successful is *clause weighting* that is used in recent state-of-the-art LS solvers, such as SATLike [28], NuWLS [18], and BandMaxSAT [50], etc. In practice, a clause weighting scheme favors increasing the weights of falsified clauses so that their variables have greater chances to be flipped subsequently to satisfy them.

While various heuristics have been proposed for designing weighting schemes [15, 46, 45, 14, 28, 18], existing work faces a key issue: *the landscape of search space is blindly adjusted with clause weighting because the relations among the soft clauses, as well as the relations between hard clauses and soft clauses, are disturbed,* and solvers have to search for better solutions in a blindly adjusted search space, so that they can easily be misled when choosing the next variable to flip.

In this paper, we carefully analyze the questions to answer when designing a clause weighting scheme and propose a novel weighting scheme named Unified-SW (Unified Soft Clause Weighting), which distinguishes feasible local optima from infeasible local optima, and increases the weights of all soft clauses in feasible local optima, whether they are satisfied or not, while preserving the hierarchy among them, which is quite different from some previous approaches which tend to only increase the weights of falsified clauses.

Based on Unified-SW, we develop a new LS solver named *USW-LS* for WPMS. We compare *USW-LS* with state-of-the-art LS solvers on unweighted and weighted benchmarks from the anytime tracks of MaxSAT Evaluations from 2018 to 2023. Experimental results

[2] `https://maxsat-evaluations.github.io`

demonstrate that *USW-LS* outperforms the competing solvers in terms of both fixed-budget performance and anytime performance. In addition, *USW-LS* considerably advances the state-of-the-art in WPMS solving. Notably, the hybrid solver that combines *USW-LS* and TT-Open-WBO-Inc [41] has won all four categories in the anytime track of MaxSAT Evaluation 2023.

The rest of the paper is organized as follows. Section 2 presents some notions used in the paper. Section 3 presents related works. Section 4 analyzes the questions to answer when designing a clause weighting scheme, describes Unified-SW and its distinguishing properties, and presents our new LS solver *USW-LS* based on Unified-SW. Section 5 empirically demonstrates the performance of *USW-LS* and the effectiveness of Unified-SW. Section 6 concludes.

2 Preliminaries

Consider a set of n Boolean variables, denoted as $\{x_1, x_2, \cdots, x_n\}$. Corresponding to these variables is the set of $2n$ literals, defined as $\{x_1, \neg x_1, x_2, \neg x_2, \cdots, x_n, \neg x_n\}$, where each literal represents either a variable or its negation. A clause c of length k is a disjunction of k literals, denoted as $c = l_1 \vee l_2, \cdots, \vee l_k$. A propositional formula F in Conjunctive Normal Form (CNF) is a conjunction of clauses, denoted as $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$. For a CNF formula F, a complete assignment is a mapping that assigns each variable in F to a Boolean value (*True* or *False*). In this paper, an assignment is always complete. A literal x ($\neg x$) is satisfied if x is assigned *True* (*False*), otherwise, it is falsified. Each clause has two states: satisfied or falsified. Given an assignment α, a clause c is satisfied if at least one literal in c is *True*; otherwise, c is falsified.

PMS divides clauses in a CNF formula into hard and soft ones, and its objective is to find an assignment that maximizes the number of satisfied soft clauses while satisfying all hard clauses. For WPMS, each soft clause has a positive integer weight representing a cost if the soft clause is falsified, so that there is a hierarchy of importance among soft clauses, and the goal is to find an assignment that maximizes the total weight of satisfied soft clauses while satisfying all hard clauses. PMS represents a special case of WPMS in which the weight of each soft clause is identical, represented by 1.

An assignment is feasible if it satisfies all hard clauses. A feasible assignment is also called a *solution*. The cost of an assignment α is the sum of weights of falsified soft clauses, denoted as $cost(\alpha)$. An optimal solution is obtained by minimizing its cost while satisfying all hard clauses. A solution α_1 is considered to be better than another solution α_2, if $cost(\alpha_1) < cost(\alpha_2)$.

Clause weighting schemes assign weights to both hard and soft clauses, and maintain them. These weights are different from the original weights of soft clauses in WPMS instances. To avoid confusion, we will call the weights assigned and maintained by a clause weighting scheme *maintenance weights* or simply *weights*, but systematically add the word "original" when talking about the original weights of soft clauses in WPMS instances. The maintenance weight of a (hard or soft) clause c is denoted $w(c)$, and the original weight of a soft clause is denoted $w_{ori}(c)$. Note that $w_{ori}(c)$ is never changed, differently from $w(c)$. The average original weight of all soft clauses is denoted $avgw_{soft}$.

The maintenance weights are used to calculate the score of each variable. The score of variable x, denoted $score(x)$, is calculated as $score(x) = make(x) - break(x)$, where $make(x)$ is the sum of maintenance weights of (hard and soft) clauses changing from falsified to satisfied upon flipping x, and $break(x)$ is the sum of maintenance weights of (hard and soft) clauses changing from satisfied to falsified upon flipping x.

Restart strategy has commonly been applied when applying LS for WPMS. When restarting a local search, apart from initializing the current assignment, the maintenance weights assigned to each clause may also be reset.

3 Related Work

Solvers for WPMS can be categorized into exact and anytime types. Exact solvers can prove the optimality of their solutions when the algorithm terminates, and these methods have made significant progress over decades. In particular, the SAT-based solvers, which are built based on classic SAT solvers, have gone through a series of advancements [21, 3, 4, 19, 25, 5, 44, 35, 36, 6, 9]. BnB MaxSAT solvers [22, 27, 26, 1, 16, 33] implement the branch-and-bound method, detecting inconsistent subsets of soft clauses through unit propagation and lower bound computation. The performance of the BnB solvers has been significantly enhanced by integrating clause learning and an efficient bounding procedure [33].

While unable to prove solution optimality, LS solvers often obtain high-quality solutions within short runtimes and exhibit complementarity with exact solvers when solving various types of WPMS instances. Early LS algorithms encoded PMS instances as weighted MaxSAT instances, which were then solved accordingly [23].

LS algorithms for solving WPMS have been developed along a line of using clause weighting schemes to guide search. These algorithms assign weights to clauses and calculate the score of variables using these weights. So, they intensify or diversify the search primarily by maintaining these assigned weights in different ways.

Note that in PMS, as well as in WPMS, a hard clause is definitely more important than all soft clauses. So, a natural way to assign weights to clauses is to set the weight of each hard clause larger than the sum of all soft clause weights. However, this approach biases the search towards satisfying hard clauses, significantly limiting the search space and often negatively affecting the performance of LS algorithms [15].

As the above natural way to assign weights to clauses is ineffective, researchers spent much effort designing better clause weighting schemes for WPMS for a long time. Examples of early clause weighting schemes can be found in [15] for statically assigning suitable weights to hard clauses or in [46, 45] for dynamically adjusting hard clause weights during the search.

Below, we go over recent clause weighting schemes, included in the state-of-the-art algorithms Dist [14], SATLike [28] and NuWLS [18].

Dist uses a weighting scheme that only updates the hard clause weights. It introduces the concepts of hard score and soft score and employs different heuristic methods for hard clauses and soft clauses. The Dist series algorithms [13, 34] greatly improve the performance compared to previous LS algorithms for WPMS. However, the concepts and heuristic methods are separately defined for hard and soft clauses, increasing the algorithm complexity. Moreover, the algorithms prioritize satisfying hard clauses, limiting the search space. Subsequently, a series of algorithms based on Dist was designed to exploit the structure of WPMS further.

SATLike [28] introduces a weighting scheme called Weighting-PMS, which utilizes different increments to update the weights of hard and soft clauses, and sets a uniform upper bound for all soft clause weights. Weighting-PMS is also used in SATLike3.0 [12] and BandMaxSAT [50].

Based on an empirical finding that initial soft clause weights have a clear effect on the effectiveness of the algorithm adopting Weighting-PMS, NuWLS [18] uses a new weighting scheme named Dist-Weighting. There are three main distinctions between Dist-Weighting and previous weighting schemes. First, Dist-Weighting initializes soft clause weights within a reasonable range. Second, it distinguishes conditions for updating hard and soft clause weights. Third, it associates a specific upper bound with each soft clause based on its original weight, instead of setting it uniformly as in Weighting-PMS.

Unfortunately, despite the above progress, designing effective clause weighting schemes in LS algorithms for WPMS remains challenging, because it is difficult to balance the weight relations between hard and soft clauses, as well as within soft clauses. After all, the initial or adjusted weights may fail to reflect the importance of hard or soft clauses as expressed by their hardness or original weights, potentially causing significant disturbance to the search space. Moreover, while falsifying a hard clause is necessarily an error that should be repaired by increasing its weight to favor its satisfaction subsequently, falsifying a soft clause is not necessarily an error because it can be falsified in an optimal solution. So, it is unclear how to distinguish between satisfied and falsified soft clauses by adjusting their weight.

Anytime solvers can employ exact methods to provide the best real-time solution. Loandra [10] and TT-Open-WBO-Inc [38, 40] (based on Open-WBO-Inc [24]), are two notable instances of such anytime solvers. The complementarity between exact and LS methods has led to another research line that combines these two methods, giving rise to hybrid solvers. Typically, these hybrid solvers apply an SAT solver to obtain a feasible assignment. Then, LS is executed, using this assignment as the initial assignment, until no improvement for k consecutive steps or a short time budget is reached. The best solution found by LS will serve as an initial model and provide an initial upper bound for an SAT-based component. This hybrid-solving process will continue until the total budget is exhausted. For the weighted part of the SAT-based component, the best-performing anytime solver TT-Open-WBO-Inc [40] combines weights approximation by Boolean Multilevel Optimization [24] and the SAT-based LS algorithm Polosat [39]; for its SAT-based unweighted component, it combines Mrs. Beaver algorithm [37] and Polosat.

Hybrid solvers demonstrate impressive performance in the anytime tracks of recent MaxSAT Evaluations (MSEs). Hybrid solvers won 2 out of 4 anytime categories at MSE 2020, 3 out of 4 anytime categories at MSE 2021, and all 4 anytime categories at the two latest evaluations, MSE 2022 and MSE 2023.

4 Proposed Methodology

In this section, we first analyze the questions to answer when designing a clause weighting scheme, then propose a novel clause weighting scheme called Unified Soft Clause Weighting Scheme (Unified-SW) to provide simple and effective responses to these questions, based on which a new LS algorithm for WPMS called *USW-LS* is designed.

4.1 Analysis of clause weighting scheme

Recent LS solvers for WPMS consist of iteratively choosing the next variable x to flip, based on its score $score(x)$. Recall that $score(x) = make(x) - break(x)$, where $make(x)$ ($break(x)$) is the sum of weights of soft and hard clauses that will become satisfied (falsified) upon flipping x. The weight of each clause is initialized at the beginning of the search process and dynamically adjusted later on. Note that no distinction is made between hard and soft clauses when calculating $score(x)$ once their weights are defined. This working scheme raises the following questions to answer when designing an LS solver:

- How to initialize and adjust the weight of each hard clause and each soft clause to distinguish between them during the search? In particular, what is the increment to be added to the weight of a soft clause?

- Can the weight of a clause be infinitely increased when the search proceeds? In other words, should there be an upper bound for the weight of a clause? If yes, what is the upper bound? Should it be uniform for all clauses or specific for each clause?
- When the search encounters a local optimum, how to select the clauses whose weights should be adjusted?

Different responses to these questions yield solvers of different performances. State-of-the-art LS solvers either perform prior extensive experimental analysis or use intuition to answer the first two questions, and the responses are often instance-type specific and are hard to obtain for new instance types. In fact, the difficulty in designing an LS solver is that a very small change in the solver can considerably deteriorate its performance.

Furthermore, existing weighting schemes usually increase the weights of falsified clauses and leave the weights of other clauses unchanged to answer the last question. This is reasonable for a falsified hard clause because a falsified hard clause represents an error that must be repaired. However, the situation is much more complicated for a falsified soft clause, because falsifying a soft clause is not necessarily an error. Moreover, only increasing the weights of falsified soft clauses while leaving the weights of satisfied soft clauses unchanged may change the hierarchy of soft clauses to mislead the solver. In other words, if the original weight of a soft clause c_1 is greater than another soft clause c_2, i.e., if c_1 is originally considered to be more important than c_2, increasing the weight of c_2 but leaving the weight of c_1 unchanged may make the solver satisfy c_2 in priority so that the solver becomes further away from the optimal solution.

In the next subsection, we propose a novel clause weighting scheme that provides simple and effective responses to all the above questions.

4.2 Unified-SW: a novel weighting scheme

We propose the Unified-SW scheme, where Unified-SW stands for unified soft clause weighting. It operates as follows:

Initialization of Clause Weights: At the beginning of each round (restart) of local search, Unified-SW initializes each clause weight as follows:

- For each hard clause c, the weight $w(c) := 1$.
- For each soft clause c, the weight $w(c) := 0$.

Update of Clause Weights: When the search encounters a local optimum α, the clause weights are updated as follows:

- If α falsifies at least one hard clause, and flipping any variable cannot increase the total weight of satisfied hard clauses, α is called an *infeasible local optimum.* In this case, for each falsified hard clause c, $w(c) := w(c) + 1$.
- Otherwise, α is a *feasible local optimum* (i.e., all hard clauses are satisfied, but α cannot be improved by flipping any variable). Let k be the number of feasible local optima encountered so far. For each soft clause c, $w(c) := k \times \frac{w_{ori}(c)}{avgw_{soft}}$, where w_{ori} is the original weight of c in the WPMS instance and $avgw_{soft}$ is the average original weight of all soft clauses.

Note that while the original weights of soft clauses are positive integers, the weights defined by Unified-SW are positive real numbers with double precision.

Unified-SW obtains the following distinguishing properties.

- At the beginning of the search, the soft clauses are not considered in the score of any variables because their weight is 0. Consequently, the solver focuses on searching for the first solution by increasing the weight of the hard clauses, and the first found solution is necessarily a feasible local optimum.

- At each feasible local optimum, the weight of *each* soft clause is increased proportionally to its original weight, whether it is satisfied or not, which significantly differs from the previous approaches in which only the weight of falsified soft clauses is increased. It is for this reason that the new clause weighting scheme is called Unified-SW. The intuition behind this unified soft clause weighting is that even a satisfied soft clause in a feasible local optimum can represent an error that should be repaired.
- The hierarchy between soft clauses is always kept, i.e., given any two soft clauses c_1 and c_2, if $w_{ori}(c_1) \leq w_{ori}(c_2)$, then $w(c_1) \leq w(c_2)$ for any k. Moreover, the difference of weights between c_1 and c_2 is $k \times (\frac{w_{ori}(c_1)}{avgw_{soft}} - \frac{w_{ori}(c_2)}{avgw_{soft}})$ which will become larger and larger when search proceeds. In other words, the more the solver encounters feasible local optima, the more important the soft clauses with great original weight will be, so that it will be considered a priority to improve the cost of a solution.
- Let $S = \{c_1, \ldots, c_s\}$ be the set of all soft clauses. After encountering k^{th} feasible local optima, the total weight of soft clauses is $k \times s$. In other words, the more the solver encounters feasible local optima, the larger the total weight of soft clauses. This is relevant because a feasible local optimum is often dominated by hard clauses, i.e., a feasible local optimum is often encountered by satisfying hard clauses. However, when there are many feasible local optima, soft clauses should be considered more in the score of variables to improve the cost of the solutions.
- The weights of hard and soft clauses are increased upon infeasible and feasible local optima, respectively, and infeasible local optima are considered in priority. The increase of weights of soft clauses upon a feasible local optimum can make the solver far from solutions, and then the solver works with the new greater weights of soft clauses to hopefully move again toward a better solution via new paths, eventually by increasing the weights of hard clauses in infeasible local optima along the new paths.
- There is no upper bound, neither for the weights of soft clauses nor for the weights of hard clauses, which greatly simplifies algorithm design.
- The best solution found so far in terms of weights defined by Unified-SW is also the best solution found so far in terms of original weights. To see this, we look at k and denote the best solution found so far by α^*. When $k = 0$, i.e., at the beginning, the first solution found is obviously the best solution found so far. When $k > 0$, let c denote a falsified soft clause, $w(c) = \sum_c k \times \frac{w_{ori}(c)}{avgw_{soft}}$ which is the smallest if and only if $cost(\alpha^*) = \sum_c w_{ori}(c)$ is the smallest. Therefore, when a solver finds the best solution so far in terms of weights defined by Unified-SW, it does not need to test if it is also the best solution found so far in terms of original weights.

Note that for PMS, which is a particular case of WPMS where both $w_{ori}(c)$ and $avgw_{soft}$ are 1 for any soft clause c, the weight of each soft clause is increased by 1 in each feasible local optimum, increasing the importance of the soft clauses w.r.t. hard clauses and hopefully making the next solution better.

4.3 The *USW-LS* algorithm

Based on the above Unified-SW, we introduce a new LS algorithm named *USW-LS*. The pseudo-code of *USW-LS* is outlined in Algorithm 1. We use α^* and $cost^*$ to denote the best-found solution so far and the *cost* value of the best-found solution, respectively, while α represents the current assignment maintained during the search.

In the *USW-LS* algorithm, α^* is initialized as empty, and $cost^*$ is initialized to positive infinity. It then iteratively executes the local search process until a termination criterion is met (Lines 2–20). A round is from line 3 to line 20.

Algorithm 1 *USW-LS*.

```
Input: WPMS instance $F$, $cutoff$ time.
Output: The best-found solution and its $cost$, or "No solution found".
1  $\alpha^* := \emptyset$; $cost^* := +\infty$;
2  while no terminating criteria are met do
3      $\alpha :=$ an initial complete assignment;
4      Initialize clause weights by Unified-SW;
5      $L = 10\,000\,000$;
6      for $step = 0; step < L; step{+}{+}$ do
7          if $\alpha$ is feasible and $cost^* > cost(\alpha)$ then
8              $\alpha^* := \alpha$; $cost^* := cost(\alpha)$; $L = step + 10^7$;
9              if $cost^* == 0$ then
10                 return $\alpha^*$ and $cost^*$;
11         if $(D : \{x|score(x) > 0\}) \neq \emptyset$ then
12             $v :=$ a variable in $D$ selected by the BMS strategy;
13         else
14             update clause weights by Unified-SW;
15             if $\exists$ falsified hard clauses then
16                 $c :=$ a random falsified hard clause;
17             else
18                 $c :=$ a random falsified soft clause;
19             $v :=$ the variable with highest score in $c$;
20         $\alpha := \alpha$ with $v$ flipped;
21 if $\alpha^* \neq \emptyset$ then return $\alpha^*$ and $cost^*$;
22 else return "No solution found";
```

In the local search process, an initial assignment is generated by a unit propagation based procedure (Line 3) [12]. *USW-LS* then initializes the weights of all clauses by the Unified-SW scheme. After initialization, *USW-LS* conducts the search process (Lines 6–20). During the search process, whenever *USW-LS* finds a solution whose cost value is lower than $cost^*$, α^* and $cost^*$ are updated accordingly.

In each search step, let D denote the set of variables with $score(x){>}0$. *USW-LS* selects a variable and flips its value depending on two situations. (I) If D is not empty, a variable is selected from D. Since traversing all elements in D would be time-consuming, a sampling strategy called BMS (Best from Multiple Selections) [11] is adopted here. Through BMS, t variables are randomly selected from the D set (Where t is a parameter in BMS, the value of which is specified in the Experimental setup.), and then the variable with the highest score among the t variables is chosen to flip. It is shown in [11] that the score of the variable selected in this way is close to the highest in D with high probability. (II) If D is empty, indicating that the search is stuck in a local optimum, *USW-LS* updates the weights of clauses according to Unified-SW. Then *USW-LS* randomly selects a falsified clause c, and picks the variable with the highest score from the selected clause c.

Finally, when any terminating criterion is met, *USW-LS* reports α^* and $cost^*$ if a solution is found; otherwise, it reports "No solution found".

5 Experimental Evaluations

In this section, we introduce experimental preliminaries and then conduct extensive experiments on unweighted and weighted benchmarks from the anytime tracks of MaxSAT Evaluations (MSEs) from 2018 to 2023. First, we compare *USW-LS* with three state-of-the-art LS solvers. Second, we demonstrate the performance enhancement achieved by combining *USW-LS* with an SAT-based solver. Third, we present experimental results to demonstrate the anytime performance exhibited by LS solvers employing different weighting schemes. Finally, we examine the effectiveness of uniformly adjusting soft clause weights.

5.1 Experimental preliminaries

Benchmarks. Our experiments are conducted on 12 benchmarks, i.e., unweighted and weighted benchmarks from the anytime tracks of MSEs from 2018 to 2023.

Competitors. In the first experiment, we compare *USW-LS* with the following three state-of-the-art LS solvers, all of which employ various weighting schemes. The source codes of these three solvers are publicly available,[3,4,5] and the parameter settings follow those presented in their papers.

- NuWLS [18], which uses the Dist-Weighting scheme.
- BandMaxSAT [50], which adopts the Weighting-PMS scheme proposed in SATLike [28]
- SATLike3.0 [12], which adopts the Weighting-PMS scheme.

In the second experiment, we combine *USW-LS* with TT-Open-WBO-Inc (MSE2020 version) [40], which is based on Open-WBO-Inc [24], resulting in a new hybrid solver named *USW-LS-c*.

First, we compare *USW-LS-c* with NuWLS-c, DT-HyWalk, and SATLike-c.

- NuWLS-c combines NuWLS with TT-Open-WBO-Inc (MSE2020 version). We use its source code from MSE 2022.[6]
- DT-HyWalk combines BandMaxSAT with other LS and TT-Open-WBO-Inc (MSE2020 version). We use its source code from MSE 2022.[6]
- SATLike-c combines SATLike with TT-Open-WBO-Inc (MSE2020 version). We use its source code from MSE 2021.[7]

Then, we compare *USW-LS-c* with all hybrid solvers from the anytime track of MSE 2023.[8]

- TT-Open-WBO-Inc-23(G) [43] combines NuWLS with TT-Open-WBO-Inc, where *Glucose4.1* [7] serves as the underlying SAT solver.
- TT-Open-WBO-Inc-23(I) [43] combines NuWLS with TT-Open-WBO-Inc, where *IntelSAT* [42] serves as the underlying SAT solver.
- NuWLS-c-Band [48] combines NuWLS with BandMaxSAT and TT-Open-WBO-Inc.
- NuWLS-c-FPS [48] combines NuWLS with the farsighted probabilistic sampling (FPS) strategy [49] and TT-Open-WBO-Inc.

The purpose of the second experiment is to investigate whether *USW-LS* could improve the performance of hybrid solvers by combining LS with SAT-based algorithms. Note that this experiment does not include NuWLS-c-2023 [17] because NuWLS-c-2023 is identical to *USW-LS-c*, combining *USW-LS* with TT-Open-WBO-Inc.

[3] `https://github.com/filyouzicha/NuWLS`
[4] `https://github.com/JHL-HUST/BandMaxSAT`
[5] `http://lcs.ios.ac.cn/%7ecaisw/MaxSAT.html`
[6] `https://maxsat-evaluations.github.io/2022`
[7] `https://maxsat-evaluations.github.io/2021`
[8] `https://maxsat-evaluations.github.io/2023`

In the third experiment, we analyze the anytime performance of *USW-LS* against NuWLS, BandMaxSAT, and SATLike3.0.

In the fourth experiment, we make an ablation study to compare *USW-LS* with a variant identical to *USW-LS* except that it only increases the weights of the falsified soft clauses in a feasible local optimum.

Experimental setup. Our *USW-LS* solver is implemented in C++. *USW-LS* employs the BMS strategy with a parameter t denoting the sample count, following the settings of NuWLS. i.e., for PMS, t is set to 96; for WPMS, t is set to 25.

All solvers are compiled with g++ using the "-O3" option. The experiments are conducted on a workstation running Ubuntu (version=" 20.04.4 LTS (Focal Fossa)") and equipped with AMD EPYC 7763 3.2GHz CPUs.

Consistent with the rules in the anytime tracks of recent MSEs, we employ two cutoff times, 60 seconds and 300 seconds. Each solver performs one run within a given cutoff time on each instance. We record the cost of the best solution found by solver S on instance I, denoted as $cost_{SI}$. The cost of the best solution found among all solvers in the same table within the same cutoff time on instance I is denoted as $cost_{bI}$. The cost of the best-known solution on instance I is denoted as $best_I$. For each solver S solving a benchmark B within a cutoff time, we use two metrics to evaluate the performance of S.

- $\#win$: the number of instances where the corresponding $cost_{bI}$ can be obtained by solver S on B (i.e., the number of instances on which S wins). The number of winning instances is a metric widely used to evaluate the performance of LS WPMS solvers.
- avg_{score}: we use $score_{SI}$ to denote the competition score of solver S on instance I, if S could not report any solution on instance I within the cutoff time, then $score_{SI} = 0$. Otherwise, $score_{SI} = \frac{best_I+1}{cost_{SI}+1}$. We use avg_{score} to denote the average competition score of a solver on a benchmark. The competition score is the metric to measure the performance of anytime solvers in recent MSEs.

The number of instances in each benchmark is indicated by '#inst'. For each of the above two metrics, if a solver obtains a larger metric value on a benchmark, then the solver exhibits better performance on the benchmark. The results highlighted in **bold** indicate the best performance for the corresponding metric.

5.2 Comparison with local search solvers

The comparative results of *USW-LS* and its LS competitors on all the benchmarks are shown in Table 1. On all the unweighted and weighted benchmarks, for both 60-second and 300-second cutoff times, *USW-LS* outperforms all competing solvers in terms of the number of winning instances ($\#win$) and average scores (avg_{score}).

- For unweighted benchmarks with a 60-second cutoff time, *USW-LS* outperforms the second-ranked solver by 28.40-58.57% for $\#win$ and 1.54-6.51% for avg_{score}. With a 300-second cutoff time, *USW-LS* outperforms the second-ranked solver by 31.48- 63.64% for $\#win$ and 1.33-8.19% for avg_{score}.
- For weighted benchmarks with a 60-second cutoff time, *USW-LS* exceeds the second-ranked solver by 45.07-157.58% for $\#win$, and 1.34-7.74% for avg_{score}. With a 300-second cutoff time, *USW-LS* exceeds the second-ranked solver by 65.29-127.08% for $\#win$ and 0.98-6.97% for avg_{score}.

Among these LS solvers, the only distinction between *USW-LS*, NuWLS, and SATLike3.0 lies in the weighting scheme utilized. Both BandMaxSAT and SATLike3.0 employ the same weighting scheme, differing only in the strategy introduced by BandMaxSAT for variable

Table 1 Comparisons of *USW-LS* with state-of-the-art LS solvers.

Benchmark	#inst	*USW-LS*		NuWLS		BandMaxSAT		SATLike3.0	
		$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}
				Unweighted (60 seconds)					
Unw_18	153	**102**	**0.7426**	69	0.7131	42	0.6455	45	0.5673
Unw_19	299	**206**	**0.7297**	151	0.7117	96	0.6575	106	0.6179
Unw_20	262	**181**	**0.7360**	127	0.7161	76	0.6679	77	0.6150
Unw_21	155	**104**	**0.6578**	81	0.6176	52	0.5778	43	0.5223
Unw_22	179	**113**	**0.7092**	78	0.6684	55	0.6287	37	0.5682
Unw_23	179	**111**	**0.6629**	70	0.6528	30	0.5738	21	0.5160
				Unweighted (300 seconds)					
Unw_18	153	**109**	**0.7726**	71	0.7479	54	0.6906	46	0.6113
Unw_19	299	**213**	**0.7546**	162	0.7447	116	0.6961	116	0.6506
Unw_20	262	**187**	**0.7616**	132	0.7435	95	0.6965	90	0.6430
Unw_21	155	**111**	**0.6877**	79	0.6419	59	0.6013	46	0.5495
Unw_22	179	**126**	**0.7514**	77	0.6945	65	0.6540	44	0.5955
Unw_23	179	**111**	**0.6977**	71	0.6881	39	0.6219	34	0.5721
				Weighted (60 seconds)					
Wei_18	172	**103**	**0.7716**	71	0.7614	24	0.6978	18	0.6809
Wei_19	297	**192**	**0.7686**	125	0.7419	65	0.6638	41	0.6581
Wei_20	253	**162**	**0.7797**	94	0.7436	43	0.6301	36	0.6391
Wei_21	151	**75**	**0.6833**	38	0.6343	24	0.5437	27	0.5740
Wei_22	197	**105**	**0.7094**	58	0.6751	35	0.6052	25	0.6095
Wei_23	160	**85**	**0.6631**	33	0.6182	31	0.5651	15	0.5671
				Weighted (300 seconds)					
Wei_18	172	**111**	**0.7848**	67	0.7772	27	0.7244	16	0.7042
Wei_19	297	**200**	**0.7916**	121	0.7715	81	0.7118	54	0.7034
Wei_20	253	**170**	**0.8085**	99	0.7784	54	0.6893	39	0.6815
Wei_21	151	**74**	**0.7199**	35	0.6817	37	0.6225	28	0.6299
Wei_22	197	**109**	**0.7459**	57	0.7124	48	0.6705	28	0.6629
Wei_23	160	**83**	**0.7093**	37	0.6630	34	0.6338	16	0.6293

selection after encountering local optima. Experimental results demonstrate that *USW-LS* outperforms NuWLS, while NuWLS outperforms SATLike3.0 in terms of $\#win$ and avg_{score}, indicating that a well-designed weighting scheme can significantly enhance the performance of LS solvers.

5.3 Improving hybrid solver through *USW-LS*

Since our experimental results in Table 1 demonstrate that *USW-LS* performs much better than state-of-the-art LS solvers for solving WPMS on all benchmarks, we are interested in investigating whether *USW-LS* could improve the performance of hybrid solvers compared to these LS solvers. Therefore, we compare *USW-LS-c* with NuWLS-c, DT-HyWalk, and SATLike-c, and the experimental results are shown in Table 2. From Table 2, in terms of $\#win$ and avg_{score}, our *USW-LS-c* solver outperforms all the competitors on all benchmarks and runtimes, indicating that *USW-LS* can considerably advance the performance of hybrid solvers.

We submitted *USW-LS-c* to the anytime track of MSE 2023, where *USW-LS-c* is referred to as NuWLS-c-2023. The official results from the anytime track of MSE 2023 indicate that the top-five solvers are all hybrid, and USW-LS-c *won all four categories*.

Table 2 Comparisons of *USW-LS-c* with hybrid solvers NuWLS-c, DT-HyWalk, and SATLike-c.

Benchmark	#inst	*USW-LS-c*		NuWLS-c		DT-HyWalk		SATLike-c	
		$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}
`Unweighted` (60 seconds)									
`Unw_18`	153	**106**	**0.8239**	74	0.8015	83	0.7960	52	0.7745
`Unw_19`	299	**234**	**0.8717**	175	0.8549	189	0.8349	158	0.8265
`Unw_20`	262	**202**	**0.8606**	138	0.8420	151	0.8257	122	0.8214
`Unw_21`	155	**115**	**0.8280**	86	0.8011	87	0.7870	80	0.7834
`Unw_22`	179	**128**	**0.8239**	75	0.7896	89	0.7776	72	0.7610
`Unw_23`	179	**109**	**0.7832**	73	0.7605	75	0.7268	66	0.7220
`Unweighted` (300 seconds)									
`Unw_18`	153	**98**	**0.8635**	77	0.8501	87	0.8507	52	0.8249
`Unw_19`	299	**228**	**0.9141**	186	0.9031	204	0.8950	160	0.8806
`Unw_20`	262	**190**	**0.9025**	144	0.8888	161	0.8754	121	0.8635
`Unw_21`	155	**116**	**0.8864**	96	0.8752	97	0.8646	80	0.8502
`Unw_22`	179	**125**	**0.8866**	90	0.8773	102	0.8601	83	0.8482
`Unw_23`	179	**118**	**0.8609**	88	0.8490	94	0.8274	84	0.8202
`Weighted` (60 seconds)									
`Wei_18`	172	**101**	**0.8811**	84	0.8772	69	0.8706	76	0.8699
`Wei_19`	297	**185**	**0.8572**	133	0.8411	122	0.8233	123	0.8352
`Wei_20`	253	**144**	**0.8518**	110	0.8386	88	0.8006	92	0.8061
`Wei_21`	151	**75**	**0.7800**	59	0.7481	44	0.7102	46	0.7090
`Wei_22`	197	**105**	**0.7862**	66	0.7617	53	0.7325	60	0.7390
`Wei_23`	160	**89**	**0.7867**	54	0.7790	34	0.7296	44	0.7458
`Weighted` (300 seconds)									
`Wei_18`	172	**108**	**0.9147**	89	0.8985	82	0.9007	87	0.9003
`Wei_19`	297	**197**	**0.9202**	164	0.9054	153	0.9021	137	0.8962
`Wei_20`	253	**163**	**0.9178**	129	0.8949	115	0.8653	106	0.8568
`Wei_21`	151	**83**	**0.8475**	65	0.8317	53	0.7798	52	0.7516
`Wei_22`	197	**116**	**0.8676**	90	0.8516	68	0.7838	62	0.7760
`Wei_23`	160	**96**	**0.8891**	63	0.8703	45	0.8384	50	0.8313

To give a more global assessment of *USW-LS-c* against state-of-the-art anytime solvers, we compare it with the four other top-five solvers NuWLS-c-Band, NuWLS-c-FPS, TT-Open-WBO-Inc-23(G), and TT-Open-WBO-Inc-23(I) in the anytime track of MSE 2023 on all benchmarks from the anytime tracks of MSE since 2018. The experimental results are reported in Table 3. In terms of $\#win$, *USW-LS-c* outperforms all the competitors on all comparisons. In terms of avg_{score}, *USW-LS-c* outperforms all the competing solvers on 18 out of 24 comparisons. On the other 6 comparisons, *USW-LS-c* ranks second and slightly worse than the best.

5.4 Anytime performance analysis

Apart from the fixed-budget performance measures that have been commonly applied for MSEs and previous work on WPMS, an anytime performance measure [47], which evaluates the performance of solvers across multiple cutoff times, has been used for assessing four WPMS LS solvers. We also take this measure into account for comparing our proposed *USW-LS* with three LS solvers.

Table 3 Comparisons of *USW-LS-c* with hybrid solvers from MSE 2023, TT-OWI-G refers to TT-Open-WBO-Inc-23(G), TT-OWI-I refers to TT-Open-WBO-Inc-23(I).

Benchmark	#inst	*USW-LS-c*		TT-OWI-G		TT-OWI-I		NuWLS-c-Band		NuWLS-c-FPS	
		$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}
`Unweighted` (60 seconds)											
`Unw_18`	153	**85**	**0.8239**	63	0.8097	58	0.8021	70	0.8092	77	0.8101
`Unw_19`	299	**200**	**0.8717**	159	0.8609	160	0.8547	170	0.8639	176	0.8623
`Unw_20`	262	**171**	**0.8606**	117	0.8445	119	0.8402	128	0.8458	139	0.8382
`Unw_21`	155	**92**	**0.8280**	77	0.8086	77	0.8093	77	0.8050	81	0.7930
`Unw_22`	179	**96**	**0.8239**	64	0.8099	63	0.8021	77	0.7921	74	0.7851
`Unw_23`	179	**89**	**0.7832**	61	0.7829	61	0.7615	66	0.7760	66	0.7770
`Unweighted` (300 seconds)											
`Unw_18`	153	**91**	**0.8635**	64	0.8519	77	0.8610	77	0.8556	77	0.8551
`Unw_19`	299	**214**	**0.9141**	163	0.9054	185	0.9108	183	0.9084	187	0.9099
`Unw_20`	262	**180**	0.9025	126	0.8913	140	**0.9035**	141	0.8891	146	0.8862
`Unw_21`	155	**105**	0.8864	86	0.8751	81	**0.8923**	84	0.8549	86	0.8508
`Unw_22`	179	**108**	**0.8866**	78	0.8779	79	0.8861	91	0.8514	88	0.8427
`Unw_23`	179	**112**	**0.8609**	81	0.8563	84	0.8568	87	0.8543	87	0.8443
`Weighted` (60 seconds)											
`Wei_18`	172	**98**	0.8811	84	**0.8816**	65	0.8713	84	0.8761	86	0.8801
`Wei_19`	297	**181**	**0.8572**	123	0.8318	122	0.8393	127	0.8406	103	0.8294
`Wei_20`	253	**140**	**0.8518**	96	0.8335	93	0.8360	95	0.8336	82	0.8291
`Wei_21`	151	**69**	**0.7800**	46	0.7502	42	0.7580	38	0.7526	44	0.7492
`Wei_22`	197	**93**	**0.7862**	67	0.7591	53	0.7542	67	0.7628	51	0.7558
`Wei_23`	160	**85**	**0.7867**	41	0.7585	35	0.7695	45	0.7755	45	0.7727
`Weighted` (300 seconds)											
`Wei_18`	172	**101**	0.9147	95	**0.9176**	71	0.9091	92	0.9062	96	0.9067
`Wei_19`	297	**186**	**0.9202**	149	0.8823	136	0.9109	144	0.8990	135	0.9000
`Wei_20`	253	**153**	**0.9178**	119	0.8926	103	0.8884	107	0.8926	107	0.8944
`Wei_21`	151	**71**	0.8475	64	**0.8550**	49	0.8459	52	0.8342	55	0.8477
`Wei_22`	197	**97**	0.8676	91	**0.8678**	68	0.8356	72	0.8350	68	0.8383
`Wei_23`	160	**90**	**0.8891**	55	0.8466	42	0.8889	49	0.8569	49	0.8686

The empirical cumulative distribution function (ECDF) is used to assess the anytime performance of solvers. Given a set of target values, i.e., cost, for each WPMS instance, an ECDF value at time t represents the proportion of targets that have been achieved by a solver within cutoff time t. Following the suggestion in the previous work [47], we consider a set of 100 cost values that are sampled within the smallest and largest cost values obtained during the optimization process of all tested solvers. We plot in Figure 1 the averaged ECDFs values across all tested WPMS instances for 100 cutoff times t that are selected within the range of $[0, 300s]$ with a logarithmic scale. We can observe that *USW-LS* generally outperforms the other tested LS solvers across all the considered cutoff times, and this advantage is significant, as shown in the figure. Note that the results are aggregated across all tested WPMS instances such that we do not observe sharp increases at a specific cutoff time or a flat line in Figure 1, but these behaviors may occur in particular instances as shown in previous work [47].

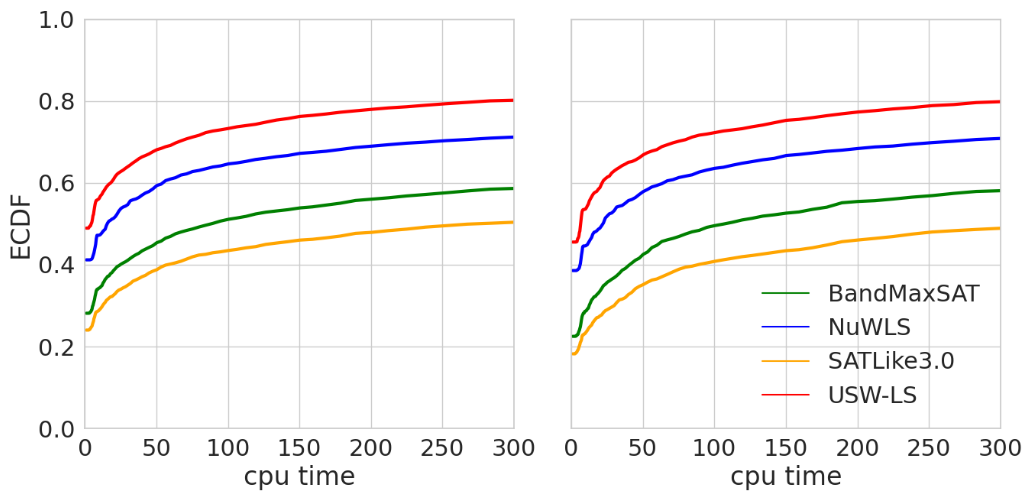

Figure 1 ECDFs of the tested solvers for two instance sets: **Left:** Unweighted, **Right:** Weighted.

5.5 Effects of uniformly adjusting soft clause weights

To examine the effect of uniformly adjusting soft clause weights, we introduce an alternative version of *USW-LS*, denoted as USW-LS-alt. The only difference between USW-LS-alt and *USW-LS* lies in their approach to adjusting soft clause weights: in USW-LS-alt, if the assignment is a feasible local optimum, only the weights of falsified soft clauses are increased by setting $k = k + 1$, and for each falsified soft clause c, $w(c) := k \times \frac{w_{ori}(c)}{avgw_{soft}}$.

Table 4 presents the results of *USW-LS* and USW-LS-alt across all the benchmarks. From table 4, *USW-LS* demonstrates superior performance over USW-LS-alt in terms of both $\#win$. and avg_{score}. On unweighted benchmarks, *USW-LS* outperforms USW-LS-alt by 45.21-71.17% in terms of $\#win$, and exhibits a 4.18-6.23% improvement over USW-LS-alt for avg_{score}. For weighted benchmarks, *USW-LS* surpasses USW-LS-alt more significantly, with 116.67-305.77% higher $\#win$ and 2.18-9.91% better avg_{score} performance.

Table 4 Comparisons of *USW-LS* with its alternative version USW-LS-alt.

Benchmark	#inst	*USW-LS*		USW-LS-alt		*USW-LS*		USW-LS-alt	
		$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}	$\#win$	avg_{score}
		60 seconds				300 seconds			
`Unw_18`	153	**106**	**0.7426**	73	0.6991	**113**	**0.7726**	75	0.7311
`Unw_19`	299	**217**	**0.7297**	142	0.6899	**226**	**0.7546**	147	0.7160
`Unw_20`	262	**190**	**0.7360**	111	0.6958	**197**	**0.7616**	120	0.7223
`Unw_21`	155	**113**	**0.6578**	77	0.6304	**119**	**0.6877**	81	0.6601
`Unw_22`	179	**133**	**0.7092**	83	0.6685	**140**	**0.7514**	88	0.7099
`Unw_23`	179	**116**	**0.6629**	75	0.6326	**121**	**0.6977**	76	0.6648
`Wei_18`	172	**133**	**0.7716**	58	0.7486	**134**	**0.7848**	58	0.7681
`Wei_19`	297	**237**	**0.7686**	72	0.7075	**244**	**0.7916**	67	0.7401
`Wei_20`	253	**201**	**0.7797**	57	0.7094	**211**	**0.8085**	52	0.7409
`Wei_21`	151	**110**	**0.6833**	44	0.6339	**114**	**0.7199**	45	0.6728
`Wei_22`	197	**142**	**0.7094**	43	0.6630	**152**	**0.7459**	39	0.6978
`Wei_23`	160	**117**	**0.6631**	54	0.6260	**119**	**0.7093**	53	0.6697

6 Conclusions and Future Work

Most solvers evaluated in the anytime tracks of recent MSEs integrate local search techniques, showing wide interest in this approach for solving MaxSAT. The main difficulty of local search is that it frequently falls into local optima, and the main technique to escape from a local optimum is clause weighting. Unfortunately, designing an effective clause weighting scheme needs to properly answer many questions, because otherwise, the solver can easily lose control of the search trajectory. In this paper, we proposed a novel clause weighting scheme called Unified-SW that provides simple, clever, and effective answers to these questions.

First, Unified-SW distinguishes between feasible local optima and infeasible local optima, to only increase the weights of hard (soft) clauses in infeasible (feasible) local optima. Second, in a feasible local optimum, instead of only increasing the weights of the falsified soft clauses as in some previous approaches, Unified-SW increases the weights of all soft clauses, whether they are satisfied or not, considering that even the satisfaction of a soft clause can represent an error that should be repaired. Third, the total increase of the weights of soft clauses is a constant in a feasible local optimum, proportionally split among all soft clauses, so that the hierarchy among the soft clauses is kept, and no upper bound is needed for their weights. Using Unified-SW, the increase of weights of soft clauses in a feasible local optimum can make the solver far from solutions, the solver then works with the new greater weights of soft clauses to hopefully move again toward a better solution via new paths, eventually by increasing the weights of hard clauses in the infeasible local optima along the paths.

We implemented Unified-SW in a new LS solver called *USW-LS*, and designed four experiments on all benchmarks from the anytime tracks of MSEs since 2018 to evaluate the effectiveness of Unified-SW, using three measures: number of winning instances, competition score used in MSEs, and anytime performance. These experiments show: (1) *USW-LS* performs much better than three state-of-the-art LS solvers NuWLS, BandMaxSAT and SATLike3.0; (2) when combined with an SAT-based solver, *USW-LS-c* also performs much better than NuWLS, BandMaxSAT and SATLike3.0 combined with the SAT-based solver; (3) Being the winning solver in all the 4 categories of the anytime tracks of MSE 2023, *USW-LS-c* also performs better than the other four top-five solvers of the anytime track of MSE 2023 on most benchmarks from the anytime tracks of MSEs since 2018; (4) Across multiple cutoff times, *USW-LS* performs also better than NuWLS, BandMaxSAT and SATLike3.0; (5) Uniformly increasing the weights of *all* soft clauses in a feasible local optimum is important, because *USW-LS* is much better than the variant that only increases the weights of falsified soft clauses.

In the future, we plan to study more deeply the landscape of the search space changed by Unified-SW for various types of instances. Currently, only falsified hard clauses see their weight increased in an infeasible local optimum, we will investigate the possibility of also increasing the weights of all soft clauses if the cost of the current assignment cannot be improved. As is said above, the total increase of weights of soft clauses in a feasible local optimum is a constant. We will investigate the possibility of increasing this constant when the search proceeds and the number of encountered feasible local optima exceeds a threshold. Finally, we also plan to apply Unified-SW to other optimization problems.

References

1 André Abramé and Djamal Habet. Ahmaxsat: Description and evaluation of a branch and bound max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1):89–128, 2014.

2 David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

3 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proceedings of SAT 2009*, pages 427–440. Springer, 2009.

4 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *Proceedings of AAAI 2010*, pages 3–8, 2010.

5 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

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

7 Gilles Audemard and Laurent Simon. On the glucose sat solver. *International Journal on Artificial Intelligence Tools*, 27(01):1840001, 2018.

8 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiabiliy. In *Handbook of satisfiability*, pages 929–991. IOS Press, 2021.

9 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided maxsat solving. In *International Conference on Automated Deduction*, pages 1–22. Springer, 2023.

10 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *Proceedings of CPAIOR 2019*, pages 39–56, 2019.

11 Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of IJCAI 2015*, pages 747–753, 2015.

12 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

13 Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.

14 Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial maxsat. In *Proceedings of AAAI 2014*, pages 2623–2629, 2014.

15 Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of AAAI 1997*, pages 263–268, 1997.

16 Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of max-sat resolution through up-resilience. *Artificial Intelligence*, 289:103397, 2020.

17 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls-c-2023: Solver description. *MaxSAT Evaluation 2023*, pages 23–24, 2023.

18 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In *Proceedings of AAAI 2023*, volume 37, pages 3915–3923, 2023.

19 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of CP 2011*, pages 225–239, 2011.

20 Emir Demirović, Nysret Musliu, and Felix Winter. Modeling and solving staff scheduling with partial weighted maxsat. *Annals of Operations Research*, 275:79–99, 2019.

21 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of SAT 2006*, pages 252–265, 2006.

22 Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

23 Yuejun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat. In *1st International Joint Workshop on Artificial Intelligence and Operations Research*, volume 20. Citeseer, 1995.

24 Saurabh Joshi, Prateek Kumar, Sukrut Rao, and Ruben Martins. Open-WBO-Inc: Approximation strategies for incomplete weighted MaxSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):73–97, 2019.

25 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1-2):95–100, 2012.

26 Adrian Kügel. Improved exact solver for the weighted MAX-SAT problem. In *POS-10. In Proceedings of Workshop Pragmatics of SAT,, Edinburgh, UK, 2010*, volume 8, pages 15–27, 2010.

27 Javier Larrosa, Federico Heras, and Simon De Givry. A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

28 Zhendong Lei and Shaowei Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In *Proceedings of IJCAI 2018*, pages 1346–1352, 2018.

29 Zhendong Lei, Shaowei Cai, Fei Geng, Dongxu Wang, Yongrong Peng, Dongdong Wan, Yiping Deng, and Pinyan Lu. Satlike-c: Solver description. *MaxSAT Evaluation*, 2021:19–20, 2021.

30 Chu Min Li and Felip Manyà. Maxsat, hard and soft constraints. In *Handbook of satisfiability*, pages 903–927. IOS Press, 2021.

31 Chu Min Li, Felip Manya, and Jordi Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

32 Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proceedings of AAAI 2010*, pages 128–133, 2010. `doi:10.1609/AAAI.V24I1.7536`.

33 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In *Proceedings of CP 2021*, pages 38:1–38:18, 2021. `doi:10.4230/LIPICS.CP.2021.38`.

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

35 Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In *Proceedings of CP 2014*, pages 531–548, 2014.

36 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular maxsat solver. In *Proceedings of SAT 2014*, pages 438–445, 2014.

37 Alexander Nadel. Solving maxsat with bit-vector optimization. In *Proceedings of SAT 2018*, pages 54–72. Springer, 2018.

38 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *Proceedings of FMCAD 2019*, pages 193–202, 2019.

39 Alexander Nadel. On optimizing a generic function in SAT. In *Proceedings of FMCAD 2020*, pages 205–213, 2020.

40 Alexander Nadel. Polarity and variable selection heuristics for sat-based anytime maxsat. *Journal on Satisfiability, Boolean Modeling and Computation*, 12(1):17–22, 2020.

41 Alexander Nadel. TT-Open-WBO-Inc-20: an anytime MaxSAT solver entering MSE'20. In *Proceedings of MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 21–22, 2020.

42 Alexander Nadel. Introducing intel (r) sat solver. In *Proceedings of SAT 2022*, pages 8:1–8:23, 2022.

43 Alexander Nadel. Tt-open-wbo-inc-23: an anytime maxsat solver entering mse'23. *MaxSAT Evaluation 2023*, page 29, 2023.

44 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of AAAI 2014*, pages 2717–2723, 2014.

45 John Thornton, Stuart Bain, Abdul Sattar, and Duc Nghia Pham. A two level local search for MAX-SAT problems with hard and soft constraints. In *Proceedings of AI 2002*, pages 603–614, 2002.

46 John Thornton and Abdul Sattar. Dynamic constraint weighting for over-constrained problems. In *Proceedings of PRICAI 1998*, pages 377–388, 1998.

47 Furong Ye, Chuan Luo, and Shaowei Cai. Better understandings and configurations in maxsat local search solvers via anytime performance analysis. *arXiv preprint*, 2024. `arXiv:2403.06568`.

48 Jiongzhi Zheng, Kun He, Mingming Jin, Zhuo Chen, and Jinghui Xue. Combining bandmaxsat and fps with nuwls-c. *MaxSAT Evaluation 2023*, pages 25–26, 2023.
49 Jiongzhi Zheng, Kun He, and Jianrong Zhou. Farsighted Probabilistic Sampling based local search for (weighted) partial maxsat. In *Proceedings of AAAI 2023*, pages 4132–4139, 2023.
50 Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li, and Felip Manya. BandMaxSAT: A local search maxsat solver with multi-armed bandit. In *Proceedings of IJCAI 2022*, pages 1901–1907, 2022.