Weight-Aware Branch-and-Bound for Weighted Maximum Satisfiability

Jialu Zhang[1] , **Chu-Min Li**[1,2,*] , **Sami Cherif**[1] and **Shuolin Li**[2,*]
[1]MIS UR 4290, Université de Picardie Jules Verne, Amiens, France
[2]LIS, Aix Marseille Université, Marseille, France
{jialu.zhang, chu-min.li, sami.cherif}@u-picardie.fr, shuolin.li@lis-lab.fr

Abstract

The Weighted Partial MaxSAT (WPMS) problem requires finding an assignment that satisfies all hard clauses while maximizing the total weight of satisfied soft clauses. From a computational perspective, WPMS is not merely an extension of the uniform-weight Partial MaxSAT (PMS); rather, it requires search strategies that actively exploit weight information to navigate the solution space. While state-of-the-art heuristic and SAT-based solvers have successfully integrated weight-aware mechanisms, Branch-and-Bound (BnB) solvers, notably WMaxCDCL, largely treat weight information passively during search and pruning. In this paper, we bridge this gap by introducing two novel weight-aware strategies into the BnB framework. Our methods integrate weight information directly into the preprocessing and lower-bound estimation. Experimental results demonstrate that these strategies significantly enhance the performance of WMaxCDCL, establishing a new state-of-the-art for exact WPMS solving.

1 Introduction

Maximum Satisfiability (MaxSAT) is an optimization extension of the Satisfiability problem (SAT) [Bacchus *et al.*, 2021], alongside other variants such as Minimum Satisfiability (MinSAT) [Li *et al.*, 2010]. Generally, MaxSAT encompasses three main variants: Partial MaxSAT (PMS), which partitions clauses into hard and soft ones; Weighted MaxSAT, where each clause has a positive integer weight representing the cost for its violation; and Weighted Partial MaxSAT (WPMS), which combines hard clauses with weighted soft clauses. Thanks to the continuous efforts of the research community, MaxSAT has been successfully applied to solve a wide range of practical combinatorial optimization problems, such as scheduling [Demirović *et al.*, 2019; Cherif *et al.*, 2024], maximum clique [Li and Quan, 2010], FPGA routing [Fu and Malik, 2006], and protein design [Allouche *et al.*, 2014].

*Corresponding authors.

Algorithms for the MaxSAT problem are generally categorized into exact (complete) and heuristic (incomplete) algorithms. Exact algorithms guarantee optimal solutions and certify optimality. Conversely, heuristic algorithms, while often competitive in finding high-quality solutions, do not provide optimality guarantees. This paper focuses on exact algorithms for WPMS, which are primarily categorized into SAT-based and Branch-and-Bound (BnB) algorithms, although it can also be exactly solved using a tableau calculus [Li *et al.*, 2016].

SAT-based algorithms reduce the MaxSAT problem to a sequence of SAT decision problems and solve them with a SAT oracle. The representative solvers include SAT4J-Maxsat [Berre and Parrain, 2010], QMaxSAT [Koshimura *et al.*, 2012], Open-WBO [Martins *et al.*, 2014], Pacose [Paxian and Becker, 2024], MaxHS [Bacchus, 2022], RC2 [Ignatiev *et al.*, 2019], EvalMaxSAT [Avellaneda, 2024], CGSS [Berg *et al.*, 2023] and UwrMaxsat [Piotrów, 2020]. SAT-based solvers demonstrate superior performance on industrial WPMS benchmarks, driven by the core-guided algorithm [Fu and Malik, 2006] and its subsequent refinements such as the stratification strategy [Ansótegui *et al.*, 2009; Ansótegui *et al.*, 2010]. More recently, [Pan *et al.*, 2025] proposed an extended stratified strategy implemented in the CASHWMaxSAT solver, enabling it to win the first place in the weighted track of the MaxSAT Evaluation (MSE) 2024.

BnB algorithms maintain a global upper bound (UB), representing the cost of the best complete assignment found so far, and a lower bound (LB) of the cost for the current partial assignment. The search space is pruned whenever $\mathrm{LB} \geq \mathrm{UB}$. Early BnB solvers, such as MaxSatz [Li *et al.*, 2007], MiniMaxSat [Heras *et al.*, 2008], Akmaxsat [Kuegel, 2010], and Ahmaxsat [Abramé and Habet, 2014; Cherif *et al.*, 2020], demonstrated strong performance on random benchmarks but struggled with industrial instances. To address this, the MaxCDCL algorithm [Li *et al.*, 2021; Li *et al.*, 2022] leverages conflict-driven clause learning (CDCL) to effectively capture the structure of industrial problems. Furthermore, its unlocking mechanism enhances the accuracy of LB estimation [Li *et al.*, 2025]. Yet, despite these advancements, its weighted variant, WMaxCDCL [Coll *et al.*, 2025], remains less competitive than state-of-the-art SAT-based solvers when solving WPMS instances.

While leading heuristic solvers and SAT-based solvers ef-

fectively leverage weight information (e.g., NuWLS [Chu *et al.*, 2023] by considering soft clause weights in the variable scoring schema, and CASHWMaxSAT by stratification), modern BnB solvers have not yet fully exploited this potential. In this paper, we enhance the state-of-the-art BnB solver WMaxCDCL by integrating two novel weight-aware strategies. The first strategy employs a stratification strategy in the preprocessing phase, which significantly simplifies WPMS instances exhibiting hierarchical weight structures. The second strategy optimizes the lookahead framework of WMaxCDCL, resulting in a substantially better lower-bound estimation for dispersed WPMS instances. Experimental evaluations demonstrate that these strategies significantly improve the performance of WMaxCDCL, establishing a new state-of-the-art in exact WPMS solving.

The rest of the paper is organized as follows. Section 2 presents preliminary background. Section 3 details our proposed weight-aware strategies for MaxSAT BnB. Experimental results are presented in Section 4. Finally, we conclude and discuss future work in Section 5.

2 Preliminaries

2.1 Maximum Satisfiability

Let X be a set of Boolean variables. A literal l is either a variable $x \in X$ or its negation $\neg x$. A clause c is a disjunction of literals, represented as a set of literals. A formula F in Conjunctive Normal Form (CNF) is a conjunction of clauses, also represented as a set of clauses. An assignment α maps variables to values in $\{1, 0\}$. It is complete if every variable is assigned a value. Otherwise, it is partial. A literal x (resp. $\neg x$) is satisfied if variable x is assigned 1 (resp. 0). It is falsified if x is assigned 0 (resp. 1). It is unassigned if x is unassigned. We represent α as the set of satisfied literals.

A clause c is satisfied if at least one of its literals is satisfied. An empty clause does not contain any literals and cannot be satisfied. A unit clause contains only one literal. Any clause can become unit by falsifying all its literals but one. Unit propagation (UP) is a procedure that iteratively propagates the only unassigned literal l of a unit clause by satisfying l and all clauses containing l, and removing $\neg l$ from all other clauses, until all clauses are satisfied or an empty clause is produced, which is falsified. A formula F is satisfied if all its clauses are satisfied. The SAT problem thus consists in finding an assignment that satisfies a given CNF formula F.

As an optimization extension of SAT, Maximum Satisfiability (MaxSAT) aims to find a solution that satisfies the maximum number of clauses, making it significantly more challenging than standard SAT [Li and Manyà, 2021]. For the Weighted Partial MaxSAT (WPMS) problem, there are two sets of clauses: H, the set of hard clauses; and S, the set of soft clauses. Each soft clause c is associated with a positive integer $w(c)$, called the weight of c, representing the cost of falsifying c. The cost of a complete assignment satisfying all hard clauses is the sum of the costs of the falsified soft clauses. The goal of WPMS is to find an assignment that satisfies all hard clauses while minimizing the total weight (or cost) of falsified soft clauses (or maximizing the weight of satisfied ones). When all soft clauses are assigned a uniform weight, WPMS reduces to Partial MaxSAT (PMS).

2.2 Boolean Multilevel Optimization

WPMS instances can be divided into three categories: crafted, random, and industrial. Crafted and random instances are primarily synthetic, typically exhibiting uniform or normal weight distributions. In contrast, industrial instances encode real-world problems and follow a characteristic power-law distribution [Ansótegui *et al.*, 2012], where a small number of soft clauses have extremely high weights. Furthermore, some industrial instances may display clear stratification, formalized as Boolean Multilevel Optimization (BMO) in [Marques-Silva *et al.*, 2011]. Equation (1) defines a BMO instance, where m denotes the number of distinct weights in the instance, w_j represents the j^{th} weight in the decreasing weight sequence, and $|C_i|$ means the number of clauses with weight w_i. This definition guarantees that w_j exceeds the sum of weights of all clauses with a lower weight, forcing higher-weight clauses to be satisfied first.

$$w_j > \sum_{j+1 \le i \le m} w_i \cdot |C_i| \qquad j = 1, \ldots, m-1 \qquad (1)$$

2.3 Lookahead Framework in WMaxCDCL

WMaxCDCL removes each soft clause c, after introducing a fresh soft literal l with weight $w(l) = w(c)$, and forcing the equivalence $l \leftrightarrow c$ by inserting a set of hard clauses into H. The computation of a lower bound is based on detecting disjoint *local cores*, where a local core is a subset of soft literals that cannot all be satisfied without falsifying a hard clause under the current partial assignment α. Each soft literal l in a core κ has a weight locked in the core, denoted as $wlock(l, \kappa)$. By definition, $wlock(l, \kappa) = 0$ if $l \notin \kappa$. Each core κ is associated with a weight $w(\kappa)$, such that $\sum_{\neg l \in ext(\alpha)} wlock(l, \kappa) \geq w(\kappa)$ for any complete extension $ext(\alpha)$ of α. Let K be the set of all cores. The lower bound LB under α is equal to $\sum_{\kappa \in K} w(\kappa)$, subject to the condition $\sum_{\kappa \in K} wlock(l, \kappa) \leq w(l)$ for any l. We use $rw(l) = w(l) - \sum_{\kappa \in K} wlock(l, \kappa)$ to denote the remaining weight of l not locked in any core. Note that the condition $\sum_{\kappa \in K} wlock(l, \kappa) \leq w(l)$ is required to ensure the soundness of LB.

If LB $\geq$ UB, where UB is the cost of the best solution found so far, the current branch can safely be pruned. In earlier versions of WMaxCDCL, only soft literals l such that $rw(l) > 0$ can be used to detect a new core. When a core $\kappa = \{l_1, \ldots, l_k\}$ is detected, $w(\kappa) = min_{l \in \kappa} rw(l)$, $wlock(l_i, \kappa) = w(\kappa)$ and $rw(l_i)$ is updated to $rw(l_i) - w(\kappa)$ for $1 \leq i \leq k$. In this way, the condition $\sum_{\kappa \in K} wlock(l, \kappa) \leq w(l)$ is naturally ensured [Coll *et al.*, 2025]. However, in this case, we necessarily have $w(\kappa) = wlock(l, \kappa)$ for any core κ and any $l \in \kappa$, limiting the capacity of WMaxCDCL to compute a tight lower bound.

In the last version of WMaxCDCL [Li *et al.*, 2025], an unlocking mechanism is introduced, allowing the reuse of the weight of literals locked in previous cores to detect a new core under some conditions. Consequently, a core κ with $w(\kappa) > wlock(l, \kappa)$ for some $l \in \kappa$ can be detected. Example 1 shows

Algorithm 1: Lookahead in WMaxCDCL

Input: H: hard clauses, S: soft literals, α: partial assignment, UB: upper bound.
Output: LB: estimated lower bound, K: local cores.

1 LB $\leftarrow cost(\alpha)$;
2 $\alpha' \leftarrow \alpha$; $A \leftarrow \{\}$;
3 $S' \leftarrow \{l \mid l \in S, l \notin \alpha, \neg l \notin \alpha\}$;
4 **while** LB $<$ UB ***and*** $S' \neq \emptyset$ **do**
5 $\quad l \leftarrow$ `PickSoftLiteral`(S') ;
6 $\quad S' \leftarrow S' \setminus \{l\}$;
7 $\quad A \leftarrow A \cup \{l\}$;
8 $\quad \alpha' \leftarrow \alpha \cup \{l\}$;
9 $\quad (fc, fs, \alpha') \leftarrow$ `UPforLA`(H, K, α', l) ;
10 $\quad$ **if** $fc \neq \bot$ ***or*** $fs \neq \bot$ **then**
11 $\quad\quad \kappa \leftarrow$ AnalyzeImplicationGraph(fc, fs);
12 $\quad\quad K \leftarrow K \cup \{\kappa\}$;
13 $\quad\quad$ LB $\leftarrow$ LB $+ w(\kappa)$;
14 $\quad\quad$ **foreach** $l \in \kappa$ **do**
15 $\quad\quad\quad rw(l) \leftarrow rw(l) - wlock(l, \kappa)$;
16 $\quad\quad S' \leftarrow S' \cup A \setminus \{l \mid l \in \kappa, rw(l) = 0\}$;
17 $\quad\quad A \leftarrow \{\}$; $\alpha' \leftarrow \alpha$;
18 **return** (LB, K);

such a core taken from [Li *et al.*, 2025]. With the unlocking mechanism and the reuse of the weight locked in previous cores, a tighter lower bound can be computed for WPMS.

Example 1. *A core* κ=$\{l_1(3), l_2(3), l_3(3), l_4(1), l_5(1)\}$ *is given in [Li* et al.*, 2025], where each integer between brackets denotes the weight of the preceding literal locked in the core, and* $w(\kappa) = 4$.

A local core under α is usually detected by propagating a sequence of soft literals until a soft literal becomes falsified or a hard clause is falsified. Algorithm 1 outlines the current lookahead framework procedure employed in WMaxCDCL, omitting details not relevant to this study.

Based on the partial assignment α and the set of unassigned soft literals S', the algorithm actively propagates unassigned soft literals from S' one by one as assumptions (Lines 5–9). Upon the falsification of a hard clause fc or a soft literal fs (Line 10), conflict analysis identifies the set δ of actively propagated soft literals contributing to the falsification and returns a local core κ, defined as δ in the former case and $\delta \cup \{fs\}$ in the latter (Line 11). Then, the lower bound LB is increased by $w(\kappa)$ (Line 13). Accordingly, the remaining weight of each soft literal $l \in \kappa$ is decreased by $wlock(l, \kappa)$ (Line 15). To prepare for the next local core detection, the set of picked soft literals (recorded in A) is restored to the candidate set S', whereas those with zero remaining weight are removed (Line 16). Finally, the algorithm restores α (Line 17).

3 Weight-Aware BnB for WPMS

Existing Branch and Bound (BnB) solvers for MaxSAT often fail to actively exploit the weight information carried by soft clauses, limiting their effectiveness on weighted instances. To address this limitation, we introduce two weight-aware mechanisms in MaxSAT BnB, namely a stratified hardening preprocessing strategy and an in-processing weight-aware lookahead framework to estimate the cost LB of the current partial assignment in state-of-the-art BnB.

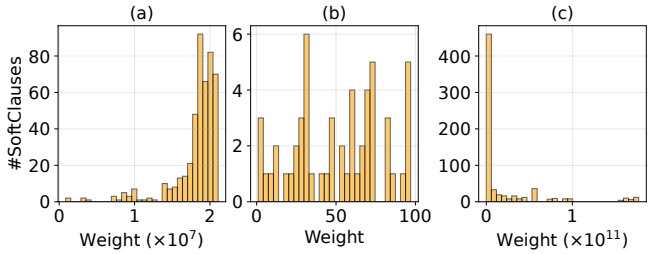

Figure 1: Three representative weight distribution patterns in WPMS benchmarks: (a) concentrated high-weight, (b) random, and (c) heavy-tailed.

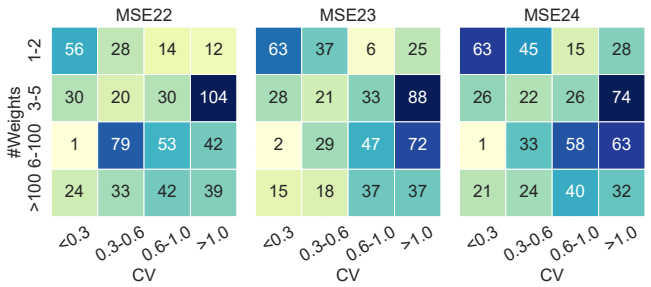

Figure 2: Distribution of WPMS benchmarks from MaxSAT Evaluation 2022–2024, categorized by the number of distinct soft weights (#Weights) and the Coefficient of Variation (CV) of weights. Each cell contains the number of instances.

3.1 Motivation

We characterize the weight structure of WPMS benchmarks from the MaxSAT Evaluations[1] (MSEs) 2022 to 2024 and find that a significant number of instances exhibit a clear hierarchical structure, as illustrated in Figure 1 in both concentrated high-weight and heavy-tailed distributions. BnB solvers such as WMaxCDCL do not exploit this hierarchical structure, which is the first motivation of our study.

Furthermore, we measure the weight distributions of the instances from MSEs 2022 to 2024, using the number of distinct weights and the Coefficient of Variation (CV) computed as $\mathrm{stdDeviation}/\mathrm{meanWeight}$. A low CV suggests a concentrated distribution, while a high CV signifies substantial diversity; notably, a distribution with $CV > 1.0$ is typically regarded as highly dispersed. As illustrated in Figure 2, over 30% of the WPMS instances display a highly dispersed weight distribution, a trend particularly evident in MSE23, where nearly 40% of the instances showed this feature. Specifically, across the entire MSE 2022–2024 benchmark set, 301 instances (17%) satisfy the BMO condition (Equation 1), comprising 80 instances (13%) in MSE22, 118 (21%) in MSE23, and 103 (18%) in MSE24.

Then, we stress out the inefficiency of the existing lookahead framework in dealing with instances exhibiting such highly dispersed weights. Specifically, classical lookahead

[1]https://maxsat-evaluations.github.io

Algorithm 2: Stratified-Hardening

Input: H: hard clauses; S: soft clauses.

1 **if *not*** `IsBMOInst(`S`)` **then**
2 **return** MaxSATSolver(H, S);
3 $w_{max} \leftarrow \infty$; $state \leftarrow$ *SAT*;
4 **while** $S \neq \emptyset$ ***and*** $state =$ *SAT* **do**
5 $w_{max} \leftarrow \max\{w(c) \mid c \subseteq S, w(c) < w_{max}\}$;
6 $hardens \leftarrow \{c \mid c \subseteq S, w(c) = w_{max}\}$;
7 $H \leftarrow H \cup hardens$;
8 $state \leftarrow$ SATsolver(H) ;
9 **if** $state =$ *SAT* **then**
10 $S \leftarrow S \setminus hardens$;
11 **else**
12 $H \leftarrow H \setminus hardens$;
13 **return** MaxSATsolver(H, S) ;

does not prioritize soft literals based on their weights, potentially mixing literals of significantly disparate magnitudes into a local core. Since the weight of a local core is constrained by the minimum weight of the soft literals within it, this leads to the generation of low-weight local cores, diminishing the efficiency of the lookahead framework.

This limitation is concretely illustrated in Example 2, where the lookahead procedure needs to detect two local cores to meet the pruning condition. This is because the first detected local core κ_1 contains the soft literal l_3 with weight $w(l_3) = 1$, which makes κ_1 low-weight and reduces the lookahead efficiency.

Example 2. *Consider soft literals* $S = \{l_1, l_2, l_3\}$ *with weights* $w(l_1) = w(l_2) = 10$, $w(l_3) = 1$, $\mathrm{UB} = 10$, *and* $H = (x_1 \vee \neg l_1 \vee \neg l_3) \wedge (\neg l_1 \vee \neg l_2 \vee l_3)$. *The remaining weight* $rw(l)$ *of each literal* l *is initialized to its weight* $w(l)$. *Given the partial assignment* $\alpha = \{\neg x_1\}$, *we illustrate a typical case of the lookahead procedure:*

1. *Propagating* l_1 *implies* $\neg l_3$, *yielding a local core* $\kappa_1 = \{l_1, l_3\}$ *with weight* $w(\kappa_1) = \min(rw(l_1), rw(l_3)) = 1$. *The remaining weights are updated to* $rw(l_1) \leftarrow 9, rw(l_3) \leftarrow 0$ *and* $\mathrm{LB} = 1$.
2. *Propagating* l_1 *makes* l_3 *falsified and the second hard clause unit, making* $l_2 = 0$. *This identifies a local core* $\kappa_2 = \{l_1, l_2\}$ *with weight* $w(\kappa_2) = \min(rw(l_1), rw(l_2)) = 9$. *Subsequently, the remaining weights are updated to* $rw(l_1) \leftarrow 0$ *and* $rw(l_2) \leftarrow 1$, *and* $LB = 10$. *Finally, the lookahead terminates since* $LB \geq UB$.

3.2 Stratified Hardening Strategy

In this subsection, we propose a preprocessing strategy for BnB solvers to exploit the hierarchical structure in WPMS instances, whenever it exists, inspired by stratification techniques in core-guided algorithms. The strategy consists of hardening high-weight soft clauses to simplify BMO instances. It is implemented in Algorithm 2.

The algorithm first checks whether the weight distribution satisfies the BMO condition, as defined in Equation (1), via the `IsBMOInst` procedure (Line 1). For confirmed BMO instances, it iteratively hardens soft clauses in descending order of weights and invokes a SAT oracle to check the satisfiability of the extended hard clause set H (Lines 5–8). If H remains satisfiable, the hardened clauses are removed from the soft clause set S (Line 10). Conversely, if H becomes unsatisfiable, the algorithm restores consistency by discarding the clauses hardened in the current iteration and terminates the hardening phase (Line 12). Finally, the simplified formula (H, S) is passed to the MaxSAT solver (Line 13).

The main difference between our algorithm and the stratified approach in core-guided MaxSAT solvers is that the latter is applied throughout the iterative solving process, whereas our approach is restricted to the preprocessing phase. Although the preprocessing procedure halts when unsatisfiability arises, which may restrict the extent of reduction, experimental results indicate that it is particularly effective for WMaxCDCL, notably boosting its performance.

Algorithm 3: UPforLA (Weight-Aware Version)

Input: H: hard clauses, K: local cores, α: partial assignment, l: a soft literal to propagate.
Output: fc: a falsified clause or $\bot$, fs: a falsified soft literal or $\bot$, α: extended partial assignment.

1 $Q \leftarrow \{l\}$; $\Gamma \leftarrow \emptyset$; $\Phi \leftarrow \emptyset$;
2 **while** $Q \neq \emptyset$ **do**
3 $p \leftarrow \mathrm{Pop}(Q)$;
4 **for *each*** *clause* $c \in H$ *containing* $\neg p$ **do**
5 **if** c *is falsified under* α **then**
6 **return** $(c, \bot, \alpha)$;
7 **else if** c *is unit under* α **then**
8 Let l be the sole unassigned literal in c;
9 $\alpha \leftarrow \alpha \cup \{l\}$;
10 **if** $\neg l$ *is a soft literal* **then**
11 **if** $rw(\neg l) > 0$ **then**
12 $\Gamma \leftarrow \Gamma \cup \{\neg l\}$;
13 **else**
14 $\Phi \leftarrow \Phi \cup \{\kappa \mid \kappa \in K, \kappa$ is unlocked$\}$;
15 Push(Q, l);
16 $fs \leftarrow$ `WA-Select`(Γ, Φ) ;
17 **return** $(\bot, fs, \alpha)$;

3.3 Weight-Aware Lookahead

In this subsection, we address the limitations of existing lookahead procedures in BnB for WPMS, which often fail to identify high-quality cores, because they do not actively exploit weight information during lower bound computation. Specifically, note that most cores $\kappa = \{l_1, \ldots, l_k\}$ are obtained by actively propagating $l_1, \ldots, l_{k-1}$ as assumptions, falsifying l_k. However, as indicated in Example 2, the weight of a core $\kappa = \{l_1, \ldots, l_k\}$ depends on the remaining weight of each soft literal in κ. The existing lookahead procedures do not actively search for high remaining weights. In this subsection, we introduce a weight-aware approach to address

the actively propagated soft literals and the falsified literals separately. The purpose is to overcome the limitations of the existing lookahead framework by detecting high-weight cores to increase LB quickly.

To address the actively propagated soft literals, we introduce a new soft literal selection heuristic in module `PickSoftLiteral` called in Line 5 of Algorithm 1. In the baseline approach, the selection of the soft literals to propagate relies primarily on their recent activity in conflicts, neglecting the significance of their associated weights. In contrast, our heuristic gives precedence to soft literals with higher remaining weights, employing activity scores only to distinguish between candidates of equal weight.

To address the falsified soft literals, we modify the unit propagation process in module `UPforLA` (Algorithm 3) called in Line 9 of Algorithm 1. In the baseline version of `UPforLA`, unit propagation aborts as soon as a soft literal is falsified (fs) (Line 10). In contrast, our proposed approach continues propagation until a hard clause is falsified (Line 6) or there are no more unit clauses to propagate (i.e., propagation queue Q becomes empty), so that several soft literals can be falsified.

Example 3. *Considering the same input as in Example 2, we illustrate the execution trace of the modified lookahead:*

Propagating l_1 implies $\neg l_3$. In Example 2, the propagation stopped and a core κ_1=$\{l_1, l_3\}$ with $w(\kappa_1)$=1 is identified. However, in our new strategy, we do not stop unit propagation, but continue falsifying l_2. Then, we have two falsified soft literals $\{l_2, l_3\}$. Since $rw(l_2)$>$rw(l_3)$, we choose l_2 to identify the local core κ_2=$\{l_1, l_2\}$ with a weight of $w(\kappa_2)$=10, increasing the lower bound to LB=10 by itself. So, only one local core is sufficient to prune the current branch, because we already have $LB \geq UB$.

Example 3 illustrates a simple situation in which all falsified literals have positive remaining weights, and we just need to choose the one with the highest remaining weight to form a high-weight core. However, in the latest version of WMaxCDCL (version of MSE 2024), a sophisticated unlocking mechanism is implemented to reuse the soft literals already locked in cores. By the unlocking mechanism, a core κ is unlocked if the total locked weights of the falsified literals in κ is equal to or greater than $w(\kappa)$ during unit propagation, namely, $\sum_{\neg l \in \alpha} wlock(l, \kappa) \geq w(\kappa)$, where α is the current partial assignment. It is shown in [Li *et al.*, 2025] that all unassigned literals in an unlocked core, together with their weights locked in this core, can be used to detect a new core. Indeed, the contribution of a core κ to LB is $w(\kappa)$. The falsification of an unassigned literal when or after κ is unlocked can produce an additional contribution to LB w.r.t. $w(\kappa)$. We need to choose a literal falsified when or after κ is unlocked for detecting a high-weight core.

Example 4. *Let κ_1=$\{l_1(4), l_2(4), l_3(6), l_4(3), l_5(3)\}$ with $w(\kappa_1)$=7, and each integer between brackets denoting the weight of the preceding literal locked in κ_1 (e.g., $wlock(l_3, \kappa_1)$=6), the remaining weight $rw(l)$=0 for each $l \in \kappa_1$, and* UB=10. *Since* LB=$w(\kappa_1)$=7. *We need to detect new cores to prune the current branch. Let l_6 be a soft literal with $rw(l_6)$=3. Propagating l_6 falsifies successively l_1, l_2 and l_3.*

The two combinations $(l_1$=$0, l_2$=$0)$ and $(l_1$=$0, l_3$=$0)$ both unlock κ_1. The current version of WMaxCDCL will use $(l_1$=$0, l_2$=$0)$ to derive a new core, because l_2 is the first soft literal unlocking κ_1, and the new core is κ_2=$\{l_1(4), l_2(4), l_3(6), l_4(3), l_5(3), l_6(1)\}$ with $w(\kappa_2)$=8, including κ_1. To see this, we distinguish two branches: l_6=0 and l_6=1. For any complete extension of α including l_6=0, the total minimum falsified weight is $w(\kappa_1) + rw(l_6)$=10. If l_6=1, the total minimum falsified weight using $l_1 = 0$ and $l_2 = 0$ is $wlock(l_1, \kappa_1) + wlock(l_2, \kappa_1) = 8$. Since $w(\kappa_2)$ is the minimum among the two branches, $w(\kappa_2) = min(10, 8) = 8$. So, LB=$w(\kappa_2)$=8 < UB, *which is not enough to prune the current branch.*

Note that if $rw(l_6)$ were 1, the total minimum falsified weight for branch l_6=1 would be reduced to $w(\kappa_1) + rw(l_6)$=8, but $w(\kappa_2)$ would always be 8. So, we only need to lock weight 1 of l_6 in κ_2.

However, we can also choose $(l_1$=$0, l_3$=$0)$ and derive a new core κ_3=$\{l_1(4), l_2(4), l_3(6), l_4(3), l_5(3), l_6(3)\}$ with $w(\kappa_3)$=10, increasing LB *to* 10 *and allowing us to prune the current branch. In fact, in branch l_6=0, the total minimum falsified weight is always $w(\kappa_1) + rw(l_6)$=10, while in branch l_6=1, the total minimum falsified weight using l_1=0 and l_3=0 is $wlock(l_1, \kappa_1)$+$w(l_3, \kappa_1)$=4+6=10. Since $w(\kappa_3)$ is the minimum among the two branches, we have $w(\kappa_3)$=$min(10, 10)$=10.*

Note that if $rw(l_6)$ were smaller than 3, the total minimum falsified weight would be smaller than 10 for branch $l_6 = 0$. So, we need to lock weight 3 of l_6 in κ_3 to make $w(\kappa_3)$=10.

Note that we can use the combination $(l_1$=$0, l_2$=$0, l_3$=$0)$ and obtain a new core κ_4 with $w(\kappa_4) = 10$. The drawback is that, in a more realistic situation, the reasons falsifying l_2 and the reasons falsifying l_3 can be different. If we take both l_2 and l_3, we need to take all these reasons into account to form the new core, which generally results in a larger new core and reduces the efficiency in detecting the next new cores.

Also note that if l_4 is falsified after l_1, we cannot take the combination (l_1, l_4), although it unlocks κ_1, because this combination does not give any additional falsified weight w.r.t. $w(\kappa_1)$. In fact, $wlock(l_1, \kappa_1)$+$wlock(l_4, \kappa_1)$−$w(\kappa_1)$=0.

In the general case, we define the notion of *additional falsified weight*, aw for short, w.r.t. existing cores, during unit propagation for detecting a new core. If the falsified literal l has a positive remaining weight $rw(l)$, then $aw(l) = rw(l)$. Otherwise, l has weight locked in existing cores. The additional falsified weight of l depends on its falsification time and the unlocking time of the cores it belongs to.

The falsification time of l during a core detection is defined to be the smallest cardinality of α after α receives $\neg l$ (See Line 9 in Algorithm 3), denoted as $\mathrm{falsifyTime}(l)$, or $\mathrm{falsifyT}(l)$ for short. The unlocking time of an existing core κ is defined to be the first $\mathrm{falsifyTime}(l)$ making $\sum_{\neg l \in \alpha} wlock(l, \kappa) \geq w(\kappa)$, denoted as $\mathrm{unlockTime}(\kappa)$, or $\mathrm{unlockT}(\kappa)$ for short. By definition, $\mathrm{unlockTime}(\kappa) = \infty$ if κ is not unlocked.

The computation of additional falsified weight of l w.r.t. κ,

Algorithm 4: WA-Select

Input: Γ: falsified soft literals, Φ: unlocked cores.
Output: fs: a falsified soft literal or $\bot$.

1 $fs \leftarrow \bot$; $w_{max} \leftarrow 0$;
2 **for** ***each*** $l \in \Gamma$ **do**
3 **if** $rw(l) > w_{max}$ **then**
4 $fs \leftarrow l$; $w_{max} \leftarrow rw(l)$;
5 **for** ***each*** $\kappa \in \Phi$ **do**
6 **for** ***each*** $l \in \kappa$ *s.t.* falsifyT$(l) >$ unlockT **(Case (1)) *or*** falsifyT$(l) \geq$ unlockT **(Case (2)) do**
7 Compute $aw(l, \kappa)$ according to **Case (1)** or **Case (2)**
8 **if** $aw(l, \kappa) > w_{max}$ **then**
9 $fs \leftarrow l$;
10 $w_{max} \leftarrow aw(l, \kappa)$;
11 **return** fs;

denoted as $aw(l, \kappa)$, depends on two cases:

Case 1: $w(\kappa) = \sum_{\neg l' \in \alpha \wedge \text{falsifyT}(l') \leq \text{unlockT}(\kappa)} wlock(l', \kappa)$

Case 2: $w(\kappa) < \sum_{\neg l' \in \alpha \wedge \text{falsifyT}(l') \leq \text{unlockT}(\kappa)} wlock(l', \kappa)$

In Case (1), $aw(l, \kappa) = wlock(l, \kappa)$ for each l falsified after κ is unlocked (i.e., such that falsifyT$(l) > unlockT(\kappa)$).

In Case (2), we define

$$buw(\kappa) = \sum_{\neg l \in \alpha \wedge \text{falsifyT}(l) < \text{unlockT}(\kappa)} wlock(l, \kappa) \quad (2)$$

for *base unlocking weight* of κ. Then, the additional falsified weight of each soft literal l falsified in κ such that falsifyT$(l) \geq$ unlockT(κ) is computed using the base unlocking weight as follows:

$$aw(l, \kappa) = wlock(l, \kappa) + buw(\kappa) - w(\kappa) \quad (3)$$

Example 5. *Consider the same core* $\kappa_1 = \{l_1(4), l_2(4), l_3(6), l_4(3), l_5(3)\}$ *in Example 4, with* $w(\kappa_1) = 7$*, the remaining weight* $rw(l) = 0$ *for each* $l \in \kappa_1$*, and* $\alpha = \emptyset$*. Propagating a literal* l_6 *falsifies successively* l_1*,* l_2 *and* l_3*, updating* α *to* $\{l_6, \neg l_1, \neg l_2, \neg l_3\}$.

We have falsifyT(l_1)=2, falsifyT(l_2)=3, falsifyT(l_3)=4, unlockT(κ_1) = falsifyT(l_2) = 3. *We are in Case (2), because* $w(\kappa_1) < wlock(l_1, \kappa_1) + wlock(l_2, \kappa_1)$. *Then,* $buw(\kappa_1) = wlock(l_1, \kappa_1) = 4$, $aw(l_2, \kappa_1) = wlock(l_2, \kappa_1) + buw(\kappa_1) - w(\kappa_1) = 1$, $aw(l_3, \kappa_1) = wlock(l_3, \kappa_1) + buw(\kappa_1) - w(\kappa_1) = 3$.

Since $aw(l_3, \kappa_1) > aw(l_2, \kappa_1)$*, we should choose the combination* $(l_1 = 0, l_3 = 0)$ *to form the new core.*

We now present Algorithm 3 and Algorithm 4, whose purpose is to select the greatest additional falsified weight to form a high-weight core. In Algorithm 3, if a hard clause is falsified, it is simply returned, because the additional falsified weight is considered to be ∞ in this case. Otherwise, every falsified soft literal with positive remaining weight is collected into Γ (Line 12), and every unlocked core is collected into Φ (Line 14). Then, Algorithm 4 is called to select the falsified literal with the greatest additional falsified weight, done via lines 2-4 which select the falsified soft literal with the greatest remaining weight from Γ, and lines 5-10 which select the falsified soft literal with the greatest additional falsified weight w.r.t. an unlocked core from Φ. The algorithm returns the falsified soft literal with the global maximum *additional falsified weight*.

Solver	MSE22 (607)		MSE23 (558)		MSE24 (571)	
	#sol	PAR2	#sol	PAR2	#sol	PAR2
Open-WBO	344	4556	365	3569	359	3854
Pacose	378	3966	387	3200	390	3346
WMaxCDCL	421	3309	392	3119	405	3095
CGSS2	413	3358	417	2668	421	2798
UwrMaxSAT	413	3374	421	2585	430	2637
EvalMaxSAT	420	3384	424	2711	426	2830
WM-SWA	**422**	3272	430	2477	**439**	2543
CASHWMS	**422**	3215	**433**	2398	436	2540

Table 1: Performance of WMaxCDCL, its variant (**WM-SWA**), and six independent state-of-the-art solvers on the complete benchmark set of the MSE 2022–2024 weighted track.

Solver	MSE22 (607)		MSE23 (558)		MSE24 (571)	
	#sol	PAR2	#sol	PAR2	#sol	PAR2
WMaxCDCL	421	3309	392	3119	405	3095
WM-WA$^-$	420	3289	404	2928	414	2955
WM-WA	**423**	3266	407	2891	414	2958
WM-S	421	3300	415	2705	429	2687
WM-SWA	422	3272	**430**	2477	**439**	2543

Table 2: Performance of WMaxCDCL, and all its variants on the complete benchmark set of the MSE 2022–2024 weighted track.

4 Experimental Results

In this section, we evaluate the proposed approach against state-of-the-art MaxSAT solvers, and the different strategies in the approach. We implemented our methods on top of the MaxSAT Evaluation 2024 version of WMaxCDCL [Li *et al.*, 2024], which integrates the unlocking mechanism in [Li *et al.*, 2025]. The code was compiled using GCC 12.2.0 with -O3 optimization and static linking.

Proposed variants. To assess our contributions, we evaluated the following variants of WMaxCDCL (**WM**), with source code available in the *supplementary material*[2]:

- **WM-S**: Incorporates the stratified hardening strategy described in Section 3.2.
- **WM-WA**: Implements the weight-aware lookahead framework detailed in Section 3.3.
- **WM-WA$^-$**: An ablation version of WM-WA that modifies Algorithm 4, by selecting all falsified literals in a core $\kappa \in \Phi$ and uses $aw = \sum_{l \in \kappa \wedge \neg l \in \alpha} wlock(l, \kappa) - w(\kappa)$, e.g., it uses the combination $\{l_1 = 0, l_2 = 0, l_3 = 0\}$ in Example 4, instead of $\{l_1 = 0, l_2 = 0\}$ or $\{l_1 = 0, l_3 = 0\}$.

[2]https://github.com/zjl9959/IJCAI26.git

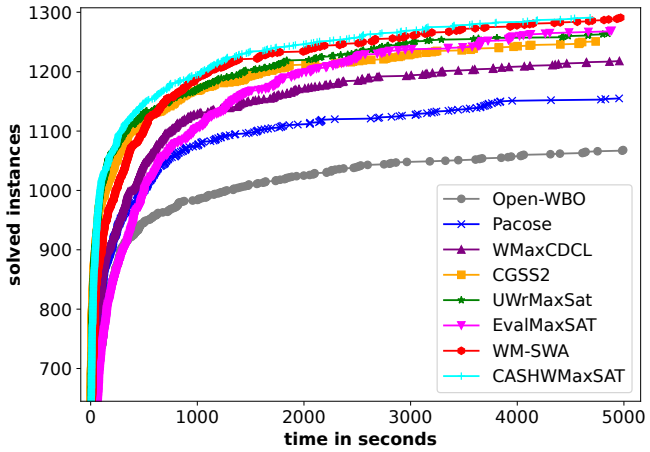

Figure 3: Cumulative distribution of solved instances for state-of-the-art MaxSAT solvers and WM-SWA across the MSE 2022–2024 benchmarks.

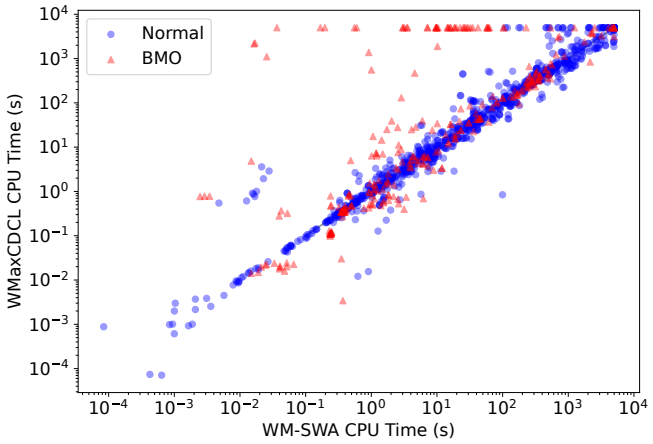

Figure 4: Scatter plot comparing CPU time (s) of WM-SWA against WMaxCDCL on the MSE 2022-2024 benchmarks. The axes are on a logarithmic scale.

- **WM-SWA**: Combines both the stratified hardening strategy and the weight-aware lookahead framework.

Compared solvers. We downloaded state-of-the-art solvers participating in the three most recent MaxSAT Evaluations (MSE 2022–2024). To ensure a fair comparison, we excluded portfolio-based approaches and focused on six independent solvers: Open-WBO [Martins *et al.*, 2023], Pacose [Paxian and Becker, 2024], CGSS2 [Ihalainen *et al.*, 2024], UwrMaxSAT [Piotrów, 2024], EvalMaxSAT [Avellaneda, 2024], and CASHWMaxSAT [Pan *et al.*, 2024]. Notably, CASHWMaxSAT, UwrMaxSAT, and EvalMaxSAT ranked as the top three solvers in the weighted track of MSE 2024. Each solver was configured using the settings that yielded the best performance in its most recent competition participation.

Benchmarks. For the empirical evaluation, we first conduct an in-depth analysis of the WPMS instances from MSE 2022, 2023, and 2024, followed by a comprehensive robustness check across the extended MSE 2019–2024 benchmarks. The dataset spans diverse domains (e.g., combinatorial optimization, AI, and bioinformatics) to ensure a representative and robust evaluation.

Experimental environment. The experiments were conducted on the MatriCS platform[3], equipped with AMD EPYC 7502 processors (2.5 GHz) running Linux. Each instance was restricted to a single CPU core and 31 GB of RAM, with a timeout of 5000 seconds. The 5000s limit was chosen to calibrate our evaluation to the MSE24 performance benchmarks, as our hardware is slower than the official MSE infrastructure.

4.1 Overall Evaluation

Table 1 presents the results of the proposed variants and state-of-the-art solvers on the MSE 2022–2024 WPMS benchmarks. The evaluation metrics include the number of solved instances (#sol) and the Penalized Average Runtime with a factor of 2 (PAR2) [Hutter *et al.*, 2009], which was first used in MSE 2024. The data indicate that while the baseline WMaxCDCL remains competitive on MSE22, it lags behind other state-of-the-art solvers on MSE23 and MSE24. By leveraging weight distribution information, our proposed variant, WM-SWA, effectively addresses this shortcoming. Across the combined MSE22-24 benchmarks, WM-SWA solves a total of 1291 instances—an improvement of 73 instances over the baseline—matching the performance of the current leading solver, CASHWMaxSAT. Notably, WM-SWA solves the most instances (439) in MSE24, surpassing all other competitors.

Figure 3 presents the number of solved instances over time, demonstrating that WM-SWA performs on par with CASHWMaxSAT while outperforming all other solvers. Notably, the solver rankings remain stable within the $[1000, 5000]$ seconds interval; the only exception is EvalMaxSAT, which stabilizes after 3933 seconds without changing the ranking of WM-SWA. Ultimately, these results demonstrate that our proposed weight-aware strategies enable the BnB solver to achieve state-of-the-art performance on WPMS solving.

4.2 Evaluation of Strategies

We analyzed the performance of the proposed variants compared to WMaxCDCL. As summarized in Table 2, all proposed variants enhance the overall performance of WMaxCDCL. Specifically, WM-S solves an additional 23 and 24 instances in MSE23 and MSE24, respectively. Meanwhile, WM-WA yields improvements across all years, solving 2, 15, and 9 more instances in MSE22, MSE23, and MSE24. Furthermore, the performance gain of WM-SWA reflects the cumulative benefits of WM-S and WM-WA, indicating that these two strategies are synergistic and complementary.

It is important to note that the improvement on MSE22 is relatively limited compared to the substantial gains on MSE23 and MSE24, which is mainly attributed to two factors: (1) Increased Complexity: MSE22 instances are harder (69.5% solved by the best solver) compared to MSE23/24 ($\approx$ 77%) (Table 1); (2) Structural Mismatch: MSE22 contains significantly fewer BMO instances (Section 3.1), which limits

[3]https://www.matrics.u-picardie.fr/

Diff(%)	MSE22	MSE23	MSE24
$\Delta coreWeight$	+27.39%	+21.94%	+16.34%
$\Delta coreSize$	-1.36%	-1.52%	-0.55%
$\Delta coresPerLA$	-10.51%	-7.08%	-7.58%

Table 3: Relative changes in average core weight, size, and cores per lookahead ($\Delta coresPerLA$) of WM-WA w.r.t WMaxCDCL.

the utility of the stratified hardening strategy. Consequently, as shown in Table 2, weight-aware lookahead emerges as the primary performance driver on MSE22. This enables WM-WA to outperform other solvers, confirming that lookahead mechanisms are more critical than stratified hardening for harder, less-structured instances.

As illustrated in Figure 4, WM-SWA effectively reduces solving times across the majority of instances when compared to WMaxCDCL. Notably, this reduction in CPU time is particularly pronounced for instances featuring the BMO structure (Equation 1). To investigate this further, we revisit the three representative weight distribution patterns presented in Figure 1. We observe that WM-S excels on instances with concentrated high-weight and heavy-tailed distributions—many of which satisfy the BMO condition. For instance, the *frb* ($CV < 0.3$) and *drmx-cryptogen* ($0.3 < CV < 0.6$) families clearly exemplify this behavior. Meanwhile, WM-WA demonstrates superior efficiency on instances exhibiting more random weight patterns, such as the *judgement-aggregation* family ($0.4 < CV < 1.3$). Ultimately, these findings suggest that the performance of the proposed strategies is governed by the interaction between the BMO property and weight diversity, rather than dispersion alone.

Additionally, the ablated variant WM-WA$^-$ solves 3 fewer instances than WM-WA in both MSE22 and MSE23. This demonstrates that the literal selection strategy proposed in Algorithm 4 outperforms the naive strategy of selecting all falsified soft literals within the core. As discussed in Example 4, although the naive strategy can select a core with a larger *additional falsified weight*, it also uses more literals to form the new core.

Finally, Table 3 offers insights into the performance of WM-WA compared to WMaxCDCL. The data indicates that for each benchmark, WM-WA detects cores with higher weights and smaller sizes, and requires fewer cores per lookahead to prune the search. Aggregated across the complete benchmark set of MSE 2022–2024, the changes are also substantial: core weight (+22.04%), core size (-1.15%), and cores per lookahead (-8.46%). This confirms that our strategy successfully guides the lookahead search to identify high-weight cores, thereby improving pruning efficiency.

4.3 Extended Evaluation

To validate robustness across a broader scope, we extend our evaluation to a six-year dataset (MSE 2019–2024). To avoid duplication, we excluded instances from the evaluation of year $2000 + k$ ($25 > k > 19$) if they appeared in years 2019, ..., 2000+k-1. The extended dataset comprises 2223 weighted partial MaxSAT instances.

We compare WM-SWA against leading MaxSAT solvers

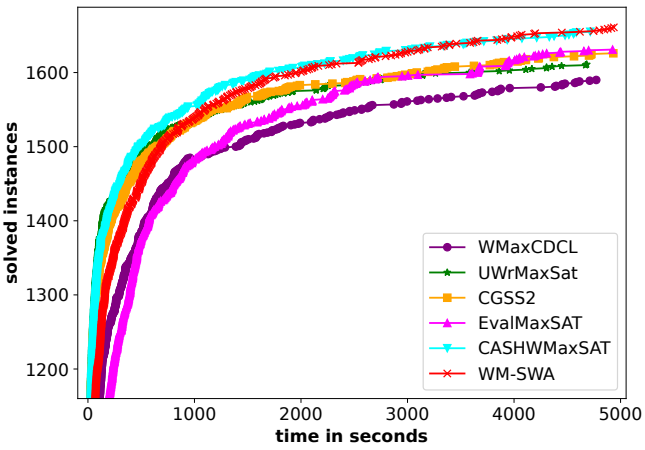

Figure 5: Cumulative distribution of solved instances for top MaxSAT solvers across the MSE 2019–2024 benchmarks.

recognized in recent MaxSAT Evaluations (Table 1). Specifically, across the comprehensive MSE 2019–2024 benchmarks, WM-SWA solves 1661 instances, successfully outperforming CASHWMaxSAT (1655), EvalMaxSAT (1631), CGSS2 (1626), UwrMaxSAT (1611), and WMaxCDCL (1590). As shown in Figure 5, WM-SWA demonstrates superior performance compared to other leading MaxSAT solvers across this six-year dataset. These results confirm that our refinements on WMaxCDCL effectively bridge the performance gap for weighted instances, establishing a consistent and significant improvement over existing independent SAT-based WPMS solvers.

5 Conclusion

In this paper, we introduced two weight-aware strategies designed to enhance Branch-and-Bound (BnB) MaxSAT solvers for WPMS. The first approach implements a hardening mechanism for high-weight soft clauses within a BMO structure. The second approach addresses fundamental limitations in current lookahead frameworks by introducing the concept of *additional falsified weight*. This metric, calculated based on literal falsification and core unlock times, enables the solver to prioritize soft literals that lead to the detection of high-weight, small-size local cores.

We integrated these approaches into WMaxCDCL, a state-of-the-art BnB solver. Our experimental evaluation on datasets from MSE 2022–2024 demonstrates that while each approach independently improves performance, their combination is particularly synergistic, allowing WMaxCDCL to solve *73 additional instances*. These results effectively *bridge the performance gap* between BnB and SAT-based solvers for WPMS.

For future work, we intend to develop an efficient weight-aware further-lookahead mechanism, aiming to break selected low-weight or large-size cores when the standard lookahead fails to increase the lower bound (LB<UB), thereby facilitating the discovery of higher-quality cores and further accelerating the search process.

Acknowledgements

This work is partially supported by the French National Research Agency (ANR) under the Chair MASSAL'IA (ANR-19-CHIA-0013-01), co-funded by the French electricity distribution network operator Enedis, as well as under the project ANR-24-CE23-6126 (BforSAT). It was granted access to HPC resources of "Plateforme MatriCS" within University of Picardie Jules Verne, co-funded by the European Union with the European Regional Development Fund (FEDER) and the Hauts-De-France Regional Council.

References

[Abramé and Habet, 2014] André Abramé and Djamal Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *J. Satisf. Boolean Model. Comput.*, pages 89–128, 2014.

[Allouche *et al.*, 2014] 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.

[Ansótegui *et al.*, 2009] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *International conference on theory and applications of satisfiability testing*, pages 427–440, 2009.

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

[Ansótegui *et al.*, 2012] Carlos Ansótegui, Maria Bonet, Jordi Levy, and Chu-Min Li. Analysis and generation of pseudo-industrial maxsat instances. *Frontiers in Artificial Intelligence and Applications*, 248:173–184, 01 2012.

[Avellaneda, 2024] Florent Avellaneda. Evalmaxsat 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 8, 2024.

[Bacchus *et al.*, 2021] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, pages 929 – 991. Frontiers in Artificial Intelligence and Applications. IOS PRESS, Netherlands, 2 edition, 2021.

[Bacchus, 2022] Fahiem Bacchus. Maxhs in the 2022 maxsat evaluation. *MaxSAT Evaluation 2022 Solver and Benchmark Descriptions*, pages 17–18, 2022.

[Berg *et al.*, 2023] 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.

[Berre and Parrain, 2010] Daniel Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7:59–64, 01 2010.

[Cherif *et al.*, 2020] Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of max-sat resolution through up-resilience. *Artif. Intell.*, 289:103397, 2020.

[Cherif *et al.*, 2024] Sami Cherif, Heythem Sattoutah, Chu-Min Li, Corinne Lucet, and Laure Brisoux Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability. In Paul Shaw, editor, *CP 2024*, volume 307 of *LIPIcs*, pages 34:1–34:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Chu *et al.*, 2023] Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):3915–3923, Jun. 2023.

[Coll *et al.*, 2025] Jordi Coll, Chu-Min Li, Shuolin Li, Djamal Habet, and Felip Manyà. Solving weighted maximum satisfiability with branch and bound and clause learning. *Computers & Operations Research*, 183:107195, 2025.

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

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

[Heras *et al.*, 2008] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, pages 1–32, 2008.

[Hutter *et al.*, 2009] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *J. Artif. Int. Res.*, 36(1):267–306, September 2009.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an Efficient MaxSAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, September 2019.

[Ihalainen *et al.*, 2024] Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Cgss2 and cgss2-abstcg in the 2024 maxsat evaluation. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 13–14, 2024.

[Koshimura *et al.*, 2012] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 95–100, 2012.

[Kuegel, 2010] Adrian Kuegel. Improved exact solver for the Weighted MAX-SAT problem. In *Proceedings of Workshop Pragmatics of SAT, POS-10, Edinburgh, UK*, pages 15–27, 2010.

[Li and Manyà, 2021] Chu Min Li and Felip Manyà. Chapter 23. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, February 2021.

[Li and Quan, 2010] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the max-

imum clique problem. In *Proceedings of AAAI 2010*, pages 128–133, 2010.

[Li *et al.*, 2007] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, pages 321–359, 2007.

[Li *et al.*, 2010] Chu-Min Li, Felip Manya, Zhe Quan, and Zhu Zhu. Exact minsat solving. In *International Conference on Theory and Applications of Satisfiability Testing (SAT-2010)*, pages 363–368, 2010.

[Li *et al.*, 2016] Chu-Min Li, Felip Manya, and JR Soler. A clause tableau calculus for maxsat. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2016, pages 766–772. International Joint Conferences on Artificial Intelligence, 2016.

[Li *et al.*, 2021] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining Clause Learning and Branch and Bound for MaxSAT. In Laurent D. Michel, editor, *CP 2021*, volume 210 of *LIPIcs*, pages 38:1–38:18, Dagstuhl, Germany, 2021.

[Li *et al.*, 2022] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound maxsat solvers with clause learning. *AI Communications*, 35(2):131–151, 2022.

[Li *et al.*, 2024] Shuolin Li, Chu Min Li, Jordi Coll, Djamal Habet, Felip Manyà, and Kun He. Wmaxcdcl in maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 17–18, 2024.

[Li *et al.*, 2025] Shuolin Li, Chu-Min Li, Jordi Coll, Djamal Habet, and Felip Manyà. Improving the lower bound in branch-and-bound algorithms for maxsat. AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025.

[Marques-Silva *et al.*, 2011] Joao Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: Algorithms & applications. 62(3–4):317–343, 2011.

[Martins *et al.*, 2014] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of SAT 2014*, pages 438–445, 2014.

[Martins *et al.*, 2023] Ruben Martins, Norbert Manthey, Miguel Terra-Neves, Vasco Manquinho, and Inês Lynce. Open-wbo @ maxsat evaluation 2023. *MaxSAT Evaluation 2023 Solver and Benchmark Descriptions*, pages 18–19, 2023.

[Pan *et al.*, 2024] Shiwei Pan, Yiyuan Wang, Shaowei Cai, Jiangnan Li, Wenbo Zhu, and Minghao Yin. Cashwmaxsat-disjcad: Solver description. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 25, 2024.

[Pan *et al.*, 2025] Shiwei Pan, Yiyuan Wang, and Shaowei Cai. An efficient core-guided solver for weighted partial maxsat. In James Kwok, editor, *IJCAI-25*, pages 2647–2656. International Joint Conferences on Artificial Intelligence Organization, 8 2025. Main Track.

[Paxian and Becker, 2024] Tobias Paxian and Bernd Becker. Pacose: An iterative sat-based maxsat solver. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, page 26, 2024.

[Piotrów, 2020] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136, 2020.

[Piotrów, 2024] Marek Piotrów. Uwrmaxsat entering the maxsat evaluation 2024. *MaxSAT Evaluation 2024 Solver and Benchmark Descriptions*, pages 27–28, 2024.