Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

An Efficient Core-Guided Solver for Weighted Partial MaxSAT

Shiwei Pan[1,3], **Yiyuan Wang**[1,3*], **Shaowei Cai**[2,4]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of System Software, Institute of Software, Chinese Academy of Sciences
[3]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, China
[4]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
{pansw779, wangyy912}@nenu.edu.cn, caisw@ios.ac.cn

Abstract

The maximum satisfiability problem (MaxSAT) is a crucial combinatorial optimization problem with widespread applications across various critical domains. This paper presents CASHWMaxSAT, an efficient core-guided MaxSAT solver based on two novel ideas. The first and most important idea is the introduction of an extended stratification technique that progressively focuses on solving high-weight soft clauses. Second, we integrate disjoint unsatisfiable cores with the goal of minimizing the unsatisfiable core, allowing the solver to learn multiple high-quality clauses in a single conflict analysis step. These innovations enable our MaxSAT solver to efficiently identify key constraints and reduce redundant reasoning, significantly enhancing solving efficiency. Experimental results on benchmarks from the complete weighted track of the MaxSAT Evaluations 2022-2024 demonstrate that the proposed methods lead to substantial improvements, with CASHWMaxSAT outperforming state-of-the-art MaxSAT solvers across all benchmarks. Additionally, it enabled us to achieve the top two positions in the exact weighted category of the MaxSAT Evaluation 2024.

1 Introduction

The maximum satisfiability problem (MaxSAT) is a generalization of the well-known and NP-complete Boolean satisfiability problem (SAT) [Cook, 1971]. While both SAT and MaxSAT are fundamentally similar, the optimization aspect of MaxSAT makes it significantly more challenging to solve in practice. Unlike SAT, where the goal is to determine if there exists a satisfied assignment, MaxSAT requires finding an assignment that satisfies the maximum number of clauses, with some clauses necessarily falsified. Many real-world problems can be naturally encoded as MaxSAT problems, such as hardware verification [Biere *et al.*, 2009], model-based diagnosis [Marques-Silva *et al.*, 2015], planning [Kautz *et al.*, 1992], and data analysis [Berg *et al.*, 2019].

*corresponding author

Research on MaxSAT algorithms can be categorized into two primary groups: exact algorithms, which guarantee optimal solutions, and heuristic algorithms, which provide solutions of good quality within a reasonable amount of time, though without guarantees of optimality. This paper focuses on exact algorithms for weighted partial MaxSAT, which seek to maximizes the total weight of satisfied clauses.

Recently, exact MaxSAT solvers can be broadly classified into two categories: solvers based on SAT solvers and solvers that employ branch-and-bound strategies. Early branch-and-bound MaxSAT solvers, such as MaxSatz [Li *et al.*, 2007], MiniMaxSat [Heras *et al.*, 2008], Ahmaxsat [Abramé and Habet, 2014], and Akmaxsat [Kügel, 2010], achieved significant improvements by effectively leveraging MaxSAT resolution rules for local propagation [Li *et al.*, 2007; Heras and Larrosa, 2006; Li *et al.*, 2009] and incorporating lower-bound estimation techniques. Specifically, these solvers utilized unit-clause propagation to identify disjoint inconsistent subsets, which were then used for lower-bound estimation [Li *et al.*, 2005; Li *et al.*, 2006]. However, following Fu and Malik's groundbreaking introduction of SAT-based MaxSAT solvers in 2006 [Fu and Malik, 2006], the performance of branch-and-bound solvers on real-world instances has gradually been surpassed by SAT-based methods. In recent years, a promising direction has emerged with the development of MaxCDCL [Li *et al.*, 2021b; Li *et al.*, 2021a], a branch-and-bound MaxSAT solver that integrates conflict-driven clause learning (CDCL). Subsequently, Li et al. [2025] further enhanced this framework by introducing an unlocking mechanism that reuses relaxation variables to extract more disjoint cores, thereby strengthening the lower bound. MaxCDCL has demonstrated outstanding performance, particularly on unweighted MaxSAT problems. Its weighted version has also shown competitive results compared to SAT-based MaxSAT solvers, marking a significant advancement in the field of branch-and-bound MaxSAT solvers.

The first SAT-based MaxSAT solver was introduced by Fu et al. [2006]. Later, Ansótegui et al. [2009; 2010; 2017] improved the SAT-based solvers for weighted partial MaxSAT. MaxSAT solvers based on SAT solvers can generally be divided into three categories: model-guided, core-guided, and minimum hitting set (MHS)-guided solvers. Model-guided solvers set a cost value k and transform the MaxSAT problem into a SAT problem by checking whether

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

a solution with a cost less than or equal to k exists. The value of k is then gradually reduced until the transformed SAT instance becomes unsatisfiable. Representative solvers of this approach include SAT4J-MaxSAT [Le Berre and Parrain, 2010], QMaxSat [Koshimura *et al.*, 2012], Open-WBO [Martins *et al.*, 2014], and Pacose [Paxian and Becker, 2024]. In contrast, core-guided and MHS-guided solvers treat the MaxSAT problem as the SAT instance and use a CDCL SAT solver to identify unsatisfiable cores. Core-guided solvers, after identifying an unsatisfiable core, learn new clauses from the core and add them to the original instance, thereby increasing the lower bound. SAT solving is then applied to the updated instance, and this process is repeated until a satisfiable solution is found. Representative core-guided solvers include UWrMaxSAT [Piotrów, 2020], CGSS2 [Ihalainen *et al.*, 2024], and EvalMaxSAT [Berg *et al.*, 2024]. For MHS-guided solvers, each time an unsatisfiable core is identified, the set of clauses to be passed to the SAT solver in the next round is updated. The goal of this update is to minimize the number of clauses not passed to the SAT solver by computing the MHS of all known unsatisfiable cores. This process is repeated iteratively until a satisfiable solution is found. Typically, mixed integer programming (MIP) solvers are used to solve the MHS. Notable MHS-guided solvers include LMHS [Saikko *et al.*, 2016] and MaxHS [Bacchus, 2022].

In this paper, we propose a core-guided MaxSAT solver, built upon the UWrMaxSAT solver [Piotrów, 2024], with two key improvements. The first and most important improvement is a novel extended stratified strategy that guides the algorithm to focus primarily on solving high-weight soft clauses. This strategy is applied adaptively, based on the specific characteristics of the instance, thereby improving overall efficiency. The second improvement is a stratified-based method for extracting multiple disjoint unsatisfiable cores in each iteration, which helps accelerate the solving process.

By incorporating these two ideas, we developed an enhanced MaxSAT solver, CASHWMaxSAT, which was submitted to the MaxSAT Evaluation 2024[1]. Extensive experiments are conducted to evaluate CASHWMaxSAT on the complete weighted benchmarks of the MaxSAT Evaluations from 2022 to 2024. The experimental results demonstrate that CASHWMaxSAT outperforms eight state-of-the-art MaxSAT exact solvers across all benchmarks. Furthermore, our analysis reveals that the proposed strategies significantly contribute to the outstanding performance of CASHWMaxSAT.

This paper is organized as follows: Section 2 provides the preliminaries. Sections 3 and 4 describe the two proposed strategies. Section 5 reviews the framework of our proposed MaxSAT solver. Section 6 presents an empirical evaluation and analysis of our solver. Section 7 concludes the paper.

2 Preliminaries

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables. A literal is either a variable x or its negation $\neg x$. A clause is a disjunction of k literals, i.e., $c_j = l_{j1} \vee l_{j2} \vee \cdots \vee l_{jk}$, where l_{ji} represents the literal in the clause and $|c_j| = k$ denotes the length of the clause. A clause is called a unit clause if it contains exactly one literal. A propositional formula in conjunctive normal form (CNF) is a conjunction of m clauses, i.e., $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, where $|F| = m$ represents the number of clauses in the formula. A truth assignment ρ assigns each variable in V a truth value from $\{0, 1\}$, where 0 represents false and 1 represents true. A literal x is satisfied by a truth assignment ρ if $x = 1$, and a literal $\neg x$ is satisfied if $x = 0$. A clause is satisfied if at least one of its literals is satisfied, and a CNF formula is satisfied if all its clauses are satisfied.

The Boolean satisfiability (SAT) problem is to determine whether there exists a truth assignment that satisfies a given CNF formula F. A SAT formula is considered satisfiable if there exists a truth assignment that satisfies all its clauses; otherwise, it is deemed unsatisfiable. For an unsatisfiable SAT formula F, an unsatisfiable core ($core$) is a subset of the clauses in F that is itself unsatisfiable. A minimal unsatisfiable core is an unsatisfiable core with the additional property that removing any clause from it results in a satisfiable set of clauses. An optimized version of SAT, known as MaxSAT, seeks to determine whether there exists a truth assignment that satisfies the maximum number of clauses in F. In partial MaxSAT, the clauses are divided into hard clauses F_{hard} and soft clauses F_{soft}, and the goal is to find a truth assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses. In weighted partial MaxSAT (WPMS), each soft clause $c \in F_{soft}$ is assigned a weight $w(c)$ to reflect its importance, and the objective is to find a truth assignment that satisfies all the hard clauses while maximizing the total weight of the satisfied soft clauses. Given a soft clause set F', we use $maxw(F')$, $minw(F')$, and $avgw(F')$ to represent the maximum, minimum, and average weight values of all soft clauses in F', respectively.

In the process of solving MaxSAT, the input MaxSAT instance is first normalized. Given a CNF formula $F = F_{hard} \cup F_{soft}$, for each non-unit soft clause $c = l_1 \vee l_2 \vee \cdots \vee l_l$ (i.e., $|c| > 1$), we introduce a relaxation variable r_i to transform the clause into a unit soft clause. Specifically, we add a literal $\neg r_i$ to the clause c, resulting in a new clause $c_{new} = l_1 \vee l_2 \vee \cdots \vee l_l \vee \neg r_i$. This new clause is treated as a hard clause and added to F_{hard}. The original clause c is then removed from F_{soft}, and a unit clause r_i, whose weight equals $w(c)$, is added to F_{soft}. If the relaxation variable r_i is assigned true, it indicates that the original clause c is satisfied; otherwise, c is not satisfied. During the search process, we will maintain a lower bound (LB) and upper bound (UB) of the total weight of unsatisfied soft clauses, and the assumption set ($assump$) is defined as the set of unit soft clauses that must be satisfied under the current assignment during a SAT solver call.

Core-guided MaxSAT solver typically starts by adding all unit soft clauses in F_{soft} to the assumption set $assump$. The solver then iteratively calls a SAT solver on the working formula $F_{hard} \cup assump$. The SAT solver will say whether the formula is satisfiable or not, and in case the formula is unsatisfiable, it will give an unsatisfiable core $core$. At this point the algorithm will produce new relax variables, one for each clause in $core$. The working formula will consist in adding

[1]https://maxsat-evaluations.github.io/2024/

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

the new variables to the formulas of the $core$, adding a cardinality constraint saying that exactly one of the new variables should be true, and increasing LB by 1. This procedure is applied until the SAT solver returns satisfiable.

3 Extended Stratified Strategy for WPMS

Many previous MaxSAT solvers, such as MaxHS [Bacchus, 2022], start with the initial assumption that all soft clauses must be satisfied and pass this assumption directly to the SAT solver for processing. However, this approach has a significant drawback: the SAT solver must handle all soft clauses from the beginning, regardless of their varying weights. As a result, the solver may spend substantial time on low-weight soft clauses, diverting focus from high-weight clauses that are more critical.

To tackle this issue, Ansótegui et al. [2012] proposed the stratified strategy. The key idea behind this strategy is to limit the clauses passed to the SAT solver, prioritizing high-weight soft clauses first.

3.1 Traditional Stratified Strategies

The stratified strategy proposed by [Ansótegui *et al.*, 2012] optimizes the solving process by dividing the soft clauses into multiple layers based on their weights and solving them layer by layer. The minimum weight value for the current layer (denoted as $assump\_weight$ in our work) is set to be the maximum weight among all unprocessed soft clauses. These unprocessed soft clauses include the remaining original soft clauses (denoted as F'_s) and the soft clauses whose weights have been decreased to below $assump\_weight$ during the handling of unsatisfiable cores (denoted as $delay$). In detail, when processing a current layer, all soft clauses with weight values greater than or equal to $assump\_weight$ should be removed from both F'_s and $delay$, and added to the assumption set $assump$ for further processing. In addition, if any clause $c \in assump$ is later found to have $w(c) < assump\_weight$, it should be removed from $assump$ and added to $delay$ for deferred handling.

The UWrMaxSAT algorithm improves upon the stratified strategy by introducing two key enhancements [Piotrów, 2020]. First, the algorithm separately evaluates F'_s and $delay$ and selects the candidate set with the larger maximum weight ($maxw$) between the two. If both sets have the same $maxw$, $delay$ is prioritized as the candidate set. Second, the algorithm refines the determination of $assump\_weight$ based on the selected candidate set. When the candidate set is F'_s, $assump\_weight$ is set to the second-highest weight if the difference between the highest and second-highest weights is only 1; otherwise, it is set to the highest weight. When the candidate set is $delay$, $assump\_weight$ is always set to the highest weight.

3.2 Details of Extended Stratified Strategy

In this subsection, we first analyze the drawbacks of traditional stratified strategies and then introduce our extended stratified strategy.

The traditional stratified strategies described above adopt a fine-grained approach to stratification, where the $assump\_weight$ value for each layer is set close to the maximum weight value among all unprocessed soft clauses. However, if the assumption sets for several successive layers (i.e., different unprocessed soft clauses) are set to true individually, and the SAT solver repeatedly returns true without generating any unsatisfiable core—particularly in cases where satisfying the hard clauses is inherently challenging—this fine-grained approach can significantly slow down the solving process. A potential improvement is to merge multiple successive layers into a single layer, allowing the SAT solver to handle them collectively in one step. On the other hand, if the $assump\_weight$ value of the current layer is set significantly lower than the maximum weight among all unprocessed soft clauses, the approach becomes coarse-grained. This results in excessive processing within a single layer, thereby reducing the algorithm's ability to prioritize high-weight soft clauses effectively.

To strike a balance between fine-grained and coarse-grained approaches, our method sets $assump\_weight$ to $\lfloor maxw(F'_s)/2 \rfloor + 1$ when handling F'_s. The rationale for this value is explained below, followed by an example to provide a more concrete illustration. With our method, each soft clause is processed at most once in the current layer. Specifically, once a soft clause appears in an unsatisfiable core and is processed, it is removed from the current layer and added to $delay$ for subsequent handling. Additionally, to prevent the top layer from becoming overloaded with too many soft clauses, we compare $\lfloor maxw(F'_s)/2 \rfloor + 1$ with $avgw(F'_s)$. The larger of these two values is selected as the $assump\_weight$ for the top layer.

Furthermore, we analyze the triggering condition for adopting our proposed extended stratified strategy. Preliminary experiments revealed that for certain instances, the algorithm performs better without using the stratified strategy. Upon analyzing the structural features of these instances, we observed that they often exhibit similar weight distributions, characterized by a ratio of the standard deviation to the average weight (denoted as *coeff*) smaller than 1. We hypothesize that when the weight distribution of a problem is highly dispersed, it is more effective to process the problem layer by layer. Based on this observation, our algorithm applies the stratified strategy only if $coeff \geq 1$; otherwise, the stratified strategy is bypassed.

Example 1. Consider an example where $maxw(F'_s)$ is 9, and let c be a unit soft clause with the maximum weight value in F'_s. According to our approach, we set $assump\_weight = \lfloor maxw(F'_s)/2 \rfloor + 1 = 5$. If c appears in an unsatisfiable core, then after processing the core, $w(c)$ becomes at most 4. Because $w(c)$ is now smaller than $assump\_weight$, the clause c is removed from the current layer.

Based on the above considerations, we propose an extended stratification strategy, outlined in Algorithm 1. First, the algorithm determines whether the stratified strategy should be applied. If the stratified strategy is not used, all soft clauses are returned directly (Line 1). Otherwise, the algorithm calculates the $assump\_weight$ value for the top layer based on the larger of $avgw(F'_s)$ and $(\lfloor maxw(F'_s)/2 \rfloor + 1)$, moves all satisfied soft clauses from F'_s to $assump$, and then

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Algorithm 1: Stratification

Input: the remaining soft clauses F'_s
Output: the modified assumption set $assump$

1 **if** *coeff* < 1 **then** $mark := -1$ and **return** F'_s ;
2 **if** *mark = 0* **then**
3 **if** $avgw(F'_s) > \lfloor maxw(F'_s)/2 \rfloor + 1$ **then** $assump\_weight := \lfloor avgw(F'_s) \rfloor$;
4 **else** $assump\_weight := \lfloor maxw(F'_s)/2 \rfloor + 1$;
5 $mark := 1$;
6 **for** *each* $c_i \in F'_s$ **do**
7 **if** $w(c_i) \geq assump\_weight$ **then** $F'_s := F'_s \setminus \{c_i\}$, $assump := assump \cup \{c_i\}$;
8 **return** $assump$;
9 **if** $state = SAT$ **then**
10 **if** $delay \neq \emptyset$ *and* $maxw(F'_s) \leq maxw(delay)$ **then**
11 $assump\_weight := maxw(delay)$;
12 **for** *each* $c_i \in delay$ **do**
13 **if** $w(c_i) \geq assump\_weight$ **then**
14 $delay := delay \setminus \{c_i\}$;
15 $assump := assump \cup \{c_i\}$;
16 **else**
17 $assump\_weight := \lfloor maxw(F'_s)/2 \rfloor + 1$;
18 **for** *each* $c_i \in F'_s$ **do**
19 **if** $w(c_i) \geq assump\_weight$ **then**
20 $F'_s := F'_s \setminus \{c_i\}$;
21 $assump = assump \cup \{c_i\}$;
22 **else if** $state = UNSAT$ **then**
23 **for** *each* $c_i \in assump$ **do**
24 **if** $w(c_i) < assump\_weight$ **then**
25 $assump := assump \setminus \{c_i\}$;
26 $delay := delay \cup \{c_i\}$;
27 **else if** $state = UNKNOWN$ **then**
28 **for** *each* $c_i \in assump$ **do**
29 **if** $w(c_i) = assump\_weight$ **then**
30 $assump := assump \setminus \{c_i\}$;
31 $delay := delay \cup \{c_i\}$;
32 $assump\_weight := minw(assump)$;
33 **return** $assump$;

returns the updated $assump$ (Lines 2–8). During the subsequent stratified process, if the SAT solver returns SAT, the algorithm selects a candidate set (either $delay$ or F'_s) for further processing (Lines 9–10). If $delay$ is selected, $assump\_weight$ is set to $maxw(delay)$ (Line 11); otherwise, it is set to $\lfloor maxw(F'_s)/2 \rfloor + 1$ (Line 17). The corresponding sets are then updated accordingly (Lines 12–15 and Lines 18–21). If the SAT solver returns *UNSAT*, all soft clauses in $assump$ with weights less than $assump\_weight$ are removed and added to $delay$ (Lines 22–26). If the SAT solver returns *UNKNOWN*, the algorithm removes the soft clauses in $assump$ with the smallest weight, adds them to $delay$, and updates $assump\_weight$ to $minw(assump)$ (Lines 27–32). Finally, $assump$ is returned (Line 33).

Algorithm 2: MUS

Input: CNF formula F_{hard}, an unsatisfiable core $core$
Output: the minimal unsatisfiable core $core$

1 **foreach** $c_i \in core$ **do**
2 $core := core \setminus \{c_i\}$;
3 $state := SATSolver(F_{hard} \cup core)$;
4 **if** $state \neq UNSAT$ **then** $core := core \cup \{c_i\}$;
5 **return** $core$;

4 Stratified-Based Multi-Unsatisfiable Cores Extraction Method for WPMS

Several existing works focus on extracting multiple unsatisfiable cores to improve MaxSAT solving efficiency. For instance, in the first stage of the MaxHS algorithm [Davies and Bacchus, 2011], identifying as many disjoint unsatisfiable cores as possible significantly enhances the algorithm's performance. Furthermore, Saikko [2015] demonstrates that extracting multiple disjoint unsatisfiable cores in each iteration can improve the efficiency of MHS-guided MaxSAT solvers. Building on this idea, Berg et al. [2017] introduces a weight-aware core extraction technique for the WPMS, which leverages disjoint unsatisfiable cores. The authors [2017] propose that, instead of requiring all unsatisfiable cores within the same disjoint phase to be strictly disjoint, soft clauses with higher weight values may be shared among multiple cores. This approach aims to increase the number of unsatisfiable cores extracted during the same disjoint phase.

Given these advancements, a natural question arises: can our proposed extended stratified strategy be combined with the multi-unsatisfiable cores extraction method? To explore this, we adapt the existing unsatisfiable core extraction techniques to align with our extended stratified strategy, resulting in a novel stratified-based multi-unsatisfiable cores extraction (MUCE) method.

4.1 Extraction of Minimal Unsatisfiability Core

To extract multiple unsatisfiable cores, the first step is to obtain a single unsatisfiable core. In the context of MaxSAT, an unsatisfiable core is a set of clauses from soft clauses that cannot be satisfied simultaneously, regardless of the truth assignments to the involved soft clauses. This subset inherently contains a conflict that prevents any possible satisfying assignment. Our proposed MaxSAT solver relies on a SAT solver to extract unsatisfiable cores. When an unsatisfiable core is obtained, at least one clause within the core is guaranteed to be unsatisfiable, enabling the derivation of a cardinality constraint. To generate these constraints, we adopt the OLL procedure [Andres *et al.*, 2012; Morgado *et al.*, 2014; Ignatiev *et al.*, 2019] and convert the resulting cardinality constraints into clauses using the encoding approach outlined in UWrMaxSAT [Piotrów, 2020].

However, the unsatisfiable core extracted by the SAT solver is typically not minimal, meaning it may include redundant clauses that are not necessary for its unsatisfiability. As highlighted by EvalMaxSAT [Avellaneda, 2020], smaller unsatisfiable cores lead to better performance in the OLL procedure. To address this, we apply a core minimization technique pro-

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Algorithm 3: MUCE

Input: CNF formula $F_{hard} \cup assump$, lower bound LB, an unsatisfiable core $core$, minimum weight value $assump\_weight$ for the current layer
Output: lower bound LB, hard clauses F_{hard}, the assumption set $assump$

1 $assump_t := assump$;
2 $w_t(c) := w(c)$, for each $c \in assump_t$;
3 $cores := core_m := \emptyset$;
4 **while** $assump_t \neq \emptyset$ **do**
5 $\quad core_m := MUS(F_{hard}, core)$;
6 $\quad LB := LB + minw(core_m)$;
7 $\quad cores := cores \cup \{core_m\}$;
8 $\quad$ **foreach** $c_i \in assump_t$ **do**
9 $\quad\quad$ **if** $c_i \in core_m$ **then**
10 $\quad\quad\quad w_t(c_i) := w_t(c_i) - minw(core_m)$;
11 $\quad\quad\quad$ **if** $w_t(c_i) < assump\_weight$ **then**
12 $\quad\quad\quad\quad assump_t := assump_t \setminus \{c_i\}$;
13 $\quad (state, \rho, core) := SATSolver(F_{hard} \cup assump_t)$;
14 $\quad$ **if** $state \neq$ *UNSAT* **then break**;
15 **for** $core_m \in cores$ **do**
16 $\quad (F_{hard}, assump) := OLL(core_m, F_{hard}, assump)$;
17 **return** $(LB, F_{hard}, assump)$

posed by Marques-Silva et al. [2010] after extracting the unsatisfiable core. This step removes redundant clauses, resulting in a minimal unsatisfiable core (MUS).

Algorithm 2 illustrates the process of minimizing the unsatisfiable core. The algorithm iterates through all the unit soft clause c_i in the core and removes each soft clause from the core (Line 2). Then, we call the SAT solver to check the satisfiability of the remaining core, which also includes the hard clauses (Line 3). If the solver returns *UNSAT*, this means c_i is a redundant soft clause in the core and should be deleted. Otherwise, c_i is an essential part of the unsatisfiable core, and the algorithm adds it back to the core (Line 4). Finally, the obtained minimal unsatisfiable core is returned (Line 5).

4.2 Details of the MUCE Method

The MUCE method, presented in Algorithm 3, begins by initializing a temporary set, $assump_t$, to store the assumption set $assump$ (Line 1), and an array, w_t, to store the weight values of each unit soft clause c in $assump_t$ (Line 2). Two core sets, $cores$ and $core_m$, are also initialized as empty sets (Line 3). The algorithm proceeds to enumerate the minimal unsatisfiable cores until $assump_t$ becomes empty (Line 4). After minimizing the current unsatisfiable core (Line 5), the algorithm updates the lower bound by the minimum weight value of the current minimal unsatisfiable core, $minw(core_m)$ (Line 6), and adds the minimal core $core_m$ to $cores$ (Line 7). For each unit soft clause in $assump_t$, if it exists in the current minimal unsatisfiable core, the algorithm reduces its weight by $minw(core_m)$ (Line 10). Because the MaxSAT algorithm uses the proposed extended stratified strategy, the weight value of every unit soft clause in $assump$ is always greater than or equal to the minimum weight of the current layer, $assump\_weight$. As a result, the algorithm removes those unsatisfied unit soft clauses from $assump_t$ (Lines 11–12), which is mainly different from previous extraction methods. Next, the algorithm calls the SAT solver to solve the formula $F_{hard} \cup assump_t$. If the result is *UNSAT*, this indicates that additional unsatisfiable cores exist in the formula; otherwise, the enumeration process terminates (Lines 13–14). Finally, for all minimal unsatisfiable cores, the algorithm converts them into corresponding clauses using the OLL procedure and adds them to F_{hard} and $assump$ (Lines 15–16). The algorithm then returns the lower bound, LB, along with the updated F_{hard} and $assump$ (Line 17).

5 The Framework for CASHWMaxSAT

Algorithm 4 presents the framework of our proposed CASHWMaxSAT. We first introduce the key variables and functions used in our proposed algorithm.

The remaining set of soft clauses is expressed as F'_s. To facilitate our stratification method, we use $assump\_weight$ to record the minimum weight value in the current layer under the proposed stratified strategy, i.e., all soft clauses whose weight is not smaller than $assump\_weight$ in F'_s being added into the assumption set $assump$ as the current layer. The tag variable $mark$ can take three possible values: *-1* indicates that the algorithm does not use the proposed stratified strategy; *0* indicates that the algorithm is using the proposed stratified strategy for the first time; and *1* indicates that the proposed stratified strategy has already been applied more than once.

The current assignment is denoted as ρ, while the best assignment found so far is represented as ρ^*. The function $cost(F, \rho)$ computes the total weight of unsatisfied soft clauses under the assignment ρ. Each SAT solver call returns a triple $(state, \rho, core)$, where $state$ represents the solver's status: satisfiable (*SAT*), unsatisfiable (*UNSAT*), or unknown (*UNKNOWN*). If the solver returns *SAT*, ρ is the current assignment that satisfies $F_{hard} \cup assump$. If the solver returns *UNSAT*, it indicates no assignments can satisfy all assumptions, and $core$ contains a subset of soft clauses from $F_{hard} \cup assump$ that form an unsatisfiable core. If the solver returns *UNKNOWN*, it means the solver fails to resolve the current formula. In our work, the termination condition of the SAT solver is that when the number of conflicts generated during the search reaches a certain threshold, the solver returns *UNKNOWN*.

The input CNF formula F is normalized, meaning that each soft clause is a unit soft clause. The algorithm begins by initializing necessary variables (Lines 1–2). It then applies an extended stratification strategy to organize the soft clauses based on their weights and returns the minimum weight of the top layer as $assump\_weight$ (Line 3).

The main iterations of the algorithm are carried out in Lines 4–13. If UB equals LB, it indicates that the optimal solution has been found, and the loop terminates (Lines 9 and 12). In each iteration, the SAT solver is invoked on the current formula, which consists of all hard clauses and the assumption set $assump$. This process essentially sets all unit soft clauses (treated as literals) in $assump$ to true. The SAT solver then returns its result, which is stored in the variable

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Algorithm 4: CASHWMaxSAT

Input: CNF formula $F = F_{hard} \cup F_{soft}$
Output: upper bound UB, truth assignment ρ^*

1 $assump := delay := \rho^* := \rho := \emptyset$;
2 $UB := \sum_{c \in F_{soft}} w(c)$, $mark := LB := 0$, $F'_s := F_{soft}$;
/* Extended Stratified Strategy */
3 $assump :=$ *Stratification*(F'_s);
4 **while** *true* **do**
5 $(state, \rho, core) := SATSolver(F_{hard} \cup assump)$;
6 **if** $state =$ *SAT* **then**
7 **if** $UB > cost(F, \rho)$ **then**
8 $UB := cost(F, \rho)$, $\rho^* := \rho$;
9 **if** $UB = LB$ **then return** (UB, ρ^*);
10 **else if** $state =$ *UNSAT* **then**
/* Cores Extraction Method */
11 $(LB, F_{hard}, assump) :=$ *MUCE*$(F_{hard} \cup assump, LB, core, minw(assump))$;
12 **if** $UB = LB$ **then return** (UB, ρ^*);
/* Extended Stratified Strategy */
13 **if** $mark \neq -1$ **then** $assump :=$ *Stratification*(F'_s) ;

$state$ (Line 5).

If the SAT solver returns *SAT*, the algorithm retrieves a truth assignment from the solver (Line 6). If UB is improved, the algorithm updates UB and ρ^* accordingly (Lines 7–8). If the SAT solver returns *UNSAT*, indicating a conflict within $assump$, the proposed *MUCE* method is employed to enumerate disjoint unsatisfiable cores (Lines 11–12). This method incorporates unsatisfiable core minimization techniques to generate a high-quality set of disjoint unsatisfiable cores. The *MUCE* method returns updated values for LB, F_{hard}, and $assump$. At the end of each iteration, the algorithm calls the proposed stratified strategy to update $assump$ accordingly (Line 13).

In our proposed algorithm, we also use some additional existing techniques to further improve the performance of our algorithm, including the harden strategy [Ansótegui *et al.*, 2013] to transform soft clauses with relatively large weights into hard unit clauses based on the current values of lower and upper bounds whenever the value of lower or upper bound is changed, the mixed strategy [Piotrów, 2020] to switch from core-guided linear search to binary search if the predefined time limit is exceeded, the preprocessing strategy [Ignatiev *et al.*, 2019] to preprocess the literals of $assump$ to detect unit cores and at-most-one cores after using the stratified strategy, and a general Boolean multi-level optimization technique [Paxian *et al.*, 2021] to detect weighted instances that encode multi-objective problems with a lexicographic optimality criterion, optimizing the search procedure accordingly.

6 Computational Experiments

We conducted an experimental investigation to evaluate the performance of the proposed CASHWMaxSAT solver. Developed based on the COMiniSatPS SAT solver [Oh, 2016], the CASHWMaxSAT was implemented in C++11 with -O3 optimization to ensure high performance. All experiments were conducted on a system running Ubuntu 22.04.5 LTS, equipped with an Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz and 512GB of memory.

We selected all test instances from the complete weighted track of the MaxSAT Evaluation (MSE) 2022, 2023, and 2024[2], resulting in a total of 1,736 instances. These instances span a wide range of domains, including combinatorial optimization, artificial intelligence, circuit design, software engineering, and bioinformatics, ensuring both diversity and representativeness for a comprehensive evaluation of algorithm performance. For each test instance and algorithm, the runtime was set to 3,600 seconds, aligning with the competition's time limit.

To ensure a fair comparison, we excluded all hybrid solvers that rely on other independent solvers and focused solely on the latest versions of independent MaxSAT exact solvers that participated in the complete weighted track of the MSEs from MSE2022 to MSE2024, excluding our algorithm. This resulted in the selection of seven solvers: Exact [Devriendt, 2024], Open-WBO [Martins *et al.*, 2023], Pacose [Paxian and Becker, 2024], CGSS2 [Ihalainen *et al.*, 2024], WMaxCDCL [Li *et al.*, 2024], UwrMaxSAT [Piotrów, 2024], and EvalMaxSAT [Berg *et al.*, 2024]. Additionally, we included SCIP (version 8.1.0) [Bestuzheva *et al.*, 2021], a constraint integer programming solver often used in combination with other MaxSAT solvers as an independent solver in recent MSEs, as a competitor. In total, we compared our method against eight state-of-the-art MaxSAT solvers.

Furthermore, many algorithms in recent MSEs have adopted hybrid approaches by combining multiple solvers to improve performance. In this context, we also compare the hybrid version of our algorithm with five state-of-the-art hybrid MaxSAT solvers from the complete weighted track of the MSEs: WMaxCDCL+Open-WBO [Li *et al.*, 2024], MaxHS [Bacchus, 2022], UwrMaxSAT+SCIP [Piotrów, 2024], WMaxCDCL+SCIP+MaxHS [Coll *et al.*, 2023], and EvalMaxSAT+SCIP [Berg *et al.*, 2024]. It is important to note that MaxHS, in addition to being a MaxSAT solver based on a SAT solver, also utilizes the commercial linear programming solver CPLEX[3] for solving the hitting set problem, which classifies MaxHS as a hybrid algorithm. In the WMaxCDCL+SCIP+MaxHS configuration, the SCIP is called by using the competition version of UwrMaxSAT+SCIP, but this version does not execute SCIP for certain large instances and instead runs UwrMaxSAT. As a result, this hybrid algorithm essentially combines four distinct solvers. Among these solvers, some provide multiple parameter configurations in recent MSEs, and for consistency, we selected the best-performing configuration for each algorithm (including our own competition version) for comparison.

6.1 Overall Results

Independent Algorithm Comparison. Table 1 presents a comparison of all algorithms across three sets of instances. #solve represents the number of instances solved, while avg denotes the average time in seconds required to solve these

[2]https://maxsat-evaluations.github.io/
[3]https://www.ibm.com/products/ilog-cplex-optimization-studio

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Solver	MSE22 (607)			MSE23 (558)			MSE24 (571)		
	#solve	avg	PAR2	#solve	avg	PAR2	#solve	avg	PAR2
SCIP	253	317.34	4331.28	264	208.75	3899.30	261	227.16	4019.80
Exact	283	190.43	3931.95	330	328.39	3141.73	317	242.27	3343.12
Open-WBO	334	320.61	3414.63	363	163.37	2626.92	352	162.71	2866.60
Pacose	364	199.19	3001.82	382	140.12	2371.13	384	197.48	2495.14
WMaxCDCL	412	256.78	2487.31	387	140.64	2308.12	398	204.37	2327.96
CGSS2	410	224.23	2488.20	412	141.13	1991.64	415	186.41	2106.25
EvalMaxSAT	412	423.66	2600.57	418	309.66	2041.69	419	300.81	2140.73
UwrMaxSAT	408	206.65	2499.36	418	140.3	1914.98	425	167.69	1969.24
CASHWMaxSAT	**423**	216.47	**2333.39**	**431**	156.64	**1759.70**	**438**	156.64	**1822.78**

Table 1: Results for all the instances of the complete weighted track of the MSE 22–24 within CASHWMaxSAT and 8 top independent solvers.

Solver	MSE22	MSE23	MSE24
WMaxCDCL+Open-WBO	418	409	414
MaxHS	428	425	443
EvalMaxSAT+SCIP	435	434	433
UwrMaxSAT+SCIP	434	435	442
WMaxCDCL+SCIP+MaxHS	**444**	435	445
CASHWMaxSAT+SCIP	443	**445**	**448**

Table 2: Results for the number of solved instances of all the benchmarks within CASHWMaxSAT and 5 top hybrid solvers.

Solver	MSE22	MSE23	MSE24
CASHWMaxSAT\MUST	405	418	417
CASHWMaxSAT+TSTR	408	412	423
CASHWMaxSAT\STRA	418	418	429
CASHWMaxSAT\MUCE	413	421	426
CASHWMaxSAT	**423**	**431**	**438**

Table 3: Results for the number of solved instances of all the benchmarks within CASHWMaxSAT and its variants.

instances. PAR2 refers to the penalized average runtime [Hutter *et al.*, 2009], with a penalization constant of 2, which is used for the first time in the MSE2024. The results indicate that our solver not only achieves the highest number of optimal solutions across all benchmark tests but also outperforms the second-best solver by more than 10 instances solved to optimality per benchmark set, while maintaining significantly lower average solving times. Furthermore, in terms of the PAR2 score, our algorithm clearly demonstrates an advantage, with at least 100 fewer penalized average runtime compared to the other algorithms, underscoring its superior performance in both solution quality and solving speed. Additionally, Figure 1 illustrates the increase in the number of solutions processed by all algorithms over time, with our algorithm consistently outperforming the others at every time interval, further emphasizing its efficiency.

Hybrid Algorithm Comparison. Table 2 presents a comparison of all hybrid algorithms, showing that our algorithm performs slightly worse than WMaxCDCL+SCIP+MaxHS on the MSE22 benchmark, where it achieved two additional optimal solutions. However, on the other two benchmarks, our algorithm outperforms all the others, clearly demonstrating its superior performance. As shown in Tables 1 and 2, our solver with SCIP solves 44 more instances than the version without SCIP. In comparison, EvalMaxSAT and UWrMaxSAT with SCIP solve 60 and 53 more instances, respectively, than their versions without SCIP. These results suggest that while incorporating SCIP into our method does lead to improvements, the gain is less significant compared to its contribution in other solvers. Additionally, our proposed CASHWMaxSAT utilizes the best-performing configuration of WMaxCDCL+SCIP+MaxHS, resulting in a new solver, CASHWMaxSAT+SCIP+MaxHS. The results show that our new algorithm achieves 449 optimal solutions, which represents the best performance on the MSE 22 instances. Due to the lower server configuration used in our experiments compared to those in MSE 23, our results are slightly less favorable than those reported in MSE 23. To address this, we extended the time limit for the top three performing solvers to 7200 seconds and re-ran these algorithms on all the instances from MSE 23. The results show that CASHWMaxSAT+SCIP, WMaxCDCL+SCIP+MaxHS, EvalMaxSAT+SCIP, and UwrMaxSAT+SCIP achieved 451, 446, 442, and 439 optimal solutions, respectively.

6.2 Evaluating Two Strategies

We conducted an effectiveness analysis on two strategies of our algorithm: the extended stratified strategy and the MUCE method. Table 3 compares CASHWMaxSAT with the following variants: 1) CASHWMaxSAT\STRA. CASHWMaxSAT ignores the proposed stratified strategy; 2) CASHWMaxSAT+TSTR. CASHWMaxSAT uses a traditional stratified strategy, as implemented in UWrMaxSAT [Piotrów, 2020], instead of the proposed stratified strategy; 3) CASHWMaxSAT\MUCE. CASHWMaxSAT ignores the MUCE; 4) CASHWMaxSAT\MUST. CASHWMaxSAT ignores the MUCE and the proposed stratified strategy.

These results highlight the impact of stratified strategy and MUCE on the algorithm's performance under various conditions. From the comparison with CASHWMaxSAT\MUST, our algorithm achieves 17.3 more instances solved to optimality per benchmark set compared to this, it is evident that the strategies we designed are highly effective in improving the solving performance. Moreover, the effectiveness of the additional techniques, including the hardening strategy, preprocessing strategy, BMO technique, mixed strategy, and others, can be assessed through the results of CASHWMaxSAT\MUST presented in Table 3.

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

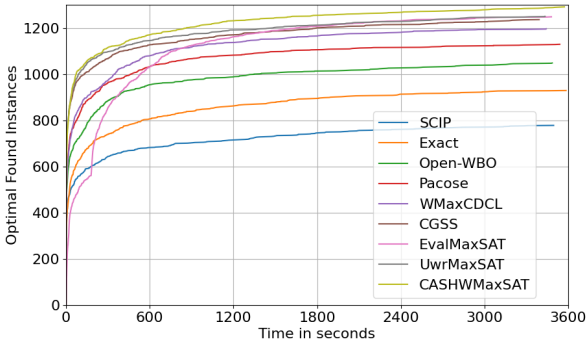

Figure 1: The cumulative distribution functions of state-of-the-art MaxSAT solvers for all the benchmarks.

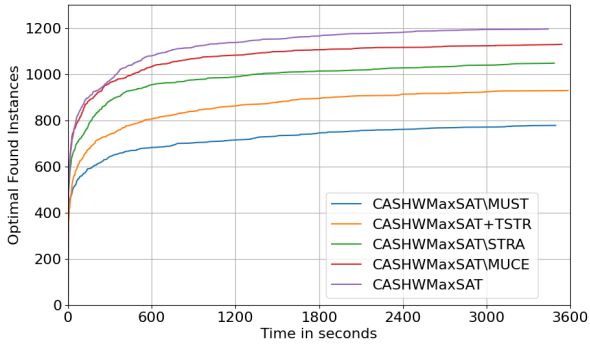

Figure 2: The cumulative distribution functions of CASHWMaxSAT and its five modified versions for all the benchmarks.

CASHWMaxSAT\MUST performs worse than both EvalMaxSAT and UwrMaxSAT, suggesting that the application of additional techniques alone is insufficient. A significant performance gain can only be achieved when these techniques are combined with the two main extensions.

The number of optimal solutions found by CASHWMaxSAT+TSTR is, on average, 7.3 fewer per benchmark compared to CASHWMaxSAT\STRA, indicating that the previous stratified strategy did not integrate well with our designed MUCE strategy. On the other hand, CASHWMaxSAT+TSTR has, on average, 16.3 fewer instances solved to optimality per benchmark set than our algorithm, further highlighting that the new stratified strategy we designed integrates exceptionally well with the MUCE strategy, leading to a significant improvement in performance. The number of optimal solutions obtained by CASHWMaxSAT\MUCE and CASHWMaxSAT\STRA both show a noticeable gap compared to our algorithm, especially in the MSE23 benchmark, where the difference is most apparent. This clearly demonstrates that both strategies are indispensable. In contrast, our algorithm outperforms both, demonstrating that our designed stratified strategy integrates more effectively with the MUCE strategy, leading to a significant improvement in the solving performance. Additionally, Figure 2 illustrates the number of optimal solutions obtained by each algorithm over time, offering a more intuitive comparison of the effectiveness of our algorithm.

7 Conclusions and Future Work

In this paper, we present CASHWMaxSAT, an efficient unsatisfiable core-guided MaxSAT solver that leverages an extended stratified strategy and a stratified-based multi-unsatisfiable core extraction method. Our experiments demonstrate the excellent performance of our solver.

As future work, while the MaxSAT community has traditionally focused on maximizing the number of optimal solutions, the introduction of the PAR2 scoring evaluation in MSE 24 has highlighted the importance of minimizing runtime. In addition, when combined with SCIP, the solving time tends to be longer. This suggests that future work should involve adjusting the model inputs to SCIP in order to improve performance.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments. This work is supported by National Cryptologic Science Fund of China 2025NCSF02046, NSFC under Grant No. 61806050, Jilin Science and Technology Department YDZJ202201ZYTS412.

References

[Abramé and Habet, 2014] 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.

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *ICLP*, 2012.

[Ansótegui and Gabàs, 2017] Carlos Ansótegui and Joel Gabàs. Wpm3: an (in) complete algorithm for weighted partial maxsat. *Artificial Intelligence*, 250:37–57, 2017.

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

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

[Ansótegui *et al.*, 2012] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabas, and Jordi Levy. Improving sat-based weighted maxsat solvers. In *CP*, pages 86–101, 2012.

[Ansótegui *et al.*, 2013] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[Avellaneda, 2020] Florent Avellaneda. A short description of the solver evalmaxsat. *MaxSAT Evaluation*, 8:364, 2020.

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

[Bacchus, 2022] Fahiem Bacchus. Maxhs in the 2022 maxsat evaluation. *MaxSAT Evaluation 2022*, page 17, 2022.

[Berg and Järvisalo, 2017] Jeremias Berg and Matti Järvisalo. Weight-aware core extraction in sat-based maxsat solving. In *CP*, pages 652–670, 2017.

[Berg *et al.*, 2019] Otto Jeremias Berg, Antti Juhani Hyttinen, and Matti Juhani Järvisalo. Applications of maxsat in data analysis. In *SAT*, pages 50–64, 2019.

[Berg *et al.*, 2024] Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. Maxsat evaluation 2024: Solver and benchmark descriptions. *MaxSAT Evaluation 2024*, 2024.

[Bestuzheva *et al.*, 2021] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The scip optimization suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[Coll *et al.*, 2023] Jordi Coll, Shuolin Li, Chu-Min Li, Felip Manyá, Djamal Habet, Mohamed Sami Cherif, and Kun He. Wmaxcdcl in maxsat evaluation 2023. *MaxSAT Evaluation 2023*, pages 16–17, 2023.

[Cook, 1971] Stephen A Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *CP*, pages 225–239, 2011.

[Devriendt, 2024] Jo Devriendt. Exact: evaluating a pseudo-boolean solver on maxsat problems (2024). *MaxSAT Evaluation 2024*, pages 11–12, 2024.

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

[Heras and Larrosa, 2006] Federico Heras and Javier Larrosa. New inference rules for efficient max-sat solving. In *AAAI*, pages 68–73, 2006.

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

[Hutter *et al.*, 2009] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of artificial intelligence research*, 36:267–306, 2009.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, António Morgado, and João Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019.

[Ihalainen *et al.*, 2024] Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Cgss2 and cgss2-abstcg in the 2024 maxsat evaluation. *MaxSAT Evaluation 2024*, page 13, 2024.

[Kautz *et al.*, 1992] Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *ECAI*, volume 92, pages 359–363, 1992.

[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*, 8(1-2):95–100, 2012.

[Kügel, 2010] Adrian Kügel. Improved exact solver for the weighted max-sat problem. *Pos@ sat*, 8:15–27, 2010.

[Le Berre and Parrain, 2010] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.

[Li *et al.*, 2005] Chu Min Li, Felip Manya, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In *CP*, pages 403–414, 2005.

[Li *et al.*, 2006] Chu Min Li, Felip Manya, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *AAAI*, pages 86–91, 2006.

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

[Li *et al.*, 2009] Chu Min Li, Felip Manya, Nouredine Mohamedou, and Jordi Planes. Exploiting cycle structures in max-sat. In *SAT*, pages 467–480, 2009.

[Li *et al.*, 2021a] 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*, pages 1–21, 2021.

[Li *et al.*, 2021b] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In *CP*, pages 38–1, 2021.

[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*, page 17, 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. In *AAAI*, pages 11272–11281, 2025.

[Marques-Silva *et al.*, 2015] Joao Marques-Silva, Mikoláš Janota, Alexey Ignatiev, and António Morgado. Efficient model based diagnosis with maximum satisfiability. In *IJCAI*, pages 1966–1972, 2015.

[Marques-Silva, 2010] Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications. In *ISMVL*, pages 9–14, 2010.

[Martins *et al.*, 2014] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver. In *SAT*, pages 438–445, 2014.

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

[Martins *et al.*, 2023] Ruben Martins, Norbert Manthey, Miguel Terra-Neves, Vasco Manquinho, and Inês Lynce. Open-wbo maxsat evaluation 2023. *MaxSAT Evaluation 2023*, page 18, 2023.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *CP*, pages 564–573, 2014.

[Oh, 2016] Chanseok Oh. *Improving SAT solvers by exploiting empirical characteristics of CDCL*. New York University, 2016.

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

[Paxian *et al.*, 2021] Tobias Paxian, Pascal Raiola, and Bernd Becker. On preprocessing for weighted maxsat. In *VMCAI*, pages 556–577, 2021.

[Piotrów, 2020] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *ICTAI*, pages 132–136, 2020.

[Piotrów, 2024] Marek Piotrów. Uwrmaxsat entering the maxsat evaluation 2024. *MaxSAT Evaluation 2024*, 2024:27–28, 2024.

[Saikko *et al.*, 2016] Paul Saikko, Jeremias Berg, and Matti Järvisalo. Lmhs: a sat-ip hybrid maxsat solver. In *SAT*, pages 539–546, 2016.

[Saikko, 2015] Paul Saikko. *Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability*. PhD thesis, Master's thesis, University of Helsinki, 2015.