WMaxCDCL in Pseudo-Boolean Competition 2026

1st Jialu Zhang
Laboratoire MIS UR 4290
Université de Picardie Jules Verne
Amiens, France
jialu.zhang@u-picardie.fr

2nd Chu-Min Li*
Laboratoire MIS UR 4290
Université de Picardie Jules Verne, Amiens, France
Aix Marseille Univ, Université de Toulon
CNRS, LIS, Marseille, France
chu-min.li@u-picardie.fr

3rd Sami Cherif
Laboratoire MIS UR 4290
Université de Picardie Jules Verne,
Amiens, France
sami.cherif@u-picardie.fr

4th Shuolin Li*
Aix Marseille Univ, Université de Toulon
CNRS, LIS, Marseille, France
shuolin.li@lis-lab.fr

5th Jinghu Liang
Laboratoire MIS UR 4290
Université de Picardie Jules Verne, Amiens, France
jinghu.liang@u-picardie.fr

I. Introduction

WMaxCDCL is a Branch-and-Bound (BnB) Weighted Partial MaxSAT (WPMS) solver. Its main algorithm is based on the Conflict-Driven Clause Learning (CDCL) framework [5], [6], integrating components tailored for WPMS and novel techniques to enhance overall efficiency [3], [7].

In this system description, we introduce WMaxCDCL2026, an updated version of WMaxCDCL for the Pseudo-Boolean Competition 2026 (PB26). Building on the foundation of WMaxCDCL2024 [8], this version integrates novel techniques to improve solving performance. Furthermore, we leverage the encoding component of UWrMaxSat [10] to translate pseudo-Boolean instances into MaxSAT formulations, and we have integrated the GNU Multiple Precision Arithmetic Library (GMP[1]) into WMaxCDCL2026 to support big integer weight operations.

II. MaxSAT Encodings

Our solving pipeline operates in three main stages. First, we employ UWrMaxSat to translate a given pseudo-Boolean instance into a MaxSAT formulation. Next, WMaxCDCL2026 solves this encoded MaxSAT instance. Finally, UWrMaxSat is utilized again to map the MaxSAT solution back to a valid solution for the original pseudo-Boolean problem. For the translation phase, we rely on the default settings of UWrMaxSat's encoding module. This configuration automatically selects the most appropriate encoding mechanisms—such as Binary Decision Diagrams (BDDs) [1] or Adder Networks [4]—depending on the structural properties of each individual pseudo-Boolean constraint.

III. New Techniques in WMaxCDCL2026

A. Stratified Hardening Strategy

Inspired by stratification techniques in core-guided algorithms, WMaxCDCL2026 implements a preprocessing strategy to exploit the hierarchical structure in WPMS instances.

The strategy consists of hardening high-weight soft clauses to simplify Boolean Multilevel Optimization (BMO [9]) instances. Specifically, for BMO instances, the hardening strategy 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. If H remains satisfiable, the hardened clauses are removed from the soft clause set S. Conversely, if H becomes unsatisfiable, the algorithm restores consistency by discarding the clauses hardened in the current iteration and terminates the hardening phase. The implementation details of this strategy are provided in [11].

B. Weight-Aware Lookahead

In WMaxCDCL2024, the lookahead framework 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. As illustrated in Example 1, 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 1: Consider soft literals $S = \{l_1, l_2, l_3\}$ with weights $w(l_1) = w(l_2) = 10$, $w(l_3) = 1$, upper bound UB $= 10$, and hard clauses $H = (x_1 \vee \neg l_1 \vee \neg l_3) \wedge (\neg l_1 \vee \neg l_2 \vee l_3)$. 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 classic 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 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 $w(\kappa_2) =$

* Corresponding authors.
[1]https://gmplib.org/

$\min(rw(l_1), rw(l_2)) = 9$. 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 $LB \geq UB$.

To overcome the limitations of the classic lookahead, WMaxCDCL2026 implements a weight-aware lookahead. The key differences from the classic lookahead are as follows:

1) **Soft literal selection:** While classic lookahead selects soft literals for propagation based primarily on their recent conflict activity, weight-aware lookahead prioritizes soft literals with higher remaining weights, using activity scores strictly as a tie-breaker.
2) **Unit propagation:** Classic lookahead aborts propagation immediately upon falsifying a single soft literal. In contrast, weight-aware lookahead exhausts unit propagation, thereby allowing multiple soft literals to be falsified and evaluated.
3) **Core extraction:** Classic lookahead extracts a core from the first falsified soft literal, which frequently yields a low-weight core. Conversely, weight-aware lookahead evaluates all falsified soft literals to construct a high-weight core.

As illustrated in Example 2, the weight-aware lookahead procedure only needs to detect a high-weight core to prune the branch. We refer the reader to [11] for detailed implementation and advanced scenarios.

Example 2: Considering the same input as in Example 1, we illustrate the execution trace of the weight-aware lookahead: Propagating l_1 implies $\neg l_3$. In Example 1, the propagation stopped and a core $\kappa_1 = \{l_1, l_3\}$ 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 $\kappa_2 = \{l_1, l_2\}$ with a weight of $w(\kappa_2) = 10$, raising the lower bound to $LB = 10$ by itself. So, only one local core is sufficient to prune the current branch, because we already have $LB \geq UB$.

IV. Versions submitted to PB26

For PB26, we submitted two distinct versions of WMaxCDCL2026, which are detailed as follows:

- **WMaxCDCL:** The standard version of the WMaxCDCL2026 solver.
- **WMaxCDCL-S9:** This version implements a sequential portfolio of WMaxCDCL2026 and SCIP [2]. Specifically, SCIP is executed initially with a 900-second time limit, followed by WMaxCDCL2026 for the remaining 2700 seconds.

Acknowledgment

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 by 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

[1] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Bdds for pseudo-boolean constraints – revisited. In Laurent Sakallah, Karem A.and Simon, editor, *SAT 2011*, pages 61–75, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[3] 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.

[4] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modelling and Computation*, 2(1-4):1–26, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] Jialu Zhang, Chu-Min Li, Sami Cherif, and Shuolin Li. Weight-aware branch-and-bound for weighted maximum satisfiability. 2026. To appear in proceedings of IJCAI-26.