

Revising the Implicit Hitting Set Solver MaxHS

Edgar Justus¹, Jakob Berg¹, André Schidler¹, Mathias Fleury¹, Tobias Paxian¹ and Armin Biere¹

¹University of Freiburg, Chair of Computer Architecture, Germany

Abstract

Maximum Satisfiability (MaxSAT) is an approach for solving NP-hard optimization problems expressed in propositional logic. Among the known approaches for solving MaxSAT is the *Implicit Hitting Set* approach that combines SAT solvers and Integer Linear Programming solvers. However, no Implicit Hitting Set solver participated in the most important evaluation of state-of-the-art MaxSAT solvers—the annual *MaxSAT Evaluation*—since *MaxHS* did in 2022. Since then, MaxHS remains unmaintained and is not allowed to participate under the current ruleset. Further, recent advances in MaxSAT testing have revealed numerous bugs in MaxHS. In this paper, we describe our revision of MaxHS, which include numerous bugfixes, the integration of an open source Integer Linear Programming solver that allows MaxHS to participate in the evaluation, and an exact mode that avoids numerical problems.

Keywords

SAT, MaxSAT, MaxHS, Optimization, Implicit Hitting Set

1. Introduction

Building on the success of modern SAT solvers [1, 2], Maximum Satisfiability (MaxSAT) solvers allow declarative solving of hard optimization problems. Several exact approaches for solving MaxSAT instances are known: (i) Core-Guided [3, 4, 5, 6, 7, 8, 9, 10], (ii) Solution-Improving [11, 12, 13] (iii) Branch & Bound [14, 15], and (iv) Implicit Hitting Set (IHS) [16, 17].

MaxHS [16] is the most up-to-date IHS MaxSAT solver but remains unmaintained since 2022. Further, MaxHS relies on the proprietary Mixed Integer Programming Solver CPLEX, which is against the Open-Source mandate of the MaxSAT Evaluation. Hence, no IHS solvers participate in the annual evaluation, although up-to-date results would be of interest due to the complementary nature of the IHS approach (see Section 4.3). Moreover, Paxian and Biere [18, 19]—using a novel MaxSAT fuzzer—showed that, as any complex piece of software, MaxSAT solvers—including MaxHS—contain numerous bugs. While current state-of-the-art MaxSAT solvers use this tool to find and remove bugs, no such effort has been undertaken for MaxHS.

In this paper, we describe our revision of MaxHS that addresses all these issues as follows:

- We fixed all MaxHS bugs identified by the fuzzer.
- We allow the use of optimizers other than CPLEX by providing an extensible optimizer interface.
- We provide an optimizer instantiation for the open source solver SCIP, enabling MaxHS to participate in the MaxSAT Evaluation.
- We leverage SCIP’s numerically stable mode for an exact mode that avoids numerical errors.
- We update the dependencies CaDiCaL (SAT solver) and MaxPre (preprocessing) to its most recent releases. Part of this update was moving all of MaxHS’s modifications of CaDiCaL to the general CaDiCaL API, thereby significantly simplifying future updates.

We first give an overview over MaxSAT and IHS, then, give an overview of the most important changes and finally evaluate our revision in extensive experiments.

17th Pragmatics of SAT International Workshop 2026 (PoS’26)

✉ schidler@cs.uni-freiburg.de (A. Schidler); fleury@cs.uni-freiburg.de (M. Fleury); paxian@cs.uni-freiburg.de (T. Paxian); biere@cs.uni-freiburg.de (A. Biere)

0000-0001-6790-7158 (A. Schidler); 0000-0002-1705-3083 (M. Fleury); 0009-0005-2044-1393 (T. Paxian); 0000-0001-7170-9242 (A. Biere)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Related Work Besides MaxHS, LMHS [17, 20] is another IHS MaxSAT solver that is no longer maintained. Furthermore, Smirnov et al. [21] proposed an IHS approach for the more general problem of pseudo-Boolean optimization (PBO), which has been improved over recent years [22, 23, 24]. Recently, it was shown that IHS solvers can provide independently checkable certificates of optimality for solutions [25, 26].

2. Preliminaries

Fixing terminology, we consider propositional formulas in conjunctive normal form (CNFs): given a set of variables V , a literal is a variable x or its negation $\neg x$. Let $L = V \cup \{\neg x : x \in V\}$ be the set of all literals, then, a clause $C \subseteq L$ is a set of literals and a CNF F is a set of clauses. Given an assignment $\sigma : V \rightarrow \{0, 1\}$, we overload σ as follows, for negations $\sigma(\neg x) = 1 - \sigma(x)$, for a clause C we define $\sigma(C) = \max_{\ell \in C} \sigma(\ell)$ and for a CNF F we define $\sigma(F) = \min_{C \in F} \sigma(C)$. A *model* of a CNF F is an assignment σ such that $\sigma(F) = 1$ and F is *satisfiable* if it has a model, otherwise F is *unsatisfiable*.

Maximum Satisfiability (MaxSAT) [27] considers how many—possibly weighted—clauses of a CNF can be satisfied and extends SAT to optimization problems. We consider the more generally stated problem of *Partial Weighted MaxSAT* to which we simply refer as MaxSAT. Here, an instance $(\mathcal{H}, \mathcal{S}, w)$ consists of two CNFs, the (assumed satisfiable) *hard clauses* $\mathcal{H}$ and the *soft clauses* $\mathcal{S}$, as well as a weight function $w : \mathcal{S} \rightarrow \mathbb{N}^+$ assigning a weight to each soft clause. Any model σ of $\mathcal{H}$ is a solution and has weight $w(\sigma) = \sum_{C \in \mathcal{S}} \sigma(C) \cdot w(C)$ and cost $c(\sigma) = \sum_{C \in \mathcal{S}} w(C) - w(\sigma)$. An optimal solution is a solution of maximum weight—or equivalently minimum cost—over all possible models of $\mathcal{H}$. An instance where all weights are 1 is considered *unweighted*, otherwise it is *weighted*.

Many MaxSAT solving approaches—including IHS—use a different characterization framing MaxSAT as a linear optimization problem. Here, an instance is a tuple (F, O) where F is a CNF and O is an objective, $O \equiv \sum_i w_i \cdot b_i$, with each b_i being a variable occurring in F and $w_i \in \mathbb{N}^+$. The cost of an assignment σ for F is $c(\sigma) = \sum_i w_i \cdot \sigma(b_i)$ and the optimal solution to (F, O) is a model for F that minimizes the cost. We can convert an instance $(\mathcal{H}, \mathcal{S}, w)$ to an instance (F, O) by setting $F \equiv \mathcal{H} \cup_{C \in \mathcal{S}} \text{CNF}(\neg b_C \leftrightarrow C)$ and $O \equiv \sum_{C \in \mathcal{S}} w(C) \cdot b_C$. This characterization allows using a SAT solver to *extract* cores and for the remainder of the paper we define a MaxSAT instance as (F, O) .

Cores An *Unsatisfiable Core* or simply a core is a set of clauses that is unsatisfiable when viewed as a CNF. We are interested in unsatisfiable cores that are subsets of the soft clauses. Such a core implies that at least one of the soft clauses cannot be satisfied and *incurs cost*. Conveniently, using the linear optimization characterization, a core $K = \{C_1, \dots, C_n\}$ with $K \subseteq \mathcal{S}$ corresponds to a set $K' = \{b_C \mid C \in K\}$ with $F \models \bigvee_{C \in K} b_C$.

2.1. Implicit Hitting Set Solving

IHS uses the fact that for an instance (F, O) any model of F is a hitting set over the instances' cores as follows. We overload the assignment σ to also refer to its set representation $\sigma = \{v \in V \mid \sigma(v) = 1\} \cup \{\neg v \in V \mid \sigma(v) = 0\}$. Then, for an arbitrary core K' of (F, O) and an arbitrary model σ of F , it must hold that $K' \cap \sigma \neq \emptyset$. Further, in case σ is a cost-minimizing hitting set over all cores, σ is an optimal solution. This also implies that any minimum hitting set over a subset of the cores gives a lower bound on the optimal cost. Hitting sets are computed using an *optimizer*.

Cores are iteratively extracted using a *core extractor*, as enumerating all cores is generally not feasible. The core extractor takes F and a partial assignment σ' (the *assumptions*), then, if possible the core extractor extends σ' to a model of F or returns a core if σ' cannot be extended.

Let B be the set of all objective variables occurring in O , γ be a hitting set over some of the cores, and $\sigma' = \{\neg b \in B \mid b \notin \gamma\}$ be the partial assignment stating that no objective outside the hitting set incurs cost. Then, in case the core extractor returns a core K' , we know that $K' \cap \gamma = \emptyset$. Further, in case the core extractor extends σ' to a model, this model is an optimal solution, as the minimum hitting set is a lower bound. SAT solvers are usually used as core extractors and can also reduce the core's size.

Algorithm 1: Implicit Hitting Set-based MaxSAT solving

Data: Instance (F, O)
Result: Optimal solution σ
cores $\leftarrow \emptyset$
while *TRUE* **do**
 $\gamma \leftarrow \text{minimum-hs}(\text{cores}, O)$
 $\sigma' \leftarrow \{\neg b \mid b \notin \gamma\}$
 $(\text{sat}, \sigma, K') \leftarrow \text{extract_core}(F, \sigma')$
 if *sat* **then return** σ ;
 cores $\leftarrow \text{cores} \cup \{K'\}$
end

Algorithm 1 shows the basic IHS loop. Starting from no cores, each iteration the algorithm computes an—initially empty—hitting set over the current set of cores using the optimizer. The algorithm then uses the core extractor to check whether any model for F exists that sets all objective variables not in the hitting set to 0. In case such a model exists, we have found an optimal solution. Otherwise, the core extractor returns a core, this core is added to our set of cores and the next iterations starts.

While this section describes the basic approach, actual IHS instantiations are more sophisticated. We also want to note that the optimizer is usually not performing a hitting set computation. Due to advanced methods that add general clausal constraints from the formula (seeding), or even pseudo-Boolean constraints (e.g., via abstract cores [28]), the optimizer generally solves a pseudo-Boolean optimization problem. However, the result is a hitting set over the cores.

3. MaxHS Revised

MaxHS is a sophisticated IHS implementation that uses simplification techniques, numerous heuristics, extensions, a linear sat-unsat solver, and many conditional choices to maximize its performance [16, 29, 30, 28, 31, 32]. As such, it is expected that MaxHS would contain bugs, and indeed Paxian and Biere [18, 19] were able to show this using a novel fuzzer, which was the starting point for our revision. The sophistication of MaxHS as measured in a thousand lines of code (KLoC) is with ≈ 13 KLoC similar to other MaxSAT solvers (EvalMaxSAT has ≈ 11 KLoC and UWMaxSAT has ≈ 10 KLoC), more than the Glucose SAT solver [33] (≈ 6.5 KLoC) and less than the CaDiCaL SAT solver (≈ 65 KLoC).

In this section, we describe our changes to MaxHS. Since we do not extend the core functionality, we prefer not to rename the solver. To distinguish the base version and our revised version, we use *MaxHS-old* to refer to the former and *MaxHS-new* to refer to the latter.

Fuzzing We use *WCNFuzz* [19] to detect bugs in *MaxHS-old* and *MaxHS-new*. *WCNFuzz* strategically generates random MaxSAT instances and tests whether the output of the MaxSAT solver is correct for the generated instance. Whether the solution is optimal is determined by comparing the optimal cost claimed by different solvers. We use *EvalMaxSAT* [34] to ensure that the optimal cost is known, as *EvalMaxSAT* did not claim a single wrong optimal result during extensive fuzzing.

3.1. Optimizer Abstraction

MaxHS-old used CPLEX as its optimizer. However, there exist numerous other possible optimizers, such as other Integer Linear Programming solvers or pseudo-Boolean solvers. Furthermore, due to CPLEX’s proprietariness, its use violates MaxSAT Evaluation’s open-source mandate¹.

We address this issue by abstracting the optimizer. Hence, *MaxHS-new* has an abstract *Optimizer* class and can use any implementation of that abstract class as its optimizer. *MaxHS-new* provides an interface

¹<https://maxsat-evaluations.github.io/2026/rules.html>

for CPLEX and the open source solver SCIP [35]. Optimizers can be chosen using `-optimizer=<value>` with possible values being `cplex`, `scip`, and `scip_exact` (see Section 3.3).

3.2. Bugfixes

In this section, we discuss the major bugs causing the most errors identified by WCNFuzz.

Duplicate Literals MaxHS-*old* internally assumed that a literal occurs at most once in a clause and violating this assumption caused many different errors. Due to several bugs, MaxHS-*old* did not remove duplicate literals, which did not trigger on MaxSAT Evaluation instances, as they do not contain duplicate literals. MaxHS-*new* ensures that duplicate literals are properly removed.

Self-Subsumption MaxHS removes clauses that are subsumed or redundant due to self-subsumption. The self-subsumption check in MaxHS-*old* was not sound, leading to more clauses being removed than should be removed, and potentially wrong results, as the resulting solution might not satisfy all clauses. MaxHS-*new* contains an adjusted check that is sound.

Rounding Errors MaxHS-*old* used floating point numbers for storing weights, as well as the current lower and upper bound. IEEE 754 double precision float point numbers can only faithfully represent integer values of up to 2^{53} and introduces rounding errors for higher values. However, the WCNF format allows for weights of up to $2^{63} - 1$ and the sum of all weights can be as much as $2^{64} - 1$. MaxHS-*old* used the *long double* datatype which on some systems maps to an 80-bit or longer floating point number, which can faithfully represent 64-bit integer values, but on other systems maps to the aforementioned double precision format. Further, even using an 80-bit representation requires care to not introduce rounding errors during arithmetic operations.

MaxHS-*new* uses 64-bit integers for all variables that represent weights and the upper bound. The lower bound is represented by a custom datatype that combines a 64-bit integer with a double precision floating point number. The latter is used to represent fractions smaller than one, as solutions for the LP relaxation are generally fractional. Unfortunately, SCIP and CPLEX internally use double precision floating point numbers, leading to rounding errors that can be reliably triggered using WCNFuzz. While we cannot avoid the rounding errors caused by the optimizer, MaxHS-*new* implements an exact mode based on the numerically stable mode supported by SCIP.

3.3. Exact Mode

MaxHS-*new*'s exact mode relies on SCIP's exact mode [36, 37] that is numerically stable and, hence, does not suffer from rounding errors at the expense of performance and memory usage. This mode is available using the `scip_exact` optimizer and, together with the aforementioned switch to integers with our custom datatype, ensures sufficient numerical stability within MaxHS-*new*. The only feature impacting soundness in exact mode is *reduced cost fixing* [30]. Since the reduced costs cannot be retrieved from SCIP via a numerically stable interface, reduced cost fixing should be deactivated (using `-no-lp-harden-reduced`) to ensure soundness.

3.4. Core Extractor

MaxHS-*old* used CaDiCaL's [38, 39] version from the 2021 SAT Competition (between 1.4.0 and 1.4.1) with several modifications giving MaxHS-*old* access to information and functionality not available over the standard CaDiCaL API. These modifications necessitate modifying each new version of CaDiCaL before it can be used within MaxHS-*old*. MaxHS-*new* does not only use the newest version 3.0, but all necessary modifications are now part of the standard CaDiCaL API, making future updates simple.

3.5. Preprocessing

MaxHS-*old* used MaxPre 1 [40] for preprocessing, which WCNFuzz found to contain bugs leading to wrong results. MaxHS-*new* uses MaxPre 2.2 [41, 42], which does not only add improvements and new techniques but also provides proofs that the preprocessing is correct. While MaxHS-*new* does not verify these proofs, fuzzing could not detect any wrong results connected to MaxPre 2.2. However, MaxPre 2.2 breaks some invariants within MaxHS-*old*, like removing variables, which requires extra consideration, and we see tuning MaxHS-*new* towards MaxPre 2.2 to achieve better performance as future work.

4. Experimental Evaluation²

We discussed our modifications to MaxHS-*old* and why they were necessary. It remains to investigate whether we achieved our goal of removing erroneous results and whether our changes impact the performance of the solver. Hence, we try to answer three questions in our evaluation:

1. How many errors occur when running MaxHS-*old* and MaxHS-*new*?
2. How does MaxHS-*new* perform in comparison to MaxHS-*old*?
3. How does MaxHS-*new* perform compared to state-of-the-art MaxSAT solvers?

We use fuzzing to answer the first question and a collection of benchmarks and solvers from the MaxSAT Evaluation to answer the other questions.

4.1. Setup

Benchmarks We use the benchmarks from the exact/complete track of the MaxSAT Evaluation 2022, 2023 and 2024 for our experiments. After eliminating duplicate instances using SHA-1 hashes, this benchmark set contains 3101 instances, 1557 of them unweighted and 1544 weighted.

Execution Environment We use the *bwForCluster Helix* for benchmarking: each node has 256 GB memory and two AMD EPYC 7513 CPUs, each with 32 cores running at 2.6 GHz. We run our experiments by exclusively reserving a node and running 16 instances in parallel, each with a memory limit of 28 GB, and a time limit of one hour. Further, fuzzing is performed using 64 parallel threads on a single machine with two AMD EPYC 7343 CPUs, 32 physical cores, 64 virtual cores, and 2 TB of memory.

MaxHS MaxHS-*new*³ uses CaDiCaL 3.0 [38, 39]⁴ as the SAT solver, MaxPre 2.2 [41, 42]⁵, CPLEX 22.1.1.0⁶, and SCIP 10.0.2 [35]⁷ with enabled exact mode [36, 37]. MaxHS-*new*, CaDiCaL, MaxPre, and SCIP are compiled with gcc 13.3.0 (Ubuntu). We compare against MaxHS-*old* using the 2022 MaxSAT Evaluation version⁸, which was the starting point for this project. We use the provided binary which uses CPLEX 20.1.0.0, MaxPre 1 [40], and CaDiCaL’s 2021 competition version (between 1.4.0 and 1.4.1). We consider four different MaxHS-*new* configurations: *MaxHS-new CPLEX* and *MaxHS-new SCIP* use the respective optimizers, while *MaxHS-new Exact* uses the exact mode without reduced-cost fixing. Further, *MaxHS-new NoPP* uses CPLEX but does not use MaxPre for preprocessing.

Fuzzing We use WCNFuzz [19]⁹ giving each solver 20 seconds to solve an instance together with *EvalMaxSAT* [34]¹⁰ to have a reliably optimal solution to compare against.

²The MaxHS-*new* version used for benchmarking and detailed results can be found at <https://doi.org/10.5281/zenodo.20072899>.

³<https://github.com/ASchidler/MaxHS>

⁴<https://github.com/arminbiere/cadical>

⁵<https://bitbucket.org/coreo-group/maxpre2/>, commit 0ec2599c

⁶<https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio>

⁷<https://scipopt.org/>

⁸<https://maxsat-evaluations.github.io/2022/descriptions.html>

⁹<https://github.com/tobipaxe/MaxSAT-Fuzzer>, commit 5e0e3656

¹⁰<https://github.com/FlorentAvellaneda/EvalMaxSAT/>, Commit 94e6774

Solvers We compare against the winners in the unweighted and weighted track of the 2024 MaxSAT Evaluation: *WMaxCDCL* [15] (OpenWBO 1200 configuration) and *CASHWMaxSAT* [43] (DisjCad S6 configuration). Further, we also use *UWrMaxSAT* [44] (with SCIP and MaxPre) as it is used within *CASHWMaxSAT* and it performs overall well on weighted and unweighted instances. For all three solvers we used the binary provided to the MaxSAT Evaluation¹¹. While *CASHWMaxSAT* and *UWrMaxSAT* are code-guided solvers, they additionally use SCIP. Hence, we also compare against the pure core-guided solver *EvalMaxSAT*, which was also the baseline used during fuzzing.

4.2. Fuzzing Results

We give the results from running the fuzzer for over 24 hours in Table 1. We distinguish between errors from instances where the sum of weights does not exceed 2^{32} and instances with a larger sum of weights. In the former case we do not expect rounding errors, while we expect some unavoidable rounding errors with high weights. A MaxSAT solver outputs the minimum cost and a solution, i.e., an assignment of minimum cost and we further distinguish the following errors:

1. *Crashes* are instances where the solver terminates in an unexpected manner, which includes the *UNKNOWN* return code and whenever the integer linear program becomes infeasible.
2. *Wrong Results* occur whenever the solver: returns a model and minimum cost that are not optimal, returns a solution for an unsatisfiable instance, claims a satisfiable instance is unsatisfiable, or returns a solution that does not satisfy the instance.
3. *Wrong Models* indicate that either the reported cost is minimal, or the solution is optimal, but not both.

Table 1

Different errors after fuzzing the different solvers with 1 496 544 executions taking a total of 1 023,39 hours. *WR* indicates wrong results and *WM* indicates wrong models.

Solver	Timeouts	Sum Weights $\leq 2^{32}$			Sum Weights $> 2^{32}$		
		Crashes	WR	WM	Crashes	WR	WM
<i>MaxHS-old</i>	5.58%	38 049	31 458	19 341	140 632	21 942	32 725
<i>MaxHS-new</i> CPLEX	0.21%	0	0	0	10 992	20 936	0
<i>MaxHS-new</i> NoPP	0.19%	0	0	0	6 273	40 671	0
<i>MaxHS-new</i> SCIP	0.22%	0	26	0	15 164	39 768	0
<i>MaxHS-new</i> Exact	0.23%	0	0	0	0	0	0

The fuzzing results show that all types of errors occur frequently with *MaxHS-old*, even with lower weights. Hence, the errors are not just related to rounding errors. Interestingly, *MaxHS-old* times out on a significantly higher fraction of generated instances, which is likely due to *MaxHS-old* getting stuck in an invalid state where no progress is made.

As expected, almost no errors occur with the different *MaxHS-new* configurations on low weights, except very few errors when using SCIP. These errors are due to an unknown edge-case or rounding error when retrieving the reduced costs, leading to a wrong lower bound. This can be avoided by disabling reduced cost fixing (*-no-lp-harden-reduced*) at the cost of some performance. The error frequency increases with higher weights but is still significantly lower than with *MaxHS-old*. The exact mode shows no errors and, hence, indeed successfully avoids rounding errors.

4.3. Benchmark Results

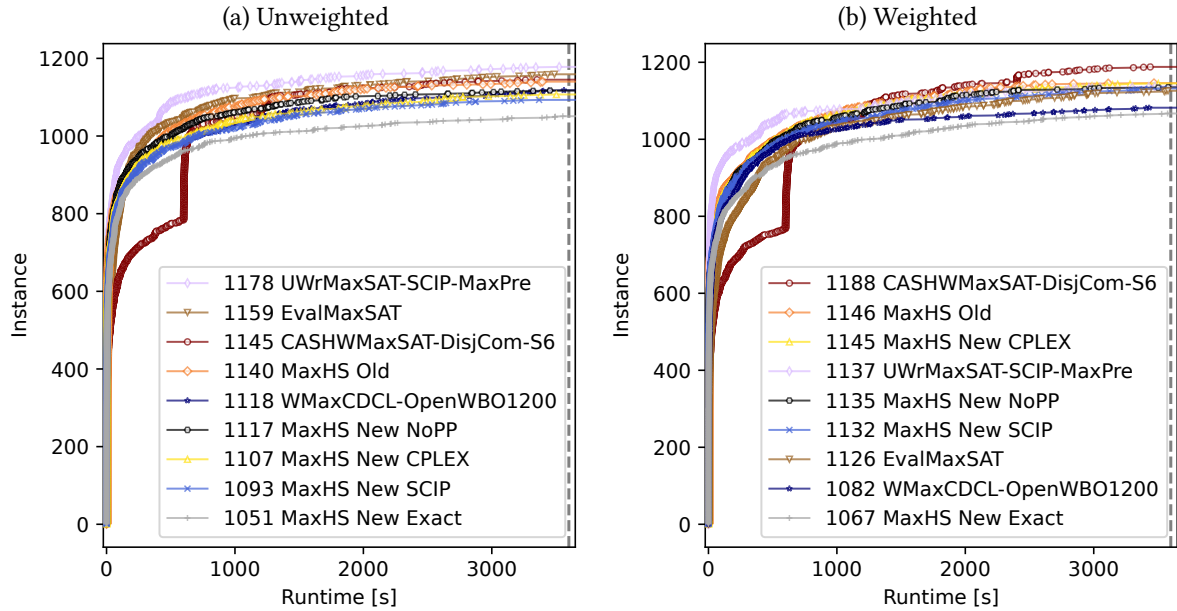
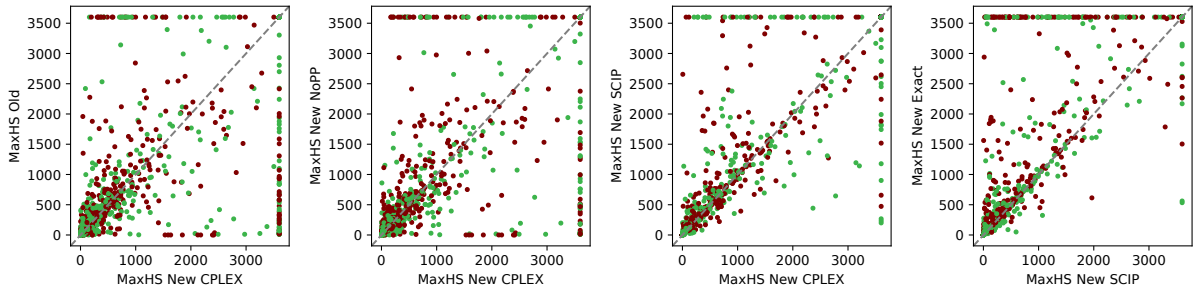
Table 2 and Figure 1 show the results from running *MaxHS-old*, *MaxHS-new* and the state-of-the-art solvers on the benchmark instances. We verified that the reported optimal cost is the same as the cost of the returned solution and that the reported cost is indeed not higher than that of any solver. Overall, only *MaxHS-new* SCIP reported one wrong result caused by high weights and numerical instability.

¹¹<https://maxsat-evaluations.github.io/2024/descriptions.html>

Table 2

Results on the number of solved instances by different solvers and MaxHS-*new* configurations.

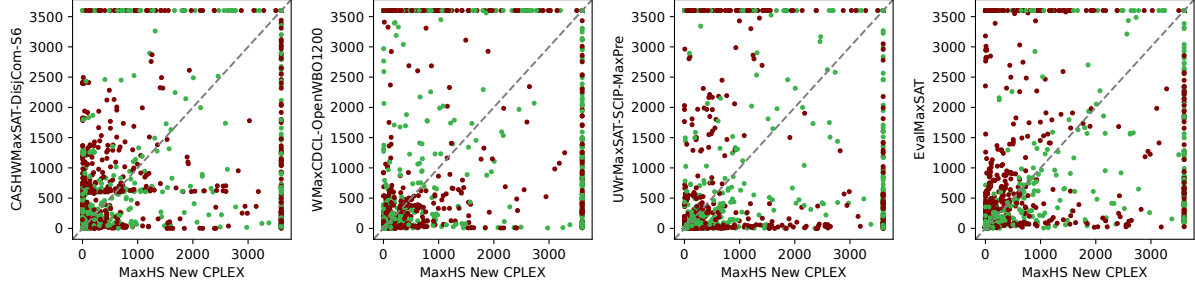
Solver	Solved	Time Out	Mem Out	Unweighted	Weighted
MaxHS New CPLEX	2252	816	33	1107	1145
MaxHS New NoPP	2252	824	25	1117	1135
MaxHS New SCIP	2225	837	39	1093	1132
MaxHS New Exact	2118	925	58	1051	1067
MaxHS Old	2286	789	26	1140	1146
WMaxCDCL-OpenWBO1200	2200	901	0	1118	1082
CASHWMaxSAT-DisjCom-S6	2333	748	20	1145	1188
UWrMaxSAT-SCIP-MaxPre	2315	769	17	1178	1137
EvalMaxSAT	2285	815	1	1159	1126
Available	3101			1557	1544

Figure 1: Runtime comparison of different MaxHS-*new* configurations and other MaxSAT solvers.**Figure 2:** Comparison of the runtime (in seconds) of MaxHS-*new* and MaxHS-*old*, as well as different configurations of MaxHS-*new*. Unweighted instances are marked light green and weighted instances dark red.

MaxHS-*old* and MaxHS-*new* The results show that MaxHS-*old* performs better on unweighted instances than MaxHS-*new*, while performance on weighted instances is similar. Further, Figure 2 suggests that MaxHS-*old* and MaxHS-*new* do not perform well on the same instances. Given the large number of changes it is not possible to pinpoint the source of the regression on unweighted instances. It is reasonable to expect that removing unsound methods leads to worse performance. Another possible

explanation is that MaxHS is a fine-tuned and complex system and our changes require re-tuning the components. We see performance improvements as future work.

Figure 3: Comparison of the runtime (in seconds) of MaxHS-*new* and state-of-the-art MaxSAT solvers. Unweighted instances are marked light green and weighted instances dark red.



MaxHS-new Configurations Using SCIP over CPLEX decreases performance, as indicated by the overview in Table 2. The decrease in solved instances is mild and Figure 2 suggests that using SCIP improves performance on several instances, weighted and unweighted, making CPLEX the better default choice but not necessarily the best choice for a given instance. Further, MaxPre 2.2 does not lead to generally better performance, the 59 more instances that could only be solved with MaxPre 2.2 are balanced by the 59 instances that could only be solved by not using MaxPre 2.2. Since performance is better on weighted instances, disabling MaxPre on unweighted instances (*-no-use-maxpre*) might be the better default choice. Exact mode does decrease performance, as expected. Indeed, Figure 2 shows that exact mode is almost strictly worse than using SCIP without exact mode and with reduced cost fixing. Additionally, the memory consumption almost doubles.

State-of-the-Art Solvers The comparison with state-of-the-art solvers shows a similar pattern as the comparison with MaxHS-*old*. MaxHS-*new* performs competitive on weighted instances and shows comparatively poor performance on unweighted instances. A closer look at the results in Figure 3 shows that MaxHS-*new* performs better on several instances compared to the other solvers, ranging from solving 89 different instances (CASHWMaxSAT) to 246 different instances (WMaxCDCL). This indicates a generally complementary behavior to non-IHS approaches.

5. Conclusion and Future Work

We presented our revision of MaxHS with the goal to motivate other researchers to contribute and keep MaxHS a state-of-the-art MaxSAT solver, as there are many avenues for improving MaxHS.

While we did investigate and remove some issues that caused decreased performance it was not our priority. Hence, our next step would be to improve the performance of MaxHS-*new*. This includes the regression when solving the problematic unweighted instances, as well as re-tuning the components of MaxHS towards changed behaviour due to CaDiCaL, MaxPre, and our changes. Due to the large number of changes and novel features in CaDiCaL 3.0, we expect great performance potential when tuning MaxHS, but also when tuning CaDiCaL to this specific application.

In this paper, we only investigated the use of CPLEX and SCIP. However, there are numerous other possible optimizers. Due to our abstraction, it should be reasonably easy to extend MaxHS-*new* to use more optimizers.

Finally, Ihalainen et al. [25] showed that it is possible to certify IHS results, as long as a certifying optimizer is used to prove optimality of the last optimizer call. This result should translate to MaxHS-*new* and would further improve the reliability of MaxHS-*new*.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

- [1] Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021. URL: <https://doi.org/10.3233/FAIA336>. doi:10.3233/FAIA336.
- [2] J. K. Fichte, D. L. Berre, M. Hecher, S. Szeider, The silent (r)evolution of SAT, *Commun. ACM* 66 (2023) 64–72. URL: <https://doi.org/10.1145/3560469>. doi:10.1145/3560469.
- [3] B. Andres, B. Kaufmann, O. Matheis, T. Schaub, Unsatisfiability-based optimization in clasp, in: *ICLP (Technical Communications)*, volume 17 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 211–221.
- [4] C. Ansótegui, F. Didier, J. Gabàs, Exploiting the structure of unsatisfiable cores in MaxSAT, in: *IJCAI*, AAAI Press, 2015, pp. 283–289.
- [5] C. Ansótegui, J. Gabàs, WPM3: an (in)complete algorithm for weighted partial MaxSAT, *Artif. Intell.* 250 (2017) 37–57.
- [6] N. S. Bjørner, N. Narodytska, Maximum satisfiability using cores and correction sets, in: *IJCAI*, AAAI Press, 2015, pp. 246–252.
- [7] Z. Fu, S. Malik, On solving the partial MAX-SAT problem, in: *SAT*, volume 4121 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 252–265. URL: https://doi.org/10.1007/11814948_25. doi:10.1007/11814948_25.
- [8] J. Marques-Silva, J. Planes, On using unsatisfiability for solving maximum satisfiability, *CoRR* abs/0712.1097 (2007). URL: <http://arxiv.org/abs/0712.1097>. arXiv:0712.1097.
- [9] A. Morgado, C. Dodaro, J. Marques-Silva, Core-guided MaxSAT with soft cardinality constraints, in: *CP*, volume 8656 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 564–573.
- [10] N. Narodytska, F. Bacchus, Maximum satisfiability using core-guided MaxSAT resolution, in: *AAAI*, AAAI Press, 2014, pp. 2717–2723.
- [11] D. L. Berre, A. Parrain, The Sat4j library, release 2.2, *J. Satisf. Boolean Model. Comput.* 7 (2010) 59–6.
- [12] N. Eén, N. Sörensson, Translating pseudo-boolean constraints into SAT, *J. Satisf. Boolean Model. Comput.* 2 (2006) 1–26.
- [13] T. Paxian, S. Reimer, B. Becker, Dynamic polynomial watchdog encoding for solving weighted MaxSAT, in: *SAT*, volume 10929 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 37–53.
- [14] A. Abramé, D. Habet, Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver, *J. Satisf. Boolean Model. Comput.* 9 (2014) 89–128. URL: <https://doi.org/10.3233/sat190104>. doi:10.3233/SAT190104.
- [15] J. Coll, C. M. Li, S. Li, D. Habet, F. Manyà, Solving weighted maximum satisfiability with branch and bound and clause learning, *Comput. Oper. Res.* 183 (2025) 107195. URL: <https://doi.org/10.1016/j.cor.2025.107195>. doi:10.1016/J.COR.2025.107195.
- [16] J. Davies, F. Bacchus, Exploiting the power of mip solvers in maxsat, in: *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference*, Helsinki, Finland, July 8–12, 2013. *Proceedings, Lecture Notes in Computer Science*, Springer, 2013, pp. 166–181. URL: https://doi.org/10.1007/978-3-642-39071-5_13. doi:10.1007/978-3-642-39071-5_13.
- [17] P. Saikko, J. Berg, M. Jarvisalo, LMHS: A SAT-IP hybrid MaxSAT solver, in: *SAT*, volume 9710 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 539–546. URL: https://doi.org/10.1007/978-3-319-40970-2_34. doi:10.1007/978-3-319-40970-2_34.
- [18] T. Paxian, A. Biere, Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging, in: *Proceedings of the 14th International Workshop on Pragmatics of SAT*, Alghero, Italy, July 4, 2023, *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 59–71. URL: <https://ceur-ws.org/Vol-3545/paper5.pdf>.

- [19] T. Paxian, A. Biere, Maxsat fuzzing and delta debugging, *J. Artif. Intell. Res.* 85 (2026). URL: <https://doi.org/10.1613/jair.1.19716>. doi:10.1613/JAIR.1.19716.
- [20] P. Saikko, Implicit Hitting Set Algorithms for Constraint Optimization, Ph.D. thesis, University of Helsinki, Finland, 2019. URL: <http://hdl.handle.net/10138/306951>.
- [21] P. Smirnov, J. Berg, M. Järvisalo, Pseudo-boolean optimization by implicit hitting sets, in: 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 51:1–51:20. URL: <https://doi.org/10.4230/LIPIcs.CP.2021.51>. doi:10.4230/LIPIcs.CP.2021.51.
- [22] P. Smirnov, J. Berg, M. Järvisalo, Improvements to the implicit hitting set approach to pseudo-boolean optimization, in: 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, Haifa, Israel, August 2-5, 2022, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 13:1–13:18. URL: <https://doi.org/10.4230/LIPIcs.SAT.2022.13>. doi:10.4230/LIPIcs.SAT.2022.13.
- [23] H. Ihalainen, J. Berg, M. Järvisalo, B. Bogaerts, Symmetric core learning for pseudo-boolean optimization by implicit hitting sets, in: 31st International Conference on Principles and Practice of Constraint Programming, CP 2025, Glasgow, Scotland, August 10-15, 2025, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, pp. 15:1–15:26. URL: <https://doi.org/10.4230/LIPIcs.CP.2025.15>. doi:10.4230/LIPIcs.CP.2025.15.
- [24] H. Ihalainen, D. Vandesande, A. Schidler, J. Berg, M. Järvisalo, HitPBO: An Implicit Hitting Set Solver for Pseudo-Boolean Optimization, in: 29th International Conference on Theory and Applications of Satisfiability Testing (SAT 2026), volume 377 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2026, pp. 34:1–34:13. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2026.34>. doi:10.4230/LIPIcs.SAT.2026.34.
- [25] H. Ihalainen, D. Vandesande, A. Schidler, J. Berg, B. Bogaerts, M. Järvisalo, Efficient and reliable hitting-set computations for the implicit hitting set approach, in: Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026, AAAI Press, 2026, pp. 14251–14260. URL: <https://doi.org/10.1609/aaai.v40i17.38439>. doi:10.1609/AAAI.V40I17.38439.
- [26] B. Bogø, X. Chen, W. Koops, P. Lu, J. Nordström, M. Vinyals, Q. Wu, Certified implicit hitting set solving with local search for pseudo-boolean optimization, in: Proceedings of the 16th Pragmatics of SAT International, Glasgow, UK, August 10-11, 2025, volume 4008 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. Presentation-Only.
- [27] F. Bacchus, M. Järvisalo, R. Martins, Maximum satisfiability, in: Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, IOS Press, 2021, pp. 929–991. URL: <https://doi.org/10.3233/FAIA201008>. doi:10.3233/FAIA201008.
- [28] J. Berg, F. Bacchus, A. Poole, Abstract cores in implicit hitting set MaxSAT solving, in: Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, Lecture Notes in Computer Science, Springer, 2020, pp. 277–294. URL: https://doi.org/10.1007/978-3-030-51825-7_20. doi:10.1007/978-3-030-51825-7_20.
- [29] J. Davies, Solving MAXSAT by Decoupling Optimization and Satisfaction, Ph.D. thesis, University of Toronto, Canada, 2014. URL: <http://hdl.handle.net/1807/43539>.
- [30] F. Bacchus, A. Hyttinen, M. Järvisalo, P. Saikko, Reduced cost fixing in MaxSAT, in: Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, Lecture Notes in Computer Science, Springer, 2017, pp. 641–651. URL: https://doi.org/10.1007/978-3-319-66158-2_41. doi:10.1007/978-3-319-66158-2_41.
- [31] F. Bacchus, Maxhs in the 2022 maxsat evaluation, MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions (2022) 17.
- [32] R. Hickey, F. Bacchus, Large neighbourhood search for anytime MaxSAT solving, in: Proceedings

- of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, *ijcai.org*, 2022, pp. 1818–1824. URL: <https://doi.org/10.24963/ijcai.2022/253>. doi:10.24963/IJCAI.2022/253.
- [33] G. Audemard, L. Simon, On the glucose SAT solver, *Int. J. Artif. Intell. Tools* 27 (2018) 1840001:1–1840001:25. URL: <https://doi.org/10.1142/S0218213018400018>. doi:10.1142/S0218213018400018.
 - [34] F. Avellaneda, A short description of the solver evalmaxsat, *MaxSAT Evaluation 2021* (2021) 10.
 - [35] C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, A. Chmiela, J. Dionísio, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, S. J. Maher, G. Mexi, E. Mühmer, M. E. Pfetsch, S. Pokutta, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, L. Xu, The SCIP Optimization Suite 10.0, Technical Report, Optimization Online, 2025. URL: <https://optimization-online.org/2025/11/the-scip-optimization-suite-10-0/>.
 - [36] W. J. Cook, T. Koch, D. E. Steffy, K. Wolter, A hybrid branch-and-bound approach for exact rational mixed-integer programming, *Math. Program. Comput.* 5 (2013) 305–344. URL: <https://doi.org/10.1007/s12532-013-0055-6>. doi:10.1007/S12532-013-0055-6.
 - [37] L. Eifler, A. M. Gleixner, A computational status update for exact rational mixed integer programming, *Math. Program.* 197 (2023) 793–812. URL: <https://doi.org/10.1007/s10107-021-01749-5>. doi:10.1007/S10107-021-01749-5.
 - [38] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleys, F. Pollitt, Cadical 2.0, in: *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I, Lecture Notes in Computer Science*, Springer, 2024, pp. 133–152. URL: https://doi.org/10.1007/978-3-031-65627-9_7. doi:10.1007/978-3-031-65627-9_7.
 - [39] F. Pollitt, M. Fleury, K. Fazekas, N. Froleys, A. Schidler, D. Schreiber, A. Biere, CaDiCaL 3.0, in: *29th International Conference on Theory and Applications of Satisfiability Testing (SAT 2026)*, volume 377 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2026, pp. 40:1–40:14. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2026.40>. doi:10.4230/LIPIcs.SAT.2026.40.
 - [40] T. Korhonen, J. Berg, P. Saikko, M. Järvisalo, Maxpre: An extended MaxSAT preprocessor, in: *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, Lecture Notes in Computer Science*, Springer, 2017, pp. 449–456. URL: https://doi.org/10.1007/978-3-319-66263-3_28. doi:10.1007/978-3-319-66263-3_28.
 - [41] H. Ihalainen, J. Berg, M. Järvisalo, Clause redundancy and preprocessing in maximum satisfiability, in: *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings, Lecture Notes in Computer Science*, Springer, 2022, pp. 75–94. URL: https://doi.org/10.1007/978-3-031-10769-6_6. doi:10.1007/978-3-031-10769-6_6.
 - [42] H. Ihalainen, A. Oertel, Y. K. Tan, J. Berg, M. Järvisalo, M. O. Myreen, J. Nordström, Certified MaxSAT preprocessing, in: *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I, Lecture Notes in Computer Science*, Springer, 2024, pp. 396–418. URL: https://doi.org/10.1007/978-3-031-63498-7_24. doi:10.1007/978-3-031-63498-7_24.
 - [43] S. Pan, Y. Wang, S. Cai, An efficient core-guided solver for weighted partial MaxSAT, in: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025, ijcai.org*, 2025, pp. 2647–2656. URL: <https://doi.org/10.24963/ijcai.2025/295>. doi:10.24963/IJCAI.2025/295.
 - [44] M. Piotrów, Uwrmaxsat: Efficient solver for MaxSAT and pseudo-boolean problems, in: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020, IEEE*, 2020, pp. 132–136. URL: <https://doi.org/10.1109/ICTAI50040.2020.00031>. doi:10.1109/ICTAI50040.2020.00031.