LMHS: A SAT-IP Hybrid MaxSAT Solver

Paul Saikko, Jeremias Berg, and Matti Järvisalo(✉)

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Helsinki, Finland
matti.jarvisalo@cs.helsinki.fi

Abstract. We describe LMHS, an open-source weighted partial maximum satisfiability (MaxSAT) solver. LMHS is a hybrid SAT-IP MaxSAT solver that implements the implicit hitting set approach to MaxSAT. On top of the main algorithm, LMHS offers integrated preprocessing, solution enumeration, an incremental API, and the use of a choice of SAT and IP solvers. We describe the main features of LMHS, and give empirical results on the influence of preprocessing and the choice of the underlying SAT and IP solvers on the performance of LMHS.

1 Introduction

LMHS is a weighted partial maximum satisfiability (MaxSAT) solver. Weighted partial MaxSAT is a common generalization of maximum satisfiability that allows some clauses to be designated as mandatory and assigns weights to clauses that may be left unsatisfied. LMHS implements the so-called implicit hitting set approach [16,18] for weighted partial MaxSAT, and can be viewed as an independent from-scratch re-implementation of the MaxHS solver [8–10]. On top of the main algorithm, LMHS integrates MaxSAT preprocessing [3–6] into the solver, and offers solution enumeration, an incremental API, as well as the use of a choice of SAT and IP solvers. The solver entered the 2015 MaxSAT Evaluation [2], where it solved the most problems (among non-portfolio solvers) in the crafted and industrial weighed partial MaxSAT categories. This paper gives an overview of key features of the LMHS solver as well as the effects of preprocessing and the choice of the SAT and IP solvers on its performance.

2 Overview of LMHS

LMHS implements an instantiation of an implicit hitting set algorithm [16] for weighted partial MaxSAT, following MaxHS [8–10]. Given an unsatisfiable CNF formula $\mathcal{F}$, the MaxSAT problem is to identify a minimum (minimum-cost

Work funded by Academy of Finland, grants 251170 COIN, 276412, and 284591; and Doctoral School in Computer Science DoCS and Research Funds of the University of Helsinki.

N. Creignou and D. Le Berre (Eds.): SAT 2016, LNCS 9710, pp. 539–546, 2016.
DOI: 10.1007/978-3-319-40970-2_34

Algorithm 1. The MaxHS implicit hitting set algorithm for MaxSAT [8].

```
1: function MaxHS(F_h, F_s, c)
2:     K ← ∅, H ← ∅
3:     (sat, κ, τ) ← SolveSAT(F_h ∪ F_s)
4:     while not sat do
5:         K ← K ∪ {κ}
6:         H ← SolveMCHS(K, c)
7:         (sat, κ, τ) ← SolveSAT(F_h ∪ (F_s \ H))
8:     return τ
```

for weighted problems) set H of (soft) clauses such that $\mathcal{F}\backslash H$ is satisfiable. A connection to implicit hitting set problems comes from the unsatisfiable subsets of clauses, or cores κ, or the formula. The set of clauses H must hit a clause from each core κ, so an optimal MaxSAT solution can be obtained by computing a minimum-cost hitting set (MCHS) over the set of all cores. When the set of all cores is not known, this becomes an implicit hitting set problem.

The implicit hitting set approach for weighted partial MaxSAT, given hard clauses $\mathcal{F}_h$, soft clauses $\mathcal{F}_s$, and cost function $c : \mathcal{F}_s \to \mathbb{R}^+$, is described in more detail in Algorithm 1 and Fig. 1. It uses both a SAT and an IP solver. During the solving process it accumulates a set $\mathcal{K}$ of cores and stores a MCHS of $\mathcal{K}$ in H. Starting with $H = \emptyset$, the algorithm tests the satisfiability of $\mathcal{F}\backslash H$ using the SAT solver. If satisfiable, the variable assignment τ returned by the SAT solver is optimal. If unsatisfiable, a new core κ of $\mathcal{F}\backslash H$ is obtained from the SAT solver and added to $\mathcal{K}$. Finally, the IP solver is used to update H to a new hitting set by computing a MCHS of $\mathcal{K}$.

In practice, every soft clause $C_i \in \mathcal{F}_s$ is augmented with a unique auxiliary variable a_i so that if $a_i = 1$, then C_i need not be satisfied, i.e., creating the clause $C_i \vee a_i$. Arbitrary sets of soft clauses can then be excluded from the formula by assuming $a_i = 1$ for the corresponding auxiliary variables in a SAT solver call. To obtain a MCHS of $\mathcal{K}$, the IP solver minimizes $\sum_{C_i \in \mathcal{F}_s} a_i \cdot c(C_i)$ subject to the constraint $\sum_{C_i \in \kappa} a_i \geq 1$ for each core $\kappa \in \mathcal{K}$, enforcing that each core in $\mathcal{K}$ is hit.

Besides the features elaborated on in Sect. 3, some design choices differentiate LMHS from the MaxHS solver of Davies and Bacchus (http://maxhs.org). LMHS never enforces the equivalence $\neg a_i \leftrightarrow C_i$ of auxiliary variables and clauses explicitly in CNF. Instead, the value of each a_i is fixed via the assumptions interface for every SAT solver call, which ensure that $\neg a_i \leftrightarrow C_i$ implicitly holds. In terms of heuristic optimizations, by default LMHS finds a maximal disjoint set of cores on each iteration and uses a greedy hitting set algorithm in place of an IP solver call whenever possible. At each iteration, the greedy hitting set algorithm is used in place of an IP solver call as long as this results in an unsatisfiable formula (i.e., a core is produced). When the greedy method does not yield a core, the IP solver is used to compute a minimum-cost hitting set. The 2015 MaxSAT Evaluation versions LMHS-I and LMHS-C differ slightly in this regard: LMHS-I did not use the greedy algorithm, while LMHS-C finds a set

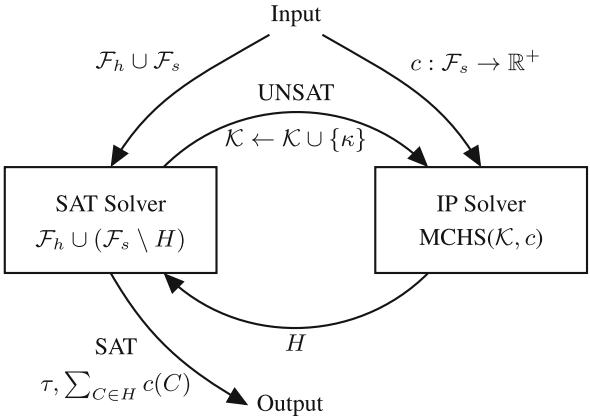

Fig. 1. Information flow in the implicit hitting set approach to MaxSAT

of possibly overlapping cores at each iteration. Furthermore, as a consequence of the integration of SAT-based preprocessing, auxiliary variables may not be limited to a single unique a_i per soft clause.

3 Features

Here we give an overview of the main features offered by LMHS on top of the main algorithm it implements.

Integrated Preprocessing. The use of SAT preprocessing techniques for MaxSAT [4] is integrated into LMHS using the Coprocessor 2.0 SAT preprocessor [17]. Many SAT preprocessing techniques, such as bounded variable elimination [11], are not sound for MaxSAT on their own [4]. However, they can be made sound by introducing a layer of auxiliary variables (labels) and forbidding their removal during preprocessing [3,4]. Concretely, a new variable l_i is introduced for each soft clause C_i prior to applying preprocessing. The original soft clause replaced by a hard clause $(C_i \vee l_i)$ with the restriction that the variable l_i may not be eliminated from the formula. After preprocessing, soft clauses $(\neg l_i)$ with the weights of the original clauses C_i are added to the formula.

Efficient integration of SAT-based preprocessing in LMHS is enabled by the observation that the MaxHS algorithm is sound even in cases where an assumption variable is shared between clauses or a clause contains more than one assumption variable [5]. This allows LMHS to re-use the variables l_i introduced by preprocessing in place of the auxiliary variables a_i, avoiding the introduction of a new layer of assumption variables for SAT-based core extraction within the main algorithm.

Following [6], we further avoid the addition of unnecessary auxiliary variables in LMHS by detecting variables in the original instance which can be reused already in the preprocessing phase. Any literal $l \in \{x, \neg x\}$ which occurs only

Algorithm 2. Enumeration of optimal solutions.

```
1: function ENUMERATE(F_h, F_s, c)
2:     K ← ∅, H ← ∅
3:     while true do
4:         while true do
5:             (sat, κ, τ) ← SOLVESAT(F_h ∪ (F_s \ H))
6:             if not sat then
7:                 K ← K ∪ {κ}
8:                 H ← SOLVEMCHS(K, c)
9:             else break
10:        if opt is undefined then opt ← cost(τ)
11:        if cost(τ) > opt then break
12:        else
13:            yield τ
14:            F_h ← F_h ∪ { ⋁_{τ(x)=1} ¬x ∨ ⋁_{τ(x)=0} x }
```

in a single unit soft clause $(\neg l)$ and some hard clauses $(C_1 \vee l), \ldots, (C_n \vee l)$ of the input instance is detected by simple pattern matching and re-used by the preprocessor and thereafter by the main algorithm. Such variables are introduced by, e.g., a straightforward encoding of group constraints [13].

Solution Enumeration. LMHS offers command-line options for enumerating MaxSAT solutions. The solver can enumerate the k best solutions or all optimal solutions. Enumeration can be based on variable assignments or satisfied clauses. In the latter case, only solutions which satisfy a unique set of soft clauses are considered. Solution enumeration in LMHS is implemented as Algorithm 2. The MaxHS algorithm is enclosed within the loop on Line 3. When the first solution is found, Line 10 records its cost as the optimal cost. On subsequent optimal solutions, Line 14 adds a single clause which forbids the latest obtained optimal solution. The termination condition on Line 11 is met when all optimal solutions have been found.

To enumerate unique solutions in terms of satisfied clauses, the refinement of $\mathcal{F}$ on Line 14 is replaced by adding the constraint $\sum_{C_i \in H} a_i - \sum_{C_i \in \mathcal{F}_s \setminus H} a_i < |H|$ to the hitting set IP, followed by a re-computation of the hitting set. A fixed number of best solutions can be found by modifying the condition of Line 11 to only break after enough solutions have been found. An application of the solution enumeration interface—and the incremental API described next—is presented in [19], where LMHS is used for MaxSAT to deriving cutting planes in an IP-based approach to learning optimal Bayesian network structures.

Incremental API. LMHS also implements a more general type of incrementality. Through a C or C++ API, the working formula can be incrementally extended with arbitrary clauses and the solver subsequently incrementally used for finding optimal solutions to the altered formula without starting search from scratch. In terms of Algorithm 2, operations performed through the API in effect replace Line 14. An overview of the interface follows.

- `reset` Resets the internal state of LMHS, allowing a new instance to be started.
- `initialize` Initializes LMHS and its components. Three variants of this method are offered. An instance can be initialized from a file, from clauses in memory, or as an empty instance to be built using the API.
- `getNewVariable` Requests a new variable from the internal SAT solver.
- `addHardClause` Adds a hard clause to the working MaxSAT instance.
- `addSoftClause` Adds a soft clause to the working MaxSAT instance. This automatically internally creates a blocking (auxiliary) variable for the clause. This variable is returned by the function in case the user wishes to make use of it. As a rule, the blocking variable created will always have a larger index number than the last variable created with `getNewVariable`.
- `addCoreConstraint` If a subset of soft clauses is known to be unsatisfiable, it can be explicitly added to the set of cores, expressed using the blocking variables of the soft clauses.
- `forbidLastModel` Internally creates a SAT constraint forbidding the previously found variable assignment.
- `forbidLastSolution` Internally creates an IP constraint forbidding the previously found set of satisfied soft clauses.
- `getOptimalSolution` Optimally solves the current MaxSAT instance.

Choice of SAT and IP Solvers. A lightweight interface between LMHS and its SAT solver component allows for flexibility in the choice of solver. Any solver which provides an assumption-based incremental interface can be integrated into LMHS by implementing a small interface class and making minor modifications to the build process. Interfaces for two such solvers, MiniSat 2.2 [12] and the inprocessing [15] SAT solver Lingeling [7], are included in the current release of LMHS. Similarly, LMHS was also designed to allow for the use of different IP solvers. Currently LMHS includes interfaces to the state-of-the-art commercial IP solver IBM CPLEX [14] and the open-source non-commercial IP solver SCIP [1].

Input Format. In addition to adhering to the DIMACS WCNF input format for weighted partial MaxSAT, LMHS also supports the use of floats (without preprocessing) in the input WCNF, i.e., MaxSAT with cost functions associating real-valued non-negative weights to clauses. Within the solver, the costs are handled by the IP solver.

4 Performance Overview

This section examines some interesting aspects of the performance of LMHS. We evaluate the solver on the complete set of 2209 crafted and industrial partial and weighted partial benchmarks of the 2015 MaxSAT Evaluation [2]. The experiments were run on machines with 32-GB memory and Intel Xeon E5540 processors under Ubuntu Linux 12.04. A per-instance time limit of 1800 seconds (30 min) was enforced. Figure 2 is a plot of the number of instances

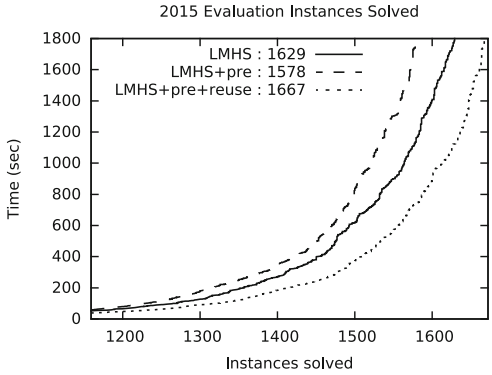

Fig. 2. Effect of integrated preprocessing on LMHS on 2015 MaxSAT evaluation instances.

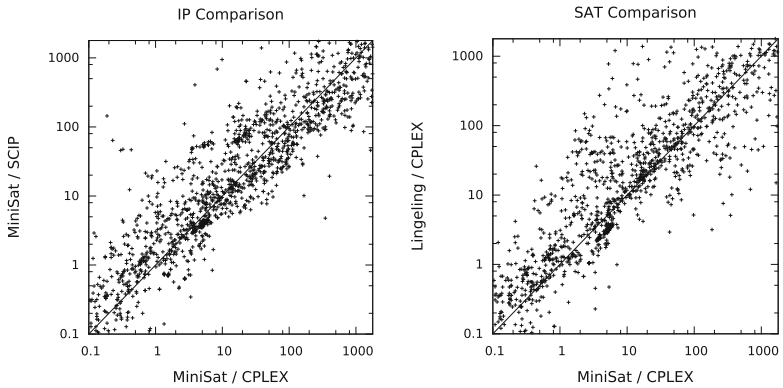

Fig. 3. A comparison of SAT (right) and IP (left) solver components within LMHS.

solved at different per-instance timeouts, showing the impact of integrating preprocessing into the solver by reusing auxiliary variables. It shows an interesting effect, in that the ordinary application of MaxSAT preprocessing to the instance (LMHS+pre) degrades solver performance compared to no preprocessing (LMHS), but the tighter integration of preprocessing by reusing variables (LMHS+pre+reuse) produces a clear improvement. While the reasons for this effect are not entirely clear at present, we suspect it to be at least in part due to the larger search space resulting from the added auxiliary variables. The extra layer of variables can also be detrimental in terms of potential additional constraints available for the IP solver; see [5, Example 1].

Figure 3 compares the use of the SCIP 3.0.1 IP solver to CPLEX 12.6.0 (left) and the MiniSat 2.2 solver to Lingeling (right) as the MIP and SAT components, respectively, in LMHS. The combination of CPLEX and MiniSat was mainly used during the development of LMHS, so these components can be expected to perform better in the default configuration. Figure 3 plots per-instance runtimes

for solved instances, and clearly shows better results with CPLEX and MiniSat. However, although SCIP and Lingeling results in worse performace overall compared to CPLEX and MiniSat, there is significant number of instances which they enable solving faster. This suggests that choosing a combination of a SAT solver and an IP solver on a per-instance basis could result in improved performance. Additionally, more in-depth analysis of which instances are best suited to each solver component could yield interesting further insights.

5 Availability and Conclusions

LMHS is competitive with the current state-of-the-art in MaxSAT solvers, recently having reached top positions in the 2015 MaxSAT Evaluation. LMHS integrates MaxSAT preprocessing into the solver. LMHS was designed to be flexible in allowing for integrating different SAT and IP solvers. The solver is open source and released under the MIT license at http://www.cs.helsinki.fi/group/coreo/lmhs/.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)
2. Argelich, J., Li, C.M., Manyá, F., Planes, J.: Max-SAT 2015: Tenth Max-SAT Evaluation (2015). http://www.maxsat.udl.cat/15/
3. Belov, A., Järvisalo, M., Marques-Silva, J.: Formula preprocessing in MUS extraction. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 108–123. Springer, Heidelberg (2013)
4. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for MaxSAT. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 96–111. Springer, Heidelberg (2013)
5. Berg, J., Saikko, P., Järvisalo, M.: Improving the effectiveness of SAT-based preprocessing for MaxSAT. In: Proceedings of IJCAI, pp. 239–245. AAAI Press (2015)
6. Berg, J., Saikko, P., Järvisalo, M.: Re-using auxiliary variables for MaxSAT preprocessing. In: Proceedings of ICTAI, pp. 813–820. IEEE (2015)
7. Biere, A.: Yet another local search solver and Lingeling and friends entering the SAT competition 2014. In: Proceedings of SAT Competition 2014, vol. B-2014-2, pp. 39–40. Department of Computer Science Series of Publications B, University of Helsinki (2014)
8. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)
9. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013)
10. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 247–262. Springer, Heidelberg (2013)

11. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
12. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
13. Heras, F., Morgado, A., Marques-Silva, J.: MaxSAT-based encodings for group MaxSAT. AI Commun. **28**(2), 195–214 (2015)
14. IBM ILOG: CPLEX optimizer 12.6.0 (2014). http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
15. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
16. Karp, R.M.: Implicit hitting set problems and multi-genome alignment. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 151–151. Springer, Heidelberg (2010)
17. Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer, Heidelberg (2012)
18. Moreno-Centeno, E., Karp, R.M.: The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. Oper. Res. **61**(2), 453–468 (2013)
19. Saikko, P., Malone, B., Järvisalo, M.: MaxSAT-based cutting planes for learning graphical models. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 347–356. Springer, Heidelberg (2015)