Journal on Satisfiability, Boolean Modeling and Computation 9 (2015) 83-88

MiFuMax—a Literate MaxSAT Solver

SYSTEM DESCRIPTION

Mikoláš Janota mikolas.janota@gmail.com
Microsoft Research, Cambridge
United Kingdom

Abstract

The main motivation behind the MaxSAT solver MiFuMax is twofold. It provides a baseline implementation of core-based algorithms for both weighted and unweighted MaxSAT. Such baseline implementation may serve for evaluation of evolving solvers. MiFuMax is written in literate programming and as such is instructive for anyone interested in learning about the implementation of modern core-based MaxSAT solvers. Despite its educative background, the solver has placed 1^{st} in the Unweighted Max-SAT-Industrial track of the 2013 MaxSAT Evaluation and it has been successfully applied in other research.

KEYWORDS: *MaxSAT, SAT, Literate programming*

Submitted December 2014; revised October 2015; published December 2015

1. Introduction

One of the most interesting problems in computer science is *formula satisfiability (SAT)*. Due to its NP completeness it had long been believed intractable. However, since the immense success of SAT solvers in the 90's, it is no longer so. Indeed, SAT solvers are commonly used in hardware/software checking, planning, and many other types of combinatorial problems.

In many real world scenarios it is often insufficient to find *a* solution but a solution with some optimal properties is sought. This is where MaxSAT solvers come into play. Given a formula in a conjunctive normal form (CNF), a MaxSAT solver finds a maximal satisfiable subset of clauses of the formula. This enables us to formulate optimization problems as MaxSAT problems (see e.g. [5]). This paper describes a MaxSAT solver MiFuMax[1], which solves MaxSAT by the *core-based* approach. In this approach, the solver tries to first satisfy all clauses in the formula and gradually *relaxes* those portions (*cores*) that are deemed as reasons for unsatisfiability. Throughout the computation, a SAT solver is used in a black-box fashion. MiFuMax uses algorithms stemming from the seminal paper of Fu&Malik [3] and it uses the SAT solver minisat2.2 [1], hence the name.

MiFuMax is written in *literate programming* [4], which enables interweaving written text with program code. MiFuMax is written in LaTeX and C^{++}, interleaved by the noweb system [10]. Like so, the interested reader can pick up a PDF retaining both the description and the code and read it as a scientific article[2]. Anyone interested in the gritty details of MiFuMax is welcome to do so. Here we will look at the most characteristic features and implementation details of the solver.

1. http://sat.inesc-id.pt/~mikolas/sw/mifumax/
2. Such PDF can be found at http://sat.inesc-id.pt/~mikolas/sw/mifumax/book.pdf.

2. Preliminaries

Throughout the paper, terminology commonly used in SAT and MaxSAT community will be used (cf. [5]). In particular, we are going to be solving problems containing *Boolean variables* (commonly denoted x,y, etc.). A *literal* is a variable or its negation, e.g. x, $\neg x$. A *clause* is a disjunction of literals (possibly none). A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. Since disjunction is associative, commutative, and idempotent, it is common to treat a clause as a set of literals. A formula in CNF is treated as a multiset of clauses. The reason for using a multiset is that in MaxSAT, we measure the number of satisfied clauses and hence multiple copies of one clause are not identical to a single clause. Note that the *empty clause*, i.e. the empty disjunction of literals, is semantically equivalent to *false*; the empty multiset of clauses is semantically equivalent *true*.

The input to a *MaxSAT problem* is a multiset of clauses and a solution is an assignment that maximizes the number of satisfied clauses.[3] MaxSAT has several variations. A simple extension is the *partial MaxSAT* where clauses are split into *hard* and *soft*. A solution is an assignment that satisfies all the hard clauses and maximizes the number of satisfied soft clauses. A further extension of MaxSAT is the *weighted MaxSAT*, where each clause is given a weight. Then, a solution is an assignment that maximizes the sum of the weights of satisfied clauses. Weighted MaxSAT also has an analogous *partial weighted MaxSAT* variant. Note that a partial (weighted) MaxSAT does not have a solution whenever the multiset of hard clauses is unsatisfiable, whereas (weighted) MaxSAT always has a solution.

Example 1. Consider the formula $(x \vee y) \wedge (y \vee z)$. If we wish to *minimize* the number of variables set to true, we construct the following partial MaxSAT problem. Let hard clauses be $\{x \vee y, y \vee z\}$ and soft clauses $\{\bar{x}, \bar{y}, \bar{z}\}$. The solution to this problem is $x = \text{false}, z = \text{false}, y = \text{true}$, unsatisfying one soft clause ($\bar{y}$). Note that this instance has only one solution but in general there may be many.

In this paper we assume that a SAT solver for a given formula ϕ in CNF returns a satisfying assignments to ϕ and returns an unsatisfiable multiset of clauses $\mathscr{C} \subseteq \phi$ otherwise. The multiset $\mathscr{C}$ is called an *unsatisfiable core of* ϕ. Whenever ϕ is clear from the context, we simply say core $\mathscr{C}$. Note that there is no requirement for the core to be minimal in any way. In particular, it can happen that the core is the whole formula. In practice, however, state-of-the-art SAT solvers return cores significantly smaller than the given formula. Nevertheless, any correct algorithm that relies on cores, must account for cases where the returned core contains redundant clauses. In pseudocode, a SAT solver will be modeled by the function $\texttt{SAT}(\phi)$ returning the triple $(\mathsf{outc}, \mu, \mathscr{C})$, where outc is true iff ϕ is satisfiable; if it is, μ is a satisfying assignment of ϕ; if it is not, the multiset of clauses $\mathscr{C}$ is an unsatisfiable core of ϕ. Since minisat2.2 does not return unsatisfiable cores directly, we show how MiFuMax computes them at the end of the following section.

3. Fu&Malik Algorithm for Unweighted MaxSAT

MiFuMax solves MaxSAT instances by the Fu&Malik algorithm [3], illustrated by Algorithm 1. The algorithm begins by trying to satisfy all the given clauses. If this is possible, the algorithm terminates and returns the obtained satisfying assignment. If, however, the

3. Sometimes only the number of satisfied clauses is required to be returned.

Algorithm 1: The Fu&Malik algorithm [3]

Input : Formula ϕ
Output: assignment μ satisfying the maximal multiset of clauses of ϕ

```
1 while true do
2     (outc, μ, 𝒞) ← SAT(φ)
3     if outc then return μ
4     remove any hard clauses from 𝒞
5     φ ← φ \ 𝒞 ∪ {r_C ∨ C | C ∈ 𝒞} ∪ Σ_{C∈𝒞} r_C ≤ 1
```

set of clauses ϕ is unsatisfiable, the algorithm transforms the formula. The motivation for the transformation is the following. If there is a satisfying assignments to some submultiset of ϕ, such assignment *cannot* satisfy all the clauses in the core $\mathscr{C}$. Hence, the algorithm adds to each clause $C \in \mathscr{C}$ a fresh variable r_C. This operation is called *relaxation* and the variable r_C is called a *relaxation variable*. A relaxed clause is trivially satisfied by setting r_C to true. As not to lose optimal solutions, the additional constraint $\sum_{C\in\mathscr{C}} r_C \leq 1$ is added, which permits us to satisfy at most one clause from the core through r_C. Note that this constraint on the cardinality of relaxation variables is not in CNF. A bevy of techniques exists that enable efficient encoding of such constraints into CNF; MiFuMax uses the *sequential counter encoding* [12]. The clauses encoding the constraint are treated as hard clauses and thus are never relaxed in the future.

Observe that a clause may be relaxed multiple times, i.e., once the formula is transformed by one iteration of the algorithm, the next iteration works on this transformed formula without relating to the original one. Other algorithms that relax a clause only once exist [8]. However, more complicated constraints are needed (rather than just at-most-one).

It is easy to modify Algorithm 1 so that it handles partial MaxSAT instances. One simply removes from any computed core any hard clauses (the instance has no solution if the core becomes empty). Like so only soft clauses may be relaxed.

MiFuMax's implementation follows Algorithm 1 rather closely. It maintains a list of clauses ϕ_H for hard clauses and a list of clauses ϕ_S for soft clauses. In each iteration of the loop of Algorithm 1 a new copy of a SAT solver is created and populated with ϕ_H and ϕ_S. The clauses in ϕ_S are modified by relaxations whereas the clauses in ϕ_H remain intact but are extended with new clauses representing the cardinality constraints. The following discusses some further implementation specifics.

Core computation. Since minisat2.2 does not directly return cores, MiFuMax calculates them using the *assumption-based method*. This method hinges on the *incremental interface* of minisat2.2 [1, 2]. This interface enables invoking the SAT solver with a set of literals called the *assumptions*. These assumptions can be understood as unit clauses that are temporarily added to the formula. To obtain a core, before we call the SAT solver, for a clause C we generate a fresh variable s_C, called the *control variable*. Instead of C, we give to the SAT solver the clause $\neg s_C \vee C$ and pass the literal s_C as an assumption when calling the SAT solver. If the formula is unsatisfiable, minisat2.2 provides us with the *final conflict clause* (conflict). That is a clause that contains only assumption literals and it must be satisfied if

Algorithm 2: WMSU1 algorithm [6]

Input : Formula ϕ
Output: assignment μ satisfying the maximal multiset of clauses of ϕ

while *true* **do**
  $(\mathsf{outc}, \mu, \mathscr{C}) \leftarrow \mathtt{SAT}(\phi)$
  if outc **then return** μ
  $w_m \leftarrow \min\{w \mid (w, C) \in \mathscr{C}\}$
  $R_{\mathscr{C}} \leftarrow \{(w_m, C \vee r_C) \mid (w, C) \in \mathscr{C}\}$
  $S_{\mathscr{C}} \leftarrow \{(w - w_m, C) \mid w > w_m, (w, C) \in \mathscr{C}\}$
  $\phi \leftarrow \phi \setminus \mathscr{C} \cup R_{\mathscr{C}} \cup S_{\mathscr{C}} \cup \sum_{C \in \mathscr{C}} r_C \leq 1$

the given formula is to be satisfied.[4] Hence, the core is calculated as $\{C \mid \neg s_C \in \mathsf{conflict}\}$. Since we are not interested in elements of the core if they are part of the hard clauses, hard clauses are never adorned with control variables and like so are automatically filtered out of the core.

Unit core optimization MiFuMax makes a small optimization not described in the original paper by Fu and Malik. Consider a core $\mathscr{C}$ with a single soft clause C (a *unit core*). What this means is that the conjunct $\phi_H \wedge C$ is unsatisfiable. In such case C, is not satisfied by *any* solution of the given MaxSAT problem. This permits us to simply remove the clause the clause C from the multiset of soft clauses. At the same time, since $\phi_H \wedge C$ is unsatisfiable (equivalently, $\phi_H \Rightarrow \neg C$), $\neg C$ can be added to the hard clauses, without affecting the satisfying assignments of ϕ_H. Hence, whenever the solver finds a unit core $\mathscr{C}$ containing a soft clause C, the following things happen in MiFuMax. 1) For any literal $l \in C$, the unit clause $\neg l$ is added to ϕ_H. 2) The clause $\neg s_C$ is added to the hard clauses to effectively remove the clause C. 3) The clause C is marked as removed so that s_C is not added to the assumptions in the next SAT calls. 4) The above changes made to ϕ_H are immediately repeated on the current copy of the SAT solver, and like so the SAT solver does not need to be rebuilt for the next iteration.

4. Weighted MaxSAT

To solve weighted MaxSAT, MiFuMax uses the algorithm by Manquinho et al. (commonly known as WMSU1) [6], whose pseudocode is shown in Algorithm 2. Weighted MaxSAT is very much similar to unweighted MaxSAT and the used algorithm is also similar. We will write (w, C) to denote a clause C with a weight w. One could solve a weighted MaxSAT by translating it to *unweighted* MaxSAT by creating w unweighted copies of C for each weighted clause (w, C). Let us pretend for a while that we do that and apply the unweighted Algorithm 1. Now we find some unweighted core $\mathscr{C}$. Let w_m be the smallest weight appearing in the weighted counterpart of the core. This means that in the unweighted version of the formula there is at least w_m copies of each of the clauses in $\mathscr{C}$. In

4. One may imagine that this clause is obtained by traditional clause learning [11] if the assumptions are the first decisions made by the solver.

another words, there is at least w_m disjoint copies of the core $\mathscr{C}$. What we could do is relax each of the copies separately right after finding the first copy of the core $\mathscr{C}$ without waiting for the other copies to be computed. This gives us one important optimization. However, we make another important observation and that is that each copy of some clause $C \in \mathscr{C}$ is satisfied (respectively unsatisfied) by the same set of assignments. Hence, if any solution to the unweighted MaxSAT satisfies some clause C, it will also satisfy its copies (and the other way around). This means, that the same relaxation variable can be used for all the w_m copies of each clause $C \in \mathscr{C}$.

These observations enable an algorithm that does not require explicitly creating all the different copies of clauses. This is shown in Algorithm 2. Whenever a core $\mathscr{C}$ is found, the algorithm computes the minimum weight w_m appearing in it (line 4). Subsequently, each clause $(w, C) \in \mathscr{C}$ is split into the clauses (w_m, C) and $(w - w_m, C)$ (lines 5,6). The clause with the weight w_m correspond to the w_m copies discussed above. The clause with weight $w - w_m$ corresponds to the surplus copies of that clause (if $w - w_m = 0$, the clause is ignored). In accordance with the discussion above, the clause (w_m, C) is relaxed. The relaxation variables are bound by the at-most-one constraint just as in the unweighted case.

The implementation of the algorithm WMSU1 in MiFuMax is similar to the implementation of the Fu&Malik algorithm. The main difference is that the number of soft clauses increases throughout the course of the algorithm, which slightly affects the bookkeeping. In order to split a clause (w, C) into the clauses $(w_m, C \vee r_C)$ and $(w - w_m, C)$, the solver changes the weight of the original clause into w_m and relaxes it. Then it creates a new soft clause C with the weight $w - w_m$; this step is skipped if $w = w_m$.

5. Conclusions and Future Work

This paper describes the solver MiFuMax, which is meant to demonstrate the basic functionality of the core-based approach to MaxSAT solving. While the solver is meant to be explanatory, it exhibits competitive performance. In fact, it has placed 1^{st} in the Unweighted Max-SAT-Industrial track of the 2013 MaxSAT Evaluation[5.] and performed rather well in the same track of 2014. The solver has also been applied in verification of Datalog programs [14] and it has served as the basis for the solver eva [9] (the winner of 2014's weighted partial industrial track).

A number of powerful techniques have been developed for MaxSAT (cf. [8]) it would be useful to add some of these to MiFuMax. In particular, *disjoint cores* [7] and *core trimming* [13]. One drawback of the Fu&Malik algorithm is that it might require addition of many relaxation variables. Other algorithms enable using a single relaxation variable per clause [7]; these would also be useful to include into MiFuMax.

Acknowledgments.

This work was supported by FCT grants POLARIS (PTDC/EIA-CCO/123051/2010) and AMOS (CMUP-EPB/TIC/0049/2013), INESC-ID's multiannual PIDDAC funding PEst-OE/EEI/LA0021/2013.

5. http://www.maxsat.udl.cat/13/results/index.html#ms-industrial

References

[1] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, pages 502–518, 2003.

[2] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, **89**(4):543–560, 2003. BMC '03, First International Workshop on Bounded Model Checking.

[3] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *SAT*, **4121** of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

[4] Donald E. Knuth. Literate programming. *Comput. J.*, **27**(2):97–111, 1984.

[5] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009.

[6] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *SAT*, **5584** of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[7] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe, DATE 2008*, pages 408–413. IEEE, 2008.

[8] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, **18**(4):478–534, 2013.

[9] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *AAAI*, pages 2717–2723, 2014.

[10] Norman Ramsey. Literate programming simplified. *IEEE Software*, **11**(5):97–105, 1994.

[11] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, **48**(5):506–521, 1999.

[12] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *CP*, **3709** of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

[13] Lintao Zhang and Sharad Malik. Extracting small unsatisfiable cores from unsatisfiable formula. In *Preliminary Proceedings of SAT*, 2003.

[14] Xin Zhang, Ravi Mangal, Radu Grigore, Mayur Naik, and Hongseok Yang. On abstraction refinement for program analyses in Datalog. In *PLDI*, pages 239–248. ACM, 2014.