A Modular Approach to MaxSAT Modulo Theories [*]

Alessandro Cimatti[1], Alberto Griggio[1],
Bastiaan Joost Schaafsma[1,2], and Roberto Sebastiani[2]

[1] FBK-IRST, Trento, Italy
[2] DISI, University of Trento, Italy

Abstract. In this paper we present a novel "modular" approach for (weighted partial) MaxSAT Modulo Theories. The main idea is to combine a lazy SMT solver with a purely-propositional (weighted partial) MaxSAT solver, by making them exchange information iteratively: the former produces an increasing set of theory lemmas which are used by the latter to progressively refine an approximation of the final subset of the soft clauses, which is eventually returned as output. The approach has several practical features. First, it is independent from the theories addressed. Second, it is simple to implement and to update, since both SMT and MaxSAT solvers can be used as blackboxes. Third, it can be interfaced with external MaxSAT and SMT solvers in a plug-and-play manner, so that to benefit for free of tools which are or will be made available.
We have implemented our approach on top of the MATHSAT5 SMT solver and of a selection of external MaxSAT solvers, and we have evaluated it by means of an extensive empirical test on SMT-LIB benchmarks. The results confirm the validity and potential of this approach.

1 Introduction

MaxSAT [19] is the problem of determining the maximum number of clauses, of a given Boolean formula, that can be satisfied by some assignment. Its *weighted* and *partial* variants allow to associate fixed weights to clauses, and to search only for solutions that satisfy a given subset of the clauses. (In this paper, unless otherwise specified, by "MaxSAT" we always consider the general case of weighted partial MaxSAT; thus, we often omit the adjectives "weighted" and "partial".) In recent years, the solvers for MaxSAT have demonstrated substantial improvements [20, 6, 17, 18, 21, 3], and have now important practical applications (e.g. Formal Verification, Automatic Test Pattern Generation, Field Programmable Gate Array routing).

The MaxSAT problems can be generalized from the Boolean case to the case of Satisfiability Modulo Theories (SMT) [8], where first order formulas are interpreted with respect to some (combinations of) background theories. Theories of interest are,

[*] We are very grateful to Carlos Ansótegui, Bruno Dutertre and Leonardo de Moura for providing to us precious information about their respective solvers. Alberto Griggio is supported by Provincia Autonoma di Trento and the European Community's FP7/2007-2013 under grant agreement Marie Curie FP7 - PCOFUND-GA-2008-226070 "progetto Trentino", project ADAPTATION; Bas Schaafsma and Roberto Sebastiani are supported in part by Semiconductor Research Corporation under GRC Research Project 2012-TJ-2266 WOLF.

e.g., those of bit vectors ($\mathcal{BV}$), of arrays ($\mathcal{AR}$), of linear arithmetic ($\mathcal{LA}$) on the rationals ($\mathcal{LA}(\mathbb{Q})$) or on the integers ($\mathcal{LA}(\mathbb{Z})$).

Because of the increase in expressiveness of SMT, the MaxSAT Modulo Theory problem (MaxSMT hereafter) has many important applications (e.g., formal verification of timed & hybrid systems and of parametric systems, planning with resources, radio frequency assignment problems.) However, MaxSMT—and, more generally, the optimization problems in SMT— have received relatively little attention in the literature. To some extent, this can be explained with the technical difficulties associated with the combination of two non-trivial components, namely an SMT engine (that requires the integration of constraints into SAT) and a MaxSAT optimization procedure.

In this paper, we propose a novel and comprehensive approach to (weighted partial) MaxSMT. The approach is highly modular, in that it combines, as black boxes, two components: (i) a lazy SMT solver, and (ii) a purely-propositional (weighted partial) MaxSAT solver. During the search, these two components exchange information iteratively: the SMT solver produces an increasing set of theory lemmas, which are used by the MaxSAT solver to progressively refine an approximation of the final subset of the soft clauses, which is eventually returned as output.

Basically, the SMT solver is used to dynamically lift the suitable amount of theory information to the Boolean level, where the MaxSAT solver performs the optimization process. We call the approach *Lemma-Lifting (LL)*, similarly to the LL approach for the extraction of unsatisfiable cores in SMT [13].

The approach has several interesting features. First, it is independent from the theories addressed: the lemmas returned by the SMT solver during the search are abstracted into Boolean formulas before being passed to the MaxSAT solver. Second, the LL algorithm is general and simple to implement: it imposes no restriction on the MaxSAT solver, while the only requirement on the SMT solver is that it is able to return the lemmas constructed during search. Third, the LL algorithm can be realized by interfacing external MaxSAT and SMT solvers in a plug-and-play manner. In this way, we can use all the available approaches and tools, and benefit of future advances in lazy SMT and MaxSAT technology.

We have proved the formal properties of the LL MaxSMT algorithm. We implemented LL on top of the MATHSAT5 SMT solver [12], and of a selection of external MaxSAT tools. We have evaluated and compared the performances of the various LL configurations, and of every MaxSMT or MaxSMT-like solver we are aware of, by means of an extensive empirical test on MaxSMT-modified SMT-LIB benchmarks. The results confirm the validity and potential of this approach.

Content. The paper is organized as follows. After having provided some background knowledge on SMT, MaxSAT and MaxSMT in §2, we present and discuss our new approach and algorithm in §3. We proceed with a discussion of related work in §4. In §5 we present and comment empirical tests. We conclude and suggest some future developments in §6.

2 Background

Terminology and notation. We consider some decidable first-order theory $\mathcal{T}$ (or a combination $\bigcup_i \mathcal{T}_i$ of theories). We call *$\mathcal{T}$-atom* (resp. -literal, -clause, -formula) a ground atomic formula (resp. literal, clause, formula) in $\mathcal{T}$. (Notice that a Boolean atom can be seen as a subcase of $\mathcal{T}$-atom, etc.) We distinguish the space of $\mathcal{T}$-formulas ($\mathcal{T}$) from that of plain Boolean formulas ($\mathcal{B}$) by denoting them with the "$^{\mathcal{T}}$" and the "$^{\mathcal{B}}$" superscripts respectively; we use no superscript when we make no such distinction. Given a $\mathcal{T}$-formula (-clause, -literal, -assignment etc.) $\varphi^{\mathcal{T}}$, we call *Boolean abstraction* of $\varphi^{\mathcal{T}}$ the formula $\varphi^{\mathcal{B}} \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}(\varphi^{\mathcal{T}})$ obtained by rewriting each non-Boolean $\mathcal{T}$-atom in $\varphi^{\mathcal{T}}$ into a fresh Boolean atom; vice versa, $\varphi^{\mathcal{T}} \stackrel{\text{def}}{=} \mathcal{B}2\mathcal{T}(\varphi^{\mathcal{B}}) \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}^{-1}(\varphi^{\mathcal{B}})$ is the *refinement* of $\varphi^{\mathcal{B}}$. (To this extent, if not otherwise specified, when some symbol $\langle sym \rangle$ is used with both the '$^{\mathcal{T}}$" and the "$^{\mathcal{B}}$" superscripts, then $\langle sym \rangle^{\mathcal{B}}$ denotes the Boolean abstraction of $\langle sym \rangle^{\mathcal{T}}$, and vice versa.) We say that a truth assignment $\mu^{\mathcal{T}}$ *propositionally satisfies* $\varphi^{\mathcal{T}}$, written $\mu^{\mathcal{T}} \models_p \varphi^{\mathcal{T}}$, iff $\mu^{\mathcal{B}} \models \varphi^{\mathcal{B}}$.

In both the $\mathcal{T}$- and $\mathcal{B}$- spaces, we assume all formulas are in CNF, and we represent them as sets of clauses; we represent truth assignments as sets of literals. The symbols $\varphi_{...}^{...}$, $\psi_{...}^{...}$, $\phi_{...}^{...}$ denote formulas, and $\mu_{...}^{...}$, $\eta_{...}^{...}$ denote truth assignments, regardless their subscripts or superscripts. A *weighted clause* is a clause C which is augmented with a value $w \in \mathbb{N} \cup \{+\infty\}$, which is called the *weight* of C, denoted by $\mathsf{Weight}(C)$; a weighted clause is called *hard,* iff its weight is $+\infty$, *soft,* otherwise. Sets of hard and soft clauses are denoted with the subscript $._h$ and $._s$ respectively. $\mathsf{Weight}(\psi_s)$ denotes the sum of the weights of the clauses in ψ_s.

2.1 Satisfiability Modulo Theories

We call a *theory solver for $\mathcal{T}$*, $\mathcal{T}$-Solver, a tool able to decide the $\mathcal{T}$-satisfiability of a conjunction/set $\mu^{\mathcal{T}}$ of $\mathcal{T}$-literals. If $\mu^{\mathcal{T}}$ is $\mathcal{T}$-unsatisfiable, then $\mathcal{T}$-Solver returns UNSAT and the subset η of $\mathcal{T}$-literals in $\mu^{\mathcal{T}}$ which was found $\mathcal{T}$-unsatisfiable; (η is hereafter called a *$\mathcal{T}$-conflict set,* and $\neg\eta$ a *$\mathcal{T}$-conflict clause*.) if $\mu^{\mathcal{T}}$ is $\mathcal{T}$-satisfiable, then $\mathcal{T}$-Solver returns SAT; it may also be able to return some unassigned $\mathcal{T}$-literal $l \notin \mu^{\mathcal{T}}$s.t. $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, where $\{l_1, ..., l_n\} \subseteq \mu^{\mathcal{T}}$. We call this process *$\mathcal{T}$-deduction* and $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ a *$\mathcal{T}$-deduction clause*. Notice that $\mathcal{T}$-conflict and $\mathcal{T}$-deduction clauses are valid in $\mathcal{T}$. We call them *$\mathcal{T}$-lemmas*.

In a lazy SMT($\mathcal{T}$) solver, the Boolean abstraction $\varphi^{\mathcal{B}}$ of the input formula φ is given as input to a CDCL SAT solver, and whenever a satisfying assignment $\mu^{\mathcal{B}}$ is found s.t. $\mu^{\mathcal{B}} \models \varphi^{\mathcal{B}}$, the corresponding set of $\mathcal{T}$-literals $\mu^{\mathcal{T}}$ is fed to the $\mathcal{T}$-Solver; if $\mu^{\mathcal{T}}$ is found $\mathcal{T}$-consistent, then φ is $\mathcal{T}$-consistent; otherwise, $\mathcal{T}$-Solver returns the $\mathcal{T}$-conflict set η causing the inconsistency, so that the clause $\neg\eta^{\mathcal{B}}$ (the Boolean abstraction of $\neg\eta$) is used to drive the backjumping and learning mechanism of the SAT solver.

Important optimizations are *early pruning* and *$\mathcal{T}$-propagation*: the $\mathcal{T}$-Solver is invoked also on an intermediate assignment $\mu^{\mathcal{T}}$: if it is $\mathcal{T}$-unsatisfiable, then the procedure can backtrack; if not, and if the $\mathcal{T}$-Solver is able to perform a $\mathcal{T}$-deduction $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, then l can be unit-propagated, and the $\mathcal{T}$-deduction clause $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ can be used in backjumping and learning. Another technique is *static learning*, where

$\mathcal{T}$-lemmas expressing "obvious" constraints on $\mathcal{T}$-atoms occurring in the input formula (e.g. mutual-exclusion, transitivity constraints) are learned a priori.

The above schema is a coarse abstraction of the procedures underlying all the state-of-the-art lazy SMT tools. The interested reader is pointed to, e.g., [23, 8] for details and further references.

2.2 MaxSAT

A *(weighted partial)* [1] *MaxSAT formula* is a set of weighted clauses in the form $\varphi^{\mathcal{B}} \stackrel{\text{def}}{=} \varphi_h^{\mathcal{B}} \cup \varphi_s^{\mathcal{B}}$, s.t. $\varphi_h^{\mathcal{B}}$ and $\varphi_s^{\mathcal{B}}$ are sets of hard and soft clauses respectively. A *MaxSAT problem* consists in finding a maximum-weight clause set $\psi_s^{\mathcal{B}}$ s.t. $\psi_s^{\mathcal{B}} \subseteq \varphi_s^{\mathcal{B}}$ and $\varphi_h^{\mathcal{B}} \cup \psi_s^{\mathcal{B}}$ is satisfiable. (Notice that such $\psi_s^{\mathcal{B}}$ is not unique in general.) $\mathsf{MaxSAT}(\varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$ denotes a function computing one such $\psi_s^{\mathcal{B}}$, and $\mathsf{MaxWeight}(\varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$ denotes $\mathsf{Weight}(\psi_s^{\mathcal{B}})$.

Notice that $\mathsf{Weight}(C^{\mathcal{B}})$ can be considered as the "cost" of non-satisfying the soft clause $C^{\mathcal{B}}$, and MaxSAT can be seen as the problem of minimizing such cost over all the soft clauses. To this extent, a *MaxSAT Solver* is a function s.t. $\mathsf{MaxSAT}(\varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$ returns a maximum-weight clause set $\psi_s^{\mathcal{B}}$ s.t. $\psi_s^{\mathcal{B}} \subseteq \varphi_s^{\mathcal{B}}$ and $\varphi_h^{\mathcal{B}} \cup \psi_s^{\mathcal{B}}$ is satisfiable.

The MaxSAT problem can be generalized to the case in which $\varphi_h^{\mathcal{B}}$ and $\varphi_s^{\mathcal{B}}$ are sets of arbitrary formulas rather than sets of single clauses. [2] Let $\lambda_s \stackrel{\text{def}}{=} \{S_i\}_i$ be a set of fresh selection variables, one for each constraint $\phi_i^{\mathcal{B}}$ in $\varphi_s^{\mathcal{B}}$, let $\varphi_s'^{\mathcal{B}} \stackrel{\text{def}}{=} \{\neg S_i \vee \phi_i^{\mathcal{B}} \mid \phi_i^{\mathcal{B}} \in \varphi_s^{\mathcal{B}}\}$, and let $\psi_h^{\mathcal{B}}$ be the set of clauses resulting from conversion of $\varphi_h^{\mathcal{B}} \cup \varphi_s'^{\mathcal{B}}$ into CNF. Thus the generalized MaxSAT problem $(\varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$ can be reduced to a standard MaxSAT problem on the sets of clauses $(\psi_h^{\mathcal{B}}, \lambda_s)$, in which all soft clauses are unit clauses.

Current state-of-the-art MaxSAT solvers can be roughly divided into 3 categories. Solvers based on branch & bound, such as [20, 17], employ specialized inference rules while performing a standard branch and bound search for $\mathsf{MaxWeight}(\varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$. Iterative solvers, like e.g. [6], work by adding to each soft clause $C_j^{\mathcal{B}} \in \varphi_s^{\mathcal{B}}$ a fresh literal R_j (called a *relaxation literal*), and by imposing bounds on the number of relaxation literals that can be assigned to true, using cardinality constraints. The space of such bounds is typically explored using binary search. Finally, core-guided solvers, such as e.g. [18, 21], improve upon iterative solvers by exploiting unsatisfiable cores to decide if/when to add a relaxation literal to a soft clause, and to minimize the number of cardinality constraints needed.

2.3 MaxSAT Modulo Theories and SMT with Cost Optimization

The MaxSAT problem generalizes straightforwardly to SMT level. Given a background theory $\mathcal{T}$ as before, a *(weighted partial) MaxSAT Modulo Theories (MaxSMT) formula* is a set of weighted $\mathcal{T}$-clauses in the form $\varphi^{\mathcal{T}} \stackrel{\text{def}}{=} \varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}$. A *MaxSAT Modulo Theories*

[1] A MaxSAT formula is not "weighted" iff $\mathsf{Weight}(C_j^{\mathcal{B}}) = 1$ for every $C_j^{\mathcal{B}} \in \varphi_s^{\mathcal{B}}$, and it is not "partial" iff $\varphi_h^{\mathcal{B}}$ is empty. Hereafter, unless otherwise specified, we consider the general case ignoring this distinction, hence dropping the adjectives "weighted" and "partial".

[2] This includes also the so-called *Block MaxSAT* problem, where each (weighted) soft constraint is itself a conjunctions of clauses, representing a "block" of clauses subject to the same weight, s.t. it suffices to violate one such clause to pay the cost of the constraint.

(MaxSMT) problem consists in finding a maximum-weight clause set $\psi_s^{\mathcal{T}}$ s.t. $\psi_s^{\mathcal{T}} \subseteq \varphi_s^{\mathcal{T}}$ and $\varphi_h^{\mathcal{T}} \cup \psi_s^{\mathcal{T}}$ is $\mathcal{T}$-satisfiable. As with the Boolean case, $\mathsf{MaxSMT}(\varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}})$ denotes a function computing one such $\psi_s^{\mathcal{T}}$, and $\mathsf{MaxWeight}(\varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}})$ denotes $\mathsf{Weight}(\psi_s^{\mathcal{T}})$. (The same considerations and conventions on "weighted", "partial", and "generalized" MaxSAT in §2.2 hereafter apply for MaxSMT.)

Importantly, a MaxSMT problem can be encoded into an SMT problem with cost minimization $\langle \varphi^{\mathcal{T}'}, \mathsf{cost}\rangle$, either with Pseudo-Boolean (PB) cost functions [22, 10]:

$$\varphi^{\mathcal{T}'} = \varphi_h^{\mathcal{T}} \cup \bigcup_{C_j^{\mathcal{T}} \in \varphi_s^{\mathcal{T}}} \{(A_j \vee C_j^{\mathcal{T}})\}; \;\; \mathsf{cost} \overset{\text{def}}{=} \sum_{C_j^{\mathcal{T}} \in \varphi_s^{\mathcal{T}}} w_j \cdot A_j \tag{1}$$

where $w_j \overset{\text{def}}{=} \mathsf{Weight}(C_j^{\mathcal{T}})$ and the A_j's are fresh Boolean atoms, or with $\mathcal{LA}$ cost functions [22, 24]:

$$\varphi^{\mathcal{T}'} = \varphi_h^{\mathcal{T}} \cup \bigcup_{C_j^{\mathcal{T}} \in \varphi_s^{\mathcal{T}}} (\{(A_j \vee C_j^{\mathcal{T}}), (\neg A_j \vee x_j = w_j), (A_j \vee x_j = 0)\});$$

$$\mathsf{cost} \overset{\text{def}}{=} \sum_{C_j^{\mathcal{T}} \in \varphi_s^{\mathcal{T}}} x_j. \tag{2}$$

where the x_j's are $\mathcal{LA}$ variables.

3 A Novel Modular MaxSMT Algorithm

In what follows, we consider a MaxSMT problem $\varphi^{\mathcal{T}} \overset{\text{def}}{=} \varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}$, and w_{max} denotes $\mathsf{MaxWeight}(\varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}})$. The symbols $\Theta^{\mathcal{T}}$ and $\Theta_i^{\mathcal{T}}$ denote sets of $\mathcal{T}$-lemmas on $\mathcal{T}$-atoms occurring in $\varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}$, whilst $\Theta_*^{\mathcal{T}}$ denotes the set of *all* such $\mathcal{T}$-lemmas.

Observe that $\Theta_*^{\mathcal{T}}$ is a finite set, since $\Theta^{\mathcal{T}}$, $\Theta_i^{\mathcal{T}}$ and $\Theta_*^{\mathcal{T}}$ are defined to be sets of $\mathcal{T}$-lemmas containing only atoms in the input formula. In general, modern SMT solvers might introduce new atoms during search, which can thus appear in some $\mathcal{T}$-lemmas. This scenario is not considered here to keep the presentation simple. However, it can be covered under the additional assumption that $\mathcal{T}$-lemmas are generated from a finite set of atoms, which is typically the case for modern SMT solvers (see e.g. [9, 7]).

3.1 The Basic Algorithm

Algorithm 1 reports a "modular" procedure for MaxSMT. Intuitively, an SMT and a MaxSAT solver are used as guided enumerators of, respectively: [3]

– a finite sequence of $\mathcal{T}$-lemma sets $\Theta_0^{\mathcal{T}}, \Theta_1^{\mathcal{T}}, ..., \Theta_n^{\mathcal{T}}$ s.t. $\Theta_0^{\mathcal{T}} = \emptyset$,

$$\Theta_0^{\mathcal{T}} \subseteq \Theta_1^{\mathcal{T}} \subset \Theta_2^{\mathcal{T}} \subset ... \subset \Theta_n^{\mathcal{T}}, \tag{3}$$

$$\Theta_n^{\mathcal{T}} \subseteq \Theta_*^{\mathcal{T}}, \tag{4}$$

[3] When referring to Algorithm 1, the index "$._i$" in $\Theta_i^{\mathcal{T}}$, $\psi_{s,i}^{\mathcal{T}}$ etc. refers to the values of $\Theta^{\mathcal{T}}$, $\psi_s^{\mathcal{T}}$ etc. at the end of the i-th cycle in the while loop.

Algorithm 1 A Lemma-Lifting procedure for $\mathsf{MaxSMT}(\varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}})$

```
Input:
    $\varphi_h^{\mathcal{T}}$: a set of hard $\mathcal{T}$-clauses;
    $\varphi_s^{\mathcal{T}}$: a set of (weighted) soft $\mathcal{T}$-clauses;
Output:
    a maximum-weight set of soft $\mathcal{T}$-clauses $\psi_s^{\mathcal{T}}$ s.t. $\psi_s^{\mathcal{T}} \subseteq \varphi_s^{\mathcal{T}}$ and $\varphi_h^{\mathcal{T}} \cup \psi_s^{\mathcal{T}}$ is $\mathcal{T}$-satisfiable

 1: $\langle \varphi_h^{\mathcal{B}}, \varphi_s^{\mathcal{B}} \rangle \leftarrow \mathcal{T}2\mathcal{B}\ (\langle \varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}} \rangle)$;
 2: $\Theta^{\mathcal{T}} \leftarrow \emptyset$;
 3: $\psi_s^{\mathcal{T}} \leftarrow \varphi_s^{\mathcal{T}}$;
 4: while (SMT.Solve $(\varphi_h^{\mathcal{T}} \cup \psi_s^{\mathcal{T}} \cup \Theta^{\mathcal{T}})$ = UNSAT) do
 5:     $\Theta^{\mathcal{T}} \leftarrow \Theta^{\mathcal{T}} \cup$ SMT.GetTLemmas ();
 6:     $\Theta^{\mathcal{B}} \leftarrow \mathcal{T}2\mathcal{B}\ (\Theta^{\mathcal{T}})$;
 7:     $\psi_s^{\mathcal{B}} \leftarrow \mathsf{MaxSAT}(\varphi_h^{\mathcal{B}} \cup \Theta^{\mathcal{B}}, \varphi_s^{\mathcal{B}})$;
 8:     $\psi_s^{\mathcal{T}} \leftarrow \mathcal{B}2\mathcal{T}\ (\psi_s^{\mathcal{B}})$;
 9: end while
10: return $\psi_s^{\mathcal{T}}$;
11:
12: SMT.Solve $(\varphi^{\mathcal{T}})$ checks whether $\varphi^{\mathcal{T}}$ is $\mathcal{T}$-satisfiable
13: SMT.GetTLemmas () returns the $\mathcal{T}$-lemmas computed by the latest call to SMT.Solve
```

which progressively rule out all the $\mathcal{T}$-unsatisfiable truth assignments which propositionally satisfy $\varphi_h^{\mathcal{T}}$ and some subset $\psi_{s,i}^{\mathcal{T}}$ of $\varphi_s^{\mathcal{T}}$ s.t. $\mathsf{Weight}(\psi_{s,i}^{\mathcal{T}}) > w$;

– (the Boolean abstraction of) a finite sequence of soft-clause sets $\psi_{s,0}^{\mathcal{T}}, ..., \psi_{s,i}^{\mathcal{T}}, ...\psi_{s,n}^{\mathcal{T}}$ where $\psi_{s,0}^{\mathcal{T}} = \varphi_s^{\mathcal{T}}$, $\psi_{s,i}^{\mathcal{T}} \subseteq \varphi_s^{\mathcal{T}}$ for every i, $\psi_{s,n}^{\mathcal{T}} = \mathsf{MaxSMT}(\varphi_h^{\mathcal{T}}, \varphi_s^{\mathcal{T}})$, and

$$\mathsf{Weight}(\psi_{s,n}^{\mathcal{T}}) \leq ... \leq \mathsf{Weight}(\psi_{s,i+1}^{\mathcal{T}}) \leq \mathsf{Weight}(\psi_{s,i}^{\mathcal{T}}) \leq ... \,. \tag{5}$$

$$\mathsf{MaxWeight}(\varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}) = \mathsf{Weight}(\psi_{s,n}^{\mathcal{T}}). \tag{6}$$

Notice that neither $\psi_{s,i+1}^{\mathcal{T}} \subseteq \psi_{s,i}^{\mathcal{T}}$ nor $\mathsf{Weight}(\psi_{s,i+1}^{\mathcal{T}}) < \mathsf{Weight}(\psi_{s,i}^{\mathcal{T}})$ hold in general.

Each $\Theta_{i+1}^{\mathcal{T}}$ results from adding to $\Theta_i^{\mathcal{T}}$ the $\mathcal{T}$-lemmas computed by an SMT solver to prove the $\mathcal{T}$-unsatisfiability of $\varphi_h^{\mathcal{T}} \cup \psi_{s,i}^{\mathcal{T}} \cup \Theta_i^{\mathcal{T}}$. Each $\psi_{s,i}^{\mathcal{T}}$ is obtained by invoking a MaxSAT solver on the Boolean abstraction of $\varphi_h^{\mathcal{T}} \cup \Theta_i^{\mathcal{T}}$ and $\varphi_s^{\mathcal{T}}$ as hard and soft component respectively.

The termination, correctness, and completeness of Algorithm 1 is formally proved in [11]. Intuitively, at every loop $i > 0$ s.t. $\varphi_h^{\mathcal{T}} \cup \psi_{s,i}^{\mathcal{T}} \cup \Theta_i^{\mathcal{T}}$ is found $\mathcal{T}$-unsatisfiable by SMT.Solve, since its Boolean abstraction $\varphi_h^{\mathcal{B}} \cup \psi_{s,i}^{\mathcal{B}} \cup \Theta_i^{\mathcal{B}}$ is satisfiable by construction of $\psi_{s,i}^{\mathcal{B}}$, then SMT.GetTLemmas returns at least one new $\mathcal{T}$-lemma; thus (3) holds, (4) holds by definition of $\Theta_*^{\mathcal{T}}$, hence (5) holds by construction of $\psi_{s,\cdot}^{\mathcal{B}}$. By (3), (4), and since $\Theta_*^{\mathcal{T}}$ is finite and it contains all the possible theory information related to $\varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}$, we have that, for some loop index n, $\Theta_n^{\mathcal{T}} \subseteq \Theta_*^{\mathcal{T}}$ and $\Theta_n^{\mathcal{T}}$ contains all $\mathcal{T}$-lemmas which rule out all $\mathcal{T}$-inconsistent truth assignments propositionally satisfying $\varphi_h^{\mathcal{B}} \cup \psi_{s,n}^{\mathcal{B}}$. Then $\varphi_h^{\mathcal{T}} \cup \psi_{s,n}^{\mathcal{T}} \cup \Theta_n^{\mathcal{T}}$ is $\mathcal{T}$-satisfiable, because $\varphi_h^{\mathcal{B}} \cup \psi_{s,n}^{\mathcal{B}} \cup \Theta_n^{\mathcal{B}}$ is satisfiable by construction of $\psi_{s,n}^{\mathcal{B}}$, so that the procedure terminates. From this, it is easy to show that (6) holds.

Notice that, in general, in the call $\mathsf{SMT.Solve}\ (\varphi_h^{\mathcal{T}} \cup \psi_s^{\mathcal{T}} \cup \Theta^{\mathcal{T}})$ the "$\cup\ \Theta^{\mathcal{T}}$" element is not necessary from the logic viewpoint, but it prevents the SMT solver to re-generate from scratch previously-computed $\mathcal{T}$-lemmas in $\Theta^{\mathcal{T}}$.

Example 1. Let $\varphi_h^{\mathcal{T}}$, $\varphi_s^{\mathcal{T}}$ be as follows (values $[v]$ denote clause weights):

$$\varphi_h^{\mathcal{T}} \overset{\text{def}}{=} \emptyset \qquad\qquad \varphi_h^{\mathcal{B}} \overset{\text{def}}{=} \emptyset$$

$$\varphi_s^{\mathcal{T}} \overset{\text{def}}{=} \left\{ \begin{array}{l} C_0 : ((x \le 0))\ [4] \\ C_1 : ((x \le 1))\ [3] \\ C_2 : ((x \ge 2))\ [2] \\ C_3 : ((x \ge 3))\ [6] \end{array} \right\} \varphi_s^{\mathcal{B}} \overset{\text{def}}{=} \left\{ \begin{array}{c} (A_0)\ [4] \\ (A_1)\ [3] \\ (A_2)\ [2] \\ (A_3)\ [6] \end{array} \right\} where : \begin{array}{r} A_0 \overset{\text{def}}{=} (x \le 0), \\ A_1 \overset{\text{def}}{=} (x \le 1), \\ A_2 \overset{\text{def}}{=} (x \ge 2), \\ A_3 \overset{\text{def}}{=} (x \ge 3). \end{array}$$

Notice that the set of all possible $\mathcal{T}$-lemmas on the $\mathcal{T}$-atoms of $\varphi_h^{\mathcal{T}} \cup \varphi_s^{\mathcal{T}}$ is:

$$\Theta_*^{\mathcal{T}} = \left\{ \begin{array}{l} \theta_1 :\ (\neg(x \le 0) \vee (x \le 1)) \\ \theta_2 :\ (\neg(x \ge 3) \vee (x \ge 2)) \\ \theta_3 :\ (\neg(x \le 0) \vee \neg(x \ge 2)) \\ \theta_4 :\ (\neg(x \le 0) \vee \neg(x \ge 3)) \\ \theta_5 :\ (\neg(x \le 1) \vee \neg(x \ge 2)) \\ \theta_6 :\ (\neg(x \le 1) \vee \neg(x \ge 3)) \end{array} \right\} \quad \Theta_*^{\mathcal{B}} = \left\{ \begin{array}{c} (\neg A_0 \vee A_1) \\ (\neg A_3 \vee A_2) \\ (\neg A_0 \vee \neg A_2) \\ (\neg A_0 \vee \neg A_3) \\ (\neg A_1 \vee \neg A_2) \\ (\neg A_1 \vee \neg A_3) \end{array} \right\}$$

Then, one possible execution of the algorithm is:

i	$\Theta_i^{\mathcal{T}}$	$\psi_{s,i}^{\mathcal{T}}$	Weight$(\psi_{s,i}^{\mathcal{T}})$	$SMT(\varphi_h^{\mathcal{T}} \cup \psi_{s,i}^{\mathcal{T}} \cup \Theta_i^{\mathcal{T}})$
0	$\{\}$	$\{C_0, C_1, C_2, C_3\}$	15	UNSAT
1	$\{\theta_4\}$	$\{C_1, C_2, C_3\}$	11	UNSAT
2	$\{\theta_4, \theta_6\}$	$\{C_0, C_1, C_2\}$	9	UNSAT
3	$\{\theta_4, \theta_6, \theta_3\}$	$\{C_2, C_3\}$	8	SAT

from which $\psi_s^{\mathcal{T}} = \{C_2, C_3\}$ and Weight$(\psi_s^{\mathcal{T}}) = 8$. A faster execution (which may be obtained, e.g., by enforcing the generation of extra $\mathcal{T}$-lemmas in the SMT solver) is:

i	$\Theta_i^{\mathcal{T}}$	$\psi_{s,i}^{\mathcal{T}}$	Weight$(\psi_{s,i}^{\mathcal{T}})$	$SMT(\varphi_h^{\mathcal{T}} \cup \psi_{s,i}^{\mathcal{T}} \cup \Theta_i^{\mathcal{T}})$
0	$\{\}$	$\{C_0, C_1, C_2, C_3\}$	15	UNSAT
1	$\{\theta_1, \theta_2, \theta_5\}$	$\{C_2, C_3\}$	8	SAT

◇

3.2 Optimizations

Algorithm 1 is very simple in principle, and it can be implemented using an SMT solver and a MaxSAT solver as black boxes in a plug-and-play manner. [4] Moreover, this allows for benefiting for free of any advanced tool available from the shelf, or for choosing the most suitable tools for a given problem.

Under the hypothesis of using the two solvers as black boxes, we consider some implementation issues which may further improve its efficiency.

[4] Provided that the SMT solver, like MATHSAT5, offers a way of retrieving the set of $\mathcal{T}$-lemmas which it used to prove the $\mathcal{T}$-inconsistency of the input formula, or, like most lazy SMT solvers, it can provide an SMT resolution proof, from which the latter set can be extracted.

Incrementality of MaxSAT**.** Since MaxSAT is invoked sequentially on incremental sets of hard clauses and on the same set of soft ones, it is natural to conjecture that having an incremental implementation of MaxSAT, which "remembers" the status of the search from call to call, should improve the efficiency of the overall procedure.

Reuse of SMT calls. $\mathsf{SMT.Solve}$ is not invoked incrementally in the classic "push-and-pop" sense because —apart from the fact that $\psi^{\mathcal{T}}_{s,i} \subseteq \varphi^{\mathcal{T}}_s$ for every i— there is no set-theoretic relation between the $\psi^{\mathcal{T}}_{s,i}$'s. However, it is possible to use SMT solving *under assumptions*: each soft clause $C^{\mathcal{T}}_j$ in $\varphi^{\mathcal{T}}_s$ is augmented with a fresh selection Boolean variable S_j (i.e., $\varphi^{\mathcal{T}}_s$ is rewritten into $\varphi'^{\mathcal{T}}_s \stackrel{\text{def}}{=} \{(\neg S_j \vee C^{\mathcal{T}}_j) \mid C^{\mathcal{T}}_j \in \varphi^{\mathcal{T}}_s\}$) and the proper set of selection variables $\lambda_s \stackrel{\text{def}}{=} \{S_j \mid C^{\mathcal{T}}_j \in \psi^{\mathcal{T}}_s\}$ is assumed at each call. This allows for "remembering" and reusing learned clauses from call to call. (Notice that, as long as the $\mathcal{T}$-lemmas are remembered from call to call, it is possible to drop the "$\cup\, \Theta^{\mathcal{T}}$" in the call $\mathsf{SMT.Solve}$ $(\varphi^{\mathcal{T}}_h \cup \psi^{\mathcal{T}}_s \cup \Theta^{\mathcal{T}})$.)

In a "white-box" integration scenario, in which it is possible to modify either or both the solvers involved, the following considerations may be of interest.

Generation of extra $\mathcal{T}$-lemmas. As illustrated in the second execution of Example 1, generating and storing extra $\mathcal{T}$-lemmas inside the SMT-solving phase —not only these explicitly involved in the conflict analysis— enlarges the $\mathcal{T}$-lemma pool and may possibly reduce the number of cycles. This can be obtained by means of SMT techniques like static learning and by storing *all* the $\mathcal{T}$-deduction clauses inferred by $\mathcal{T}$-propagation [5] (see [23, 8]). Notice that, to avoid introducing overhead for the underlying SAT solver, it suffices to *store* such $\mathcal{T}$-lemmas, without *learning* them.

4 Related work

Maximum satisfiability in SMT was first studied in [22], in the context of a general framework for optimization in SMT using "progressively stronger theories". An implementation for MaxSMT of this framework is described, but it is not publicly available.

An explicit reference to MaxSMT is found in [4], which describes the evaluation of an implementation of the WPM procedure [6] based on the YICES [2] SMT solver. This implementation is not publicly available. Another reference is in [5], where weighted Constraint Satisfaction Problems are translated into weighted MaxSMT instances.

The YICES solver provides also native support for MaxSMT. The approach used is based on incrementally invoking the solver in a mixed linear/binary-search fashion onto an SMT encoding of the MaxSMT problem, similar to that described in §2.1. The algorithm is not described in any publication, but we could obtain such information from personal communications with the authors.

The source distribution of the Z3 [15] solver provides an example implementation of an SMT version of the core-guided MaxSAT algorithm of [16], using the Z3 API. The algorithm is based on enumerating and counting unsatisfiable subformulas.

[5] In many SMT solvers implementing $\mathcal{T}$-propagation, $\mathcal{T}$-deduction clauses are generated on demand, only if they are needed by the underlying CDCL SAT solver for conflict analysis.

Also related are the works on optimization in SMT [10, 24], that can be used to encode the various MaxSMT problems. The work in [10] introduces the notion of "Theory of Costs" $\mathcal{C}$ to handle Pseudo-Boolean (PB) cost functions and constraints by an ad-hoc and independent "$\mathcal{C}$-solver" in the standard lazy SMT schema. MaxSMT can be handled by encoding it straightforwardly into a PB optimization problem (see §2.1). The implementation is available. The work in [24] introduced a wider notion of optimization in SMT, OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), with cost functions on variables on the *reals*, which allows for encoding also MaxSMT and SMT with PB cost functions (see §2.1). Some OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures combining lazy SMT and standard LP minimization techniques are presented. The implementation, done on top of the MATHSAT5 [12] SMT solver, is available.

Davies and Bacchus [14] proposed a MaxSAT algorithm (hereafter "DB") which, similarly to LL, works by iteratively ruling out subsets of the soft clauses of the input problem. In particular, DB builds iteratively a set $\mathcal{K}$ of unsat cores for $\varphi_h^{\mathcal{B}} \cup \varphi_s^{\mathcal{B}}$, i.e., at each loop iteration: (i) computes a new subset of soft clauses hs to drop as *minimum-cost hitting set* of $\mathcal{K}$; (ii) computes a new unsat core κ of $\varphi_h^{\mathcal{B}} \cup \varphi_s^{\mathcal{B}} \setminus hs$; this is repeated as long as $\varphi_h^{\mathcal{B}} \cup \varphi_s^{\mathcal{B}} \setminus hs$ is unsatisfiable.

Although [14] does not mention SMT, in principle this algorithm could be leveraged to SMT level (hereafter "DB-SMT"), by substituting SAT-level solving and unsat-core extraction with SMT-level ones. (Notice, however, that unlike with the SAT domain, efficiently finding minimal or nearly-minimal unsat cores in SMT is still an open research problem, see [13].) If so, LL and DB-SMT would be based on similar principles: [6]

- both algorithms would be based on constraint generation, producing constraints at every loop iteration which rule out subsets of the soft clauses;
- both would decouple solving and minimizing into two different subroutines.

The technical differences, however, would be manifold:

- Unlike with DB-SMT, LL is not a generalization of DB to SMT: unlike with DB, if it is fed a pair of purely-Boolean formulas, then it terminates in one iteration.
- DB-SMT would be driven by the combinatorics of the unsat cores to rule out, whilst LL is driven by the theory-information to be provided.
- The $\mathcal{T}$-lemma sets $\Theta_i^{\mathcal{T}}$ in LL are not the SMT counterpart of the unsat cores κ_i in DB-SMT: the former contain only *novel* clauses, the latter do not; there is not one-to-one correspondence between the generated sets of $\mathcal{T}$-lemmas and unsat cores. [7]
- MaxSAT is not the SMT counterpart of minimum-cost hitting set extraction: the latter starts from more fine-grained information, in the form of sets of unsat cores.
- It is easy to see that it would take at least N cycles to DB-SMT to rule out N clauses from $\psi_s^{\mathcal{B}}$. With LL the number of soft clauses discharged at each loop depends only on the quantity and quality of the $\mathcal{T}$-lemmas generated: in many cases (see Fig. 1) one iteration is enough to generate all the necessary $\mathcal{T}$-lemmas. [7]

[6] We are grateful to an anonymous reviewer who pointed out an analogy between DB and LL.

[7] For example, consider the second execution in Example 1: with DB-SMT there would be no unsat core κ_1 "equivalent" to $\Theta_1^{\mathcal{T}} \overset{\text{def}}{=} \{\theta_1, \theta_2, \theta_5\}$, allowing to directly pass from step 0 to step 1, since one needs 2 cores (and hence 2 loops) to generate a minimum-cost hs of size 2.

– the LL schema requires no SMT unsat-core extraction, nor minimum-cost hitting-set computation. (The MaxSAT subroutine is not committed to any MaxSAT schema.)

Finally, and importantly, DB/DB-SMT and LL radically differ in the *context* they were conceived (MaxSAT vs. MaxSMT), in their *usability* (the two schemas would pose very different constraints to a MaxSMT implementer) and *goals* (DB was conceived to address some efficiency issues in MaxSAT solvers [14], whilst LL is proposed as a modular approach to build MaxSMT solvers).

5 Experimental Evaluation

We have implemented our LL MaxSMT approach on top of our MATHSAT5 SMT solver [12] and of a selection of external MaxSAT tools. We have evaluated and compared the performances of the various LL instances and of every MaxSMT or MaxSMT-like solver available we are aware of, by means of an extensive empirical test on MaxSMT-modified SMT-LIB benchmarks. [8] In this section we present such evaluation.

5.1 Test Description

Benchmarks. As benchmark problems, we took the unsatisfiable SMT instances in the $\mathcal{LA}(\mathbb{Q})$ and $\mathcal{LA}(\mathbb{Z})$ categories of SMT-COMP [1], and we converted them into two groups of MaxSMT problems —partial MaxSMT and weighted partial MaxSMT respectively— by a random partition into hard and (weighted) soft clauses. In order to handle both CNF and non-CNF formulas, for each instance, we created the set of soft constraints by randomly selecting 20% assigning them a weight of 1 for the partial MaxSMT experiments, and a weight uniformly selected in the range $1 \ldots 100$ for the weighted partial MaxSMT ones, and then applied the process described in §2.2.

Competing MaxSMT solvers. Before starting our evaluation, we have asked to the main scientists of the MaxSAT community about the existence of *incremental* MaxSAT procedures, obtaining a negative answer. Thus, we decided to produce ourselves an implementation of the WPM MaxSAT algorithm [6] on top of MINISAT, and we also tried to enhance it with some degree of incrementality. Recently, Carlos Ansótegui has kindly sent us the code of a modified version of the WPM, which he had also adapted to get some incrementality, which invokes the YICES-1.0.36 as an external solver. Moreover, to guarantee a more interesting comparison, we have implemented on top of MATHSAT5 the same MaxSMT extension of the core-guided algorithm of [16] implemented in Z3. Finally, since the original implementation of SMT with PB cost functions and constraints of [10] was implemented on top of MATHSAT4, and in order to have a more significant comparison, we have recently ported it into MATHSAT5.

Thus, in this evaluation the first competitors were four instances of our Lemma-Lifting (LL) implementation, each using a different external MaxSAT solver:

[8] We also asked the authors of the related MaxSMT papers of §4 for other benchmarks, but none provided any meaningful benchmark.

LL$_{\text{WPM}}$, which uses the publicly-available WPM [6] implementation used in the 2012 MaxSAT-evaluation (which uses PICOSAT);

LL$_{\text{YICES-WPM}}$, which uses the above-mentioned non-public implementation of WPM provided to us by Carlos Ansótegui;

LL$_{\text{OWPM}}$, which uses our own incremental WPM implementation;

LL$_{\text{NI-OWPM}}$, as before, non-incremental version.

Other competitors were the following MaxSMT solvers:

YICES, the MaxSMT extension of YICES (see §4);

Z3, the MaxSMT extension of Z3 (see §4);

MATHSAT5-MAX, our own implementation on top of MATHSAT5 of the core-guided algorithm of [16], as with Z3.

Notice that the last two solvers handle only unweighted partial MaxSMT problems. The final competitors were the following solvers based on SMT with cost optimization (see §4), using the encodings described in §2.3:

MATHSAT4+$\mathcal{C}$(L), the tool from [10], using linear-search mode;

MATHSAT5+$\mathcal{C}$(L), the porting of the above procedure into MATHSAT5, using linear-search mode;

MATHSAT5+$\mathcal{C}$(B), as before, using binary-search mode;

OPTIMATHSAT, the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) tool of [24] described in §4, using its adaptive binary/linear search heuristics.

Notice that, if $\mathcal{T}$ is the $\mathcal{LA}(\mathbb{Z})$ theory, it is not possible to encode MaxSMT into OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) because the current implementation of OPTIMATHSAT cannot handle $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{LA}(\mathbb{Z})$.[9]

The Experiments. The experiments we performed can be divided into two groups. In the first group, we tested and compared the performances of all the eleven MaxSMT solvers on the benchmarks described above. This was performed on `Intel(R) Xeon(R) CPU E5650 2.67GHz` platform, with a 4GB memory limit and a 20 minute time limit for each run. In the second group, we made some more accurate analysis of the behaviour of the LL implementations. This was performed on a `Intel(R) Xeon(R) CPU E5520 2.27GHz` platform, using the same memory and time limits.

Check of the Results. We have checked the correctness of the results of all our own tools (i.e., those based on MATHSAT4 or MATHSAT5) by checking the models returned and by independently proving the unsatisfiability of the formula $\varphi^{\mathcal{T}'} \cup \{\mathsf{cost} < k\}$, where $\varphi^{\mathcal{T}'}$ and cost are defined as in (2) and k is the value of the cost returned by the tool. All results agreed with one another and were found correct by the above test.

Table 1. Results of the eleven MaxSMT solvers on partial MaxSMT instances

Solver	$\mathcal{LA}(\mathbb{Z})$		$\mathcal{LA}(\mathbb{Q})$		Total	
	#Solved	time (sec)	#Solved	time (sec)	#Solved	time (sec)
MATHSAT5-MAX	**95 / 106**	**6575.60**	88 / 93	2274.69	**183 / 199**	**8850.29**
LL_{OWPM}	92 / 106	5942.20	88 / 93	1785.48	180 / 199	7727.68
YICES	92 / 106	14478.43	87 / 93	5537.47	179 / 199	20015.9
$LL_{NI\text{-}OWPM}$	89 / 106	4439.98	88 / 93	1780.97	177 / 199	6220.95
$LL_{YICES-WPM}$	89 / 106	4937.91	87 / 93	1855.45	176 / 199	6793.36
LL_{WPM}	88 / 106	7154.19	88 / 93	2071.27	176 / 199	9225.46
MATHSAT5+$\mathcal{C}$(L)	84 / 106	7112.43	87 / 93	2175.34	171 / 199	9287.77
MATHSAT4+$\mathcal{C}$(L)	83 / 106	5220.14	85 / 93	1944.48	168 / 199	7164.62
Z3	89 / 106	4066.92	76 / 93	2427.59	165 / 199	6494.51
MATHSAT5+$\mathcal{C}$(B)	78 / 106	5030.85	87 / 93	2545.69	165 / 199	7576.54
OPTIMATHSAT	—	—	**89 / 93**	**1360.05**	—	—

Table 2. Results of the eleven MaxSMT solvers on partial weighted MaxSMT instances

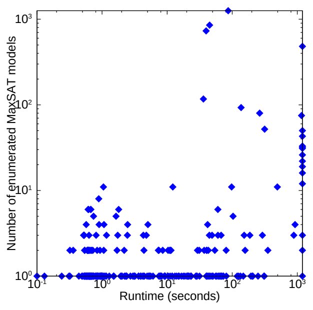
Solver	$\mathcal{LA}(\mathbb{Z})$		$\mathcal{LA}(\mathbb{Q})$		Total	
	#Solved	time (sec)	#Solved	time (sec)	#Solved	time (sec)
LL_{WPM}	**90 / 106**	**5194.73**	87 / 93	3033.66	**177 / 199**	**8228.39**
$LL_{NI\text{-}OWPM}$	86 / 106	1672.41	88 / 93	2062.35	174 / 199	3734.76
MATHSAT5+$\mathcal{C}$(L)	89 / 106	5501.38	84 / 93	2359.61	173 / 199	7860.99
LL_{OWPM}	85 / 106	1304.13	87 / 93	1836.53	172 / 199	3140.66
MATHSAT4+$\mathcal{C}$(L)	87 / 106	3105.01	85 / 93	2541.83	172 / 199	5646.84
$LL_{YICES-WPM}$	82 / 106	1423.53	87 / 93	2350.02	169 / 199	3773.55
YICES	83 / 106	12305.88	80 / 93	9804.16	163 / 199	22110.04
MATHSAT5+$\mathcal{C}$(B)	79 / 106	9482.61	83 / 93	2627.35	162 / 199	12109.96
OPTIMATHSAT	—	—	**88 / 93**	**1947.06**	88 / 93	1947.06
Z3	—	—	—	—	—	—
MATHSAT5-MAX	—	—	—	—	—	—

5.2 Results

The results of the evaluation of the eleven MaxSMT solvers are presented in Tables 1 and 2, reporting for each solver the number of instances solved within the timeout and the total runtime taken to solve them. (Rows are sorted according to total ($\mathcal{LA}(\mathbb{Q})$ + $\mathcal{LA}(\mathbb{Z})$) performance, best performances for each category are in **bold**.) Note that, for the reasons highlighted above, Z3 and MATHSAT5-MAX were not run on weighted instances, and OPTIMATHSAT was not ran on the $\mathcal{LA}(\mathbb{Z})$ instances (this is marked with a "—").

Looking at the data in Tables 1 and 2 some considerations are in order.

[9] This is due to the fact that the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) framework requires that $\mathcal{LA}(\mathbb{Q})$ and $\mathcal{T}$ are signature-disjoint theories [24], which is not the case if $\mathcal{T}$ is $\mathcal{LA}(\mathbb{Z})$.



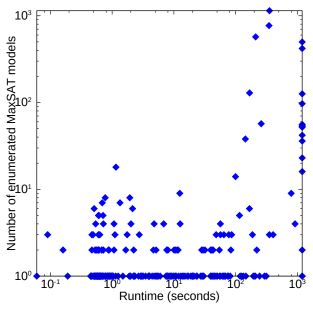


Fig. 1. Number of while-cycles wrt. runtime for LL$_{\text{OWPM}}$ (left) and LL$_{\text{NI-OWPM}}$ (right).

(i) MATHSAT5-MAX is the overall best performer on the unweighted group, whilst LL$_{\text{WPM}}$ is the overall best performer on the weighted one. If we restrict to $\mathcal{LA}(\mathbb{Q})$ theory, OPTIMATHSAT is the winner in both unweighted and weighted groups. However, there is no hands-down winner, and the performance gaps among the eleven solvers are not dramatic.

(ii) Overall, the LL tools behave quite well, all being in the highest part of the ranking. Among them, there is not an absolute winner: LL$_{\text{OWPM}}$ is the best performer on the unweighted group, whilst LL$_{\text{WPM}}$ is the best performer on the weighted one.

(iii) There is no definite winner between LL$_{\text{OWPM}}$ and LL$_{\text{NI-OWPM}}$: the former is better on unweighted test, the latter on weighted ones, and the performance gaps are very limited. Thus, incrementality does not seem to pay as much as one could expect. Similarly, there is no definite winner between LL$_{\text{YICES-WPM}}$ and LL$_{\text{WPM}}$. (Notice, however, that these two call two different backend solvers, YICES and PICOSAT.)

Overall, the results are too limited and heterogeneous to infer the superiority of one approach wrt. another. However, we can safely conclude that, despite its simplicity, the LL approach is competitive wrt state-of-the-art ones.

In Figure 1 and Table 3 we analyze the behaviour of the LL solvers on all tests (weighted/unweighted, $\mathcal{LA}(\mathbb{Q})$/$\mathcal{LA}(\mathbb{Z})$).

Figure 1 shows the number of while-cycles performed by the LL algorithm with LL$_{\text{OWPM}}$ (left) and LL$_{\text{NI-OWPM}}$ (right), plotted against runtime. We notice that a very high percentage of instances is solved at the first loop, and the vast majority of instances is solved in less than 10 loops. This induces us to conjecture that SMT.Solve in many cases is able to produce very soon all the $\mathcal{T}$-lemmas which are necessary to MaxSAT to rule out the wrong truth assignments.

Table 3 analyzes the percentage of CPU time spent inside MaxSAT calls for the four LL tools. [10] We notice that for most solvers and most instances, the solver spends less than 20% and this fact is particularly evident in the easiest problems. Thus, the overall CPU time is mostly dominated by the time spent inside SMT.Solve. In particular, in the

[10] For instance (1st block, 3rd column): out of 116 instance problem for which LL$_{\text{OWPM}}$ took less than one second to execute, with 81 instances MaxSAT calls required less than 20% 40%

Table 3. An overview of runtime spent on MaxSAT calls compared to total runtime for LL solvers

	Runtime LL_{OWPM} (in seconds)					
		$[0,1[$	$[1,10[$	$[10,100[$	$[100,1000[$	≥ 1000
% Time MaxSAT	$0 \leq p < 20$	81	41	15	5	13
	$20 \leq p < 40$	19	12	9	17	27
	$40 \leq p < 60$	10	6	5	19	12
	$60 \leq p < 80$	1	0	6	15	3
	$80 \leq p \leq 100$	5	15	0	9	0
	Total	116	74	35	65	55
	Runtime LL_{OWPM} (in seconds)					
		$[0,1[$	$[1,10[$	$[10,100[$	$[100,1000[$	≥ 1000
% Time MaxSAT	$0 \leq p < 20$	115	53	43	49	59
	$20 \leq p < 40$	5	6	1	1	9
	$40 \leq p < 60$	1	1	2	0	1
	$60 \leq p < 80$	0	1	0	0	0
	$80 \leq p \leq 100$	0	1	4	0	0
	Total	121	62	50	50	69
	Runtime $\text{LL}_{\text{NI-OWPM}}$ (in seconds)					
		$[0,1[$	$[1,10[$	$[10,100[$	$[100,1000[$	≥ 1000
% Time MaxSAT	$0 \leq p < 20$	95	49	32	48	53
	$20 \leq p < 40$	11	9	8	2	4
	$40 \leq p < 60$	4	0	5	0	0
	$60 \leq p < 80$	8	1	0	0	1
	$80 \leq p \leq 100$	4	2	11	0	4
	Total	122	61	56	50	62
	Runtime LL_{WPM} (in seconds)					
		$[0,1[$	$[1,10[$	$[10,100[$	$[100,1000[$	≥ 1000
% Time MaxSAT	$0 \leq p < 20$	115	58	16	43	57
	$20 \leq p < 40$	1	1	11	8	1
	$40 \leq p < 60$	4	1	9	2	0
	$60 \leq p < 80$	8	0	2	1	1
	$80 \leq p \leq 100$	1	0	6	3	4
	Total	129	60	44	57	63

samples in which the solution is found in one loop, most time is taken by SMT.Solve to enumerate the necessary $\mathcal{T}$-lemmas in one shot.

This also explains in part the low effect of incrementality in our experiments, since the cost of MaxSAT calls does not seem to represent the actual bottleneck of the process. Notice that, if we compare the data on LL_{OWPM} and $\text{LL}_{\text{NI-OWPM}}$ in Table 3, we notice that indeed in the incremental version the percentage of time spent inside MaxSAT is smaller than in the non-incremental one. However, since the total cost is mostly dominated by SMT.Solve, the benefits of this fact are not significant. (Also, we must recall that, since we are not expert MaxSAT developers, our implementation of incrementality is quite naive.)

6 Conclusions and Future Work

In this paper we have presented a novel "modular" Lemma-Lifting approach for MaxSMT, which combines a lazy SMT solver with a purely-propositional MaxSAT solver. Despite its simplicitly, LL proves competitive with previous approaches.

Depending on one's expertise on and access to SMT and MaxSAT technology, we see different ways the LL approach can be implemented into a MaxSMT tool.

- Whoever cannot or does not want to put the hands on either solver's code, can take both an SMT and a MaxSAT solver off-the-shelf and implement our algorithm on top of their API (or even interface with them via file exchange). In this case, implementation is straightforward.
- MaxSAT-solver developers can leverage to SMT level the expressiveness of their own tool by interfacing with one SMT solver, without implementing any SMT functionality in-house. They can also customize their own MaxSAT tool to improve the synergy of the two tools (in particular, by making it as incremental as possible).
- SMT-solver developers can extend their own tool with MaxSMT functionality by interfacing with one or more MaxSAT solvers off-the-shelf, with no need of implementing MaxSAT functionalities in-house. They can also customize their own solver (e.g., by maximizing the generation of theory lemmas).
- A person with access to, and enough expertise on, both SMT- and MaxSAT- solver development can adopt our approach to produce a highly efficient MaxSMT tool, with the possibility of customizing both tools. Notice that, in this case, our approach can also be combined with other SMT optimization techniques (e.g., those described in [22, 10, 24, 4]).

We believe that this paper opens novel research avenues in MaxSMT. In particular, we see many directions along which the LL approach can be improved and extended.

Customizing SMT and MaxSAT solvers. The LL approach would strongly benefit from more effective $\mathcal{T}$-lemma generators and incremental MaxSAT solvers.

Interleaving of SMT- and MaxSAT-solving steps. Algorithm 1 interleaves *complete* calls to $\mathsf{SMT.Solve}$ and MaxSAT. This can be generalized to more fine-grained interleaving schemas, in which *steps* of such executions can be interleaved. For instance, it is possible to interrupt $\mathsf{SMT.Solve}$ as soon as some amount of $\mathcal{T}$-lemmas has been generated, and invoke MaxSAT afterwords. Vice versa, it is possible to interrupt MaxSAT as soon as a non-optimal $\psi_s^{\mathcal{B}}$ is generated, and feed it back to $\mathsf{SMT.Solve}$. (As an extreme case, one could feed to $\mathsf{SMT.Solve}$ only the assignment $\mu^{\mathcal{T}}$ produced by MaxSAT: if so, $\mathsf{SMT.Solve}$ would be simply used as a $\mathcal{T}$-Solver.)

Combination with other approaches. The lemma-Lifting approach can be combined with other approaches in various ways. For instance, one could enhance the use of SMT by exploiting SMT with cost constraints [10] and extraction of $\mathcal{T}$-unsatisfiable cores [13] to further prune the search.

Overall, novel strategies and heuristics can be investigated to extend and improve Algorithm 1 along the above directions.

References

1. SMT-COMP. `http://www.smtcomp.org/2010/`.
2. Yices. `http://yices.csl.sri.com/`.
3. Max-SAT 2013, Eighth Max-SAT Evaluation, 2013. `http://maxsat.ia.udl.cat`.
4. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *SARA*, 2011.
5. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Solving weighted CSPs with meta-constraints by reformulation into Satisfiability Modulo Theories. *Constraints*, 18(2), 2013.
6. C. Ansótegui, M. L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196, 2013.
7. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on Demand in SAT Modulo Theories. In *LPAR*, volume 4246 of *LNAI*. Springer, 2006.
8. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, ch. 26. In Biere et al., editors. *Handbook of Satisfiability*. IOS Press, 2009.
9. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10), 2006.
10. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*. Springer, 2010.
11. A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. A Modular Approach to MaxSAT Modulo Theories., 2013. Extended version. `http://disi.unitn.it/~rseba/sat13/extended.pdf`.
12. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT 5 SMT Solver. In *TACAS*, volume 7795 of *LNCS*. Springer, 2013.
13. A. Cimatti, A. Griggio, and R. Sebastiani. Computing Small Unsatisfiable Cores in SAT Modulo Theories. *JAIR*, 40, 2011.
14. J. Davies and F. Bacchus. Solving MaxSAT by solving a sequence of simpler SAT instances. In *CP*, volume 6876 of *LNCS*. Springer, 2011.
15. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*. Springer, 2008.
16. Z. Fu and S. Malik. On solving the partial Max-SAT problem. In *SAT*, volume 4121 of *LNCS*. Springer, 2006.
17. F. Heras, J. Larrosa, and A. Oliveras. Minimaxsat: An Efficient Weighted Max-SAT solver. *JAIR*, 31, 2008.
18. F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *AAAI*, 2011.
19. C. M. Li and F. Manyà. *MaxSAT, Hard and Soft Constraints*, ch. 19. In Biere et al., editors. *Handbook of Satisfiability*. IOS Press, 2009.
20. C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *JAIR*, 30, 2007.
21. A. Morgado, F. Heras, and J. Marques-Silva. Improvements to core-guided binary search for MaxSAT. In *SAT*, volume 7317 of *LNCS*. Springer, 2012.
22. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *SAT*, volume 4121 of *LNCS*. Springer, 2006.
23. R. Sebastiani. Lazy Satisfiability Modulo Theories. *JSAT*, 3(3-4), 2007.
24. R. Sebastiani and S. Tomasi. Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*. Springer, 2012.