Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories

Katalin Fazekas[1(✉)], Fahiem Bacchus[2], Armin Biere[1]

[1] Johannes Kepler University, Linz, Austria,
`katalin.fazekas@jku.at`
[2] University of Toronto, Toronto, Canada

Abstract. Solving optimization problems with SAT has a long tradition in the form of MaxSAT, which maximizes the weight of satisfied clauses in a propositional formula. The extension to maximum satisfiability modulo theories (MaxSMT) is less mature but allows problems to be formulated in a higher-level language closer to actual applications. In this paper we describe a new approach for solving MaxSMT based on lifting one of the currently most successful approaches for MaxSAT, the implicit hitting set approach, from the propositional level to SMT. We also provide a unifying view of how optimization, propositional reasoning, and theory reasoning can be combined in a MaxSMT solver. This leads to a generic framework that can be instantiated in different ways, subsuming existing work and supporting new approaches. Experiments with two instantiations clearly show the benefit of our generic framework.

1 Introduction

SMT solvers have become indispensable tools for solving a wide range of problems in many areas. Such solvers provide either a satisfying assignment (e.g., a witness for a bug) or a proof of unsatisfiability (e.g., proving that a particular abstraction does not display a bug). However, in many applications the problem to be solved is more naturally cast as an optimization problem: find an assignment that minimizes some cost function. Li et al. [1], for instance, give a range of applications where optimization is critical. The need to solve such applications has led to a range of work addressing optimization in SMT (e.g., [1–8]).

Work on SMT optimization varies in the generality of the objective functions that can be modeled. For example, [1, 4] address optimizing objective functions stated in the theory of linear real arithmetic, while [7] can deal with linear objective functions in which some variables are restricted to be integer. MaxSMT [2] is a restricted but important sub-problem in which the objective functions are linear expressions over Boolean variables (Pseudo Boolean expressions).

In this paper we focus on MaxSMT. Although MaxSMT is not as general as some other optimization approaches, MaxSMT specific solvers are often more efficient on problems where Pseudo Boolean objectives suffice [8], and recent rapid progress in the efficiency of MaxSAT solvers [9] indicates that this special case may more likely scale to practical problems than more general optimization approaches. Furthermore, MaxSAT already has a wide and growing range of applications including planning, fault localization in C code, design debugging, and a variety of problems in data analysis (see [10]). This indicates that Pseudo Boolean objectives are sufficient in a range of applications, and hence MaxSMT, with its addition of theories, is likely to have even greater applicability.

The implicit hitting set (IHS) approach [11] for solving MaxSAT has seen considerable recent progress and is now one of the most effective ways of solving MaxSAT. For example, IHS solvers have been the top performing solvers on weighted problems in the most recent 2016 and 2017 evaluations of MaxSAT solvers [9]. One of the key benefits of the IHS approach is that it provides a clear separation between optimization and propositional reasoning. In particular, in IHS solvers optimization is performed by a separate minimum cost hitting set solver, while the SAT solver is used solely for propositional reasoning. This separation of concerns supports the observed improved performance by allowing the exploitation of more efficient specialized solvers for each component.

Since MaxSAT and MaxSMT are quite similar problems, this naturally leads to the question of how MaxSMT can be similarly separated into optimization, propositional reasoning, and theory reasoning. In this paper we provide a general view of how these separate components can be combined to solve MaxSMT by providing a formal reasoning calculus [12] for MaxSMT solvers that achieves a clear separation of these different components. The calculus formalizes a notion of state that abstracts the more complex notions of state used in implemented solvers, and a set of inference rules for transforming the state that abstracts the operations performed by implemented solvers. The power of the calculus is that almost any scheme for scheduling the application of these rules leads to a solution. Hence, it supports the design of a wide range of different implementations of the basic inferences and of control structures for scheduling their application. It also provides a formal framework for effective harvesting of advances in MaxSAT for improving MaxSMT and vice versa.

2 Preliminaries

We consider formulas F in conjunctive normal form (CNF) consisting of a set of clauses, where each clause C is a disjunction of literals, which are first-order atoms or propositional variables, or their negation. MaxSAT problems are specified by a purely propositional CNF F, without first-order atoms, partitioned into **hard** and **soft** clauses, $\mathit{hard}(F)$ and $\mathit{soft}(F)$. A **feasible solution** to the MaxSAT problem is a truth assignment that satisfies all of the hard clauses. A **core** in MaxSAT solving is a *set of soft clauses* (a subset of $\mathit{soft}(F)$) that when combined with the hard clauses forms an unsatisfiable set of clauses.

Each soft clause C has a positive weight, denoted by $\mathit{cost}(C)$, which specifies the cost of falsifying it. The cost of a set of soft clauses S is the sum of the costs of the soft clauses it contains: $\mathit{cost}(S) = \sum_{C \in S} \mathit{cost}(C)$. The cost of a feasible solution π is the sum of the costs of the soft clauses it falsifies: $\mathit{cost}(\pi) = \mathit{cost}(\{C \in S \mid \pi \not\models C\})$. An **optimal solution** for a MaxSAT problem is a feasible solution with *minimum cost* among all feasible solutions. Solving a MaxSAT problem is the task of finding an optimal solution.

We can restrict our attention, w.l.o.g, to formulas F in which all soft clauses are unit. In particular, any non-unit soft clause C can be converted to a unit soft clause by (i) adding a new (*relaxed*) hard clause $C \vee v$ where v is a *new* propositional variable (called a *relaxing* or *selector* variable), and (ii) replacing the soft clause C with a new unit soft clause $(\neg v)$ with $\mathit{cost}((\neg v)) = \mathit{cost}(C)$. This transformation is sound since any optimal solution satisfies $C \leftrightarrow \neg v$.

Considering ground first-order atoms generalizes MaxSAT to MaxSMT [2], as SMT [13] generalizes SAT. As in MaxSAT, a MaxSMT problem consists of a set of hard and soft clauses with each soft clause having a weight. However, in MaxSMT literals can be formed from theory atoms as well as from propositional variables. For example, over the theory of linear real arithmetic (LRA) we could form clauses like $(p \vee \neg(1 \leq y) \vee (x + y \geq 2))$, with a propositional variable p and LRA theory atoms $(1 \leq y)$ and $(x + y \geq 2)$.

Let $\mathit{atoms}(F)$ be the set of atoms in F, which range over propositional variables as well as theory atoms. We extend this notion to literals, clauses, and sequences of literals accordingly. A (partial truth) **assignment** over $\mathit{atoms}(F)$ is a sequence of literals from $\mathit{atoms}(F)$ that (i) does not contain both x and $\neg x$ for any $x \in \mathit{atoms}(F)$ and (ii) has no repeated literals. If A and M are two sequences of literals we write AM to indicate their concatenation. An assignment π over $\mathit{atoms}(F)$ is called a **propositional model** of F, denoted by $\pi \models F$, if it satisfies the Boolean abstraction of F in which theory atoms are treated simply as new independent propositional variables. A propositional model of F, π, is also a **theory consistent model** of F if the conjunction of theory literals made true by π is consistent with all theory axioms, denoted $\pi \models_T F$.

A feasible solution π for a MaxSMT formula F is required to be theory consistent model of $\mathit{hard}(F)$ ($\pi \models_T \mathit{hard}(F)$). The cost of π is defined as in MaxSAT. Accordingly, solving MaxSMT means finding an optimal (minimum cost) feasible solution. Again, w.l.o.g., we can assume that all soft clauses are unit clauses. Similarly, a **core** in MaxSMT is a subset of *soft clauses* that when combined with the hard clauses does not have a theory consistent model.

Let K be a set of cores, i.e., a set of sets of soft clauses. A **hitting set** η of K is a set of soft clauses that has a non-empty intersection with every set in K: $\forall \kappa \in K.\, \eta \cap \kappa \neq \emptyset$. As defined above $\mathit{cost}(\eta) = \sum_{C \in \eta} \mathit{cost}(C)$.

3 Abstract Hitting Set based MaxSMT Solving

The main contribution of our paper is to introduce and formalize a calculus for the implicit hitting set (IHS) approach for MaxSMT, which at the same time provides the first formal calculus for the IHS approach to MaxSAT [11]. Our calculus captures a flexible separation between optimization, propositional reasoning, and theory reasoning, supporting a number of different implementation strategies. The separation between optimization and propositional reasoning is achieved by exploiting the IHS approach for solving MaxSAT/MaxSMT. Other

Table 1: Transition rules for solving SAT under assumptions (**A-Sat**)

UnitProp $A \mid M \mid F \Longrightarrow A \mid M\,\ell \mid F$	**if**	$\begin{cases} \text{There is a clause } (C \vee \ell) \in F \text{ s.t.} \\ AM \models \neg C \text{ and } \mathit{atom}(\ell) \notin \mathit{atoms}(AM) \end{cases}$
Decide $A \mid M \mid F \Longrightarrow A \mid M\,\ell^d \mid F$	**if**	$\mathit{atom}(\ell) \in \big(\mathit{atoms}(F) \setminus \mathit{atoms}(AM)\big)$
Backjump $A \mid M\ell^d N \mid F \Longrightarrow A \mid M\,\ell' \mid F$	**if**	$\begin{cases} \text{There is a clause } C \in F \text{ s.t. } AM\ell^d N \models \neg C \\ \text{and a clause } C' \vee \ell' \text{ s.t. } F \models C' \vee \ell', \\ \quad AM \models \neg C' \text{ and } \mathit{atom}(\ell') \in \mathit{atoms}(\ell^d N) \end{cases}$
Learn $A \mid M \mid F \Longrightarrow A \mid M \mid F, C$	**if**	$\begin{cases} F \models C \text{ and } C \notin F \\ \mathit{atoms}(C) \subseteq \big(\mathit{atoms}(F) \cup \mathit{atoms}(AM)\big) \end{cases}$
Forget $A \mid M \mid F, C \Longrightarrow A \mid M \mid F$	**if**	$F \models C$
SatModel $A \mid M \mid F \Longrightarrow \mathit{SAT}(AM, F)$	**if**	$AM \models F$
UnSat $A \mid M \mid F \Longrightarrow \mathit{conflict}(F, C)$	**if**	$\begin{cases} \text{There is a clause } D \in F \text{ s.t. } AM \models \neg D \\ M \text{ contains no decision literals} \\ \text{and } C \text{ is a clause s.t. } F \models C \text{ and } A \models \neg C \end{cases}$

approaches to MaxSAT solving, e.g., [14–16], employ exclusively propositional reasoning, doing optimization by solving a sequence of SAT decision problems.

Our calculus can be modified to model such approaches by combining the optimization and propositional reasoning components into a single "MaxSAT" component. This would provide a formal model of MaxSMT approaches like [3]. However, as we will demonstrate below, even without such a formal model our calculus still provides a framework for understanding the approach of [3].

IHS and the above cited approaches to solving MaxSAT use propositional reasoning to find cores by exploiting SAT solving under assumptions [17]. In particular, for any subset of soft clauses S we can determine if S and the hard clauses are satisfiable by assuming that the literal of each (unit) soft clause in S is true. If the conjunction of these literals with the hard clauses is unsatisfiable, the SAT solver assumption mechanism returns a clause falsified by the assumptions. Hence, this clause contains only negations of assumed literals, identifying a subset of S that, with the hard clauses, is unsatisfiable (i.e., a core).

Hence, as a first step towards a formal calculus for IHS MaxSMT solving, we provide a calculus for assumption based SAT and SMT reasoning. To the best of our knowledge, such a calculus has not been specified before. This contribution should be useful independent of MaxSMT since assumption based reasoning is used in many different applications besides optimization.

3.1 SAT/SMT Solving under Assumptions

To formalize assumption based incremental SAT solving [17] and lift it to SMT, we extend the DPLL(T) calculus originally presented in [12]. As above let F be a first-order quantifier-free CNF formula over theory T. The states of our calculus are specified by a triple $A \mid M \mid F$, where F is a CNF formula (initially the input formula), A and M are non-overlapping assignments over $\mathit{atoms}(F)$. A is the given set of assumptions, and M is the solver's current set of implied and decided (noted by a superscript d, e.g., ℓ^d) literals.

Table 2: Additional rules for solving SMT under assumptions (**A-Smt**)

T-**Backjump** $A \mid M\ell^d N \mid F \Longrightarrow A \mid M\,\ell' \mid F$	**if** $\begin{cases} \text{There is a clause } C \in F \text{ s.t. } AM\ell^d N \models \neg C \\ \text{and a clause } C' \vee \ell' \text{ s.t. } F \models_T C' \vee \ell', \\ \quad AM \models \neg C' \text{ and } atom(\ell') \in atoms(\ell^d N) \end{cases}$
T-**Learn** $A \mid M \mid F \Longrightarrow A \mid M \mid F, C$	**if** $\begin{cases} F \models_T C \text{ and } C \notin F \\ atoms(C) \subseteq \big(atoms(F) \cup atoms(AM)\big) \end{cases}$
T-**Forget** $A \mid M \mid F, C \Longrightarrow A \mid M \mid F$	**if** $F \models_T C$
T-**Model** $A \mid M \mid F \Longrightarrow T\text{-}SAT(AM, F)$	**if** $AM \models_T F$

The transition rules given in Table 1 specify an abstract assumption based SAT solver (**A-Sat**). These rules follow [12] but are adapted to handle assumptions; the main changes are as follows. First, the abstract states and rules have been extended with a (possibly empty) set of assumption literals A over $atoms(F)$. For example, **Learn** is the same, but **UnitProp** requires $AM \models \neg C$, instead of $M \models \neg C$. Second, we modified the rule Fail to obtain a new rule **UnSat** that transitions into a *conflict(F,C)* state when M has no decision literals and $AM \models \neg D$ for some $D \in F$. In that case, $F \wedge A$ must be unsatisfiable, and we can always find a clause C implied by F and falsified by A (e.g., by resolving all literals negated by M from the clause D). And third, we introduce a transition rule that leads to an explicit $SAT(AM, F)$ state when $AM \models F$ holds. This facilitates combining the assumption based transitions with a MaxSAT or MaxSMT transition system. It can be noted that our calculus captures the technique of [17] which uses one particular control scheme to derive the clauses D and C used in the **UnSat** rule (it is irrelevant that [17] intermixes A and M).

Abstract assumption based SMT solving (**A-Smt**) is specified by the rules of Table 2 along with the rules **UnitProp**, **Decide** and **UnSat** of Table 1. Note that T-entailment subsumes propositional entailment, i.e., $F \models C$ implies $F \models_T C$. Hence, T-**Learn** can learn any clauses that **Learn** can, and T-**Learn** need not always employ theory reasoning (it can also use propositional reasoning to perform learning). This can be important in practice if reasoning in T is expensive. The same remark holds for T-**Backjump** and T-**Forget**.

It can also be noted that **UnSat** requires a falsified clause D to be in F. Hence, when $F \wedge A$ is propositionally satisfiable but T-unsatisfiable our calculus requires sufficient theory lemmas from T-**Learn** so as to obtain a falsified clause in F and to derive a clause C falsified by A.

We say that a state S in a transition system is **final** when no rules are applicable to it. Given a set of assumed literals A and a formula F, the initial state of assumption based SAT/SMT solving is $A \mid \emptyset \mid F$. Deciding the satisfiability/T-satisfiability of F assuming A is a derivation of the form $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S_n$, where S_n is a final state in the **A-Sat**/**A-Smt** system.

Theorem 1 (Termination). *Any sequence of transitions $A \mid \emptyset \mid F \Longrightarrow \cdots$ in **A-Sat** (**A-Smt**) that contains no infinite subsequence consisting only of rules from the set \{**Learn**, **Forget**\} (\{T-**Learn**, T-**Forget**\}), is finite.*

Table 3: Transition rules for Optimization ($*$ is any SAT/SMT state) (**A-MaxSMT**)

SAT/SMT-Transition $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $(LB, UB, \mu) \mid K \mid \langle *' \rangle$	**if**	$\begin{cases} *' \text{ is reachable from } * \text{ by} \\ \text{a single } \textbf{A-Sat}/\textbf{A-Smt} \text{ transition step} \\ \text{(see Table 1 and Table 2)} \end{cases}$
Core $(LB, UB, \mu) \mid K \mid \langle \mathit{conflict}(F, C) \rangle \Longrightarrow$ $(LB, UB, \mu) \mid K, \kappa \mid \langle \mathit{conflict}(F, C) \rangle$	**if**	$\begin{cases} \kappa = \{(\neg \ell) \mid \ell \in C\} \text{ and } \kappa \notin K \\ (\kappa \text{ is set of soft clauses}) \end{cases}$
HS $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $(LB, UB, \mu) \mid K \mid \langle A' \mid \emptyset \mid F \rangle$	**if**	$\begin{cases} \eta = \mathit{HS}(K) \\ A' = \{\ell \mid (\ell) \in (\mathit{soft}(F) - \eta)\} \end{cases}$
MinHS $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $(LB', UB, \mu) \mid K \mid \langle A' \mid \emptyset \mid F \rangle$	**if**	$\begin{cases} \eta = \mathit{minHS}(K) \\ A' = \{\ell \mid (\ell) \in (\mathit{soft}(F) - \eta)\} \\ LB' = \max(LB, \mathit{cost}(\eta)) \end{cases}$
ImprovedSolution $(LB, UB, \mu) \mid K \mid \langle \mathit{T\text{-}SAT}(AM, F) \rangle \Longrightarrow$ $(LB, \mathit{cost}(AM), AM) \mid K \mid \langle \mathit{T\text{-}SAT}(AM, F) \rangle$	**if**	$\mathit{cost}(AM) < UB$
OptimalSolution $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow \mathit{optSoln}(\mu)$	**if**	$LB \geq UB$

Theorem 2 (Soundness). *For any derivation $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S$ in **A-Sat (A-Smt)** where S is final with respect to **A-Sat (A-Smt)** we have*

1. $S = \mathit{conflict}(F', C)$ with $F' \models C$, $A \models \neg C$ iff $F \wedge A$ is $(T\text{-})$unsatisfiable.

2. $S = (T\text{-})SAT(AM, F')$ with $AM \models_{(T)} F'$ iff $F \wedge A$ is $(T\text{-})$satisfiable.

We can treat A as a prefix of decision literals of M that can not be changed by backjumping. Under this interpretation the results of [12] can be extended to obtain proofs for Theorems 1 and 2. We omit the details due to space constraints.

3.2 IHS MaxSAT/MaxSMT Solving

To obtain an abstract IHS based MaxSMT solver we add the rules given in Table 3. These rules extend the states of **A-Smt** by adding K and the triple (LB, UB, μ), where K is a set of cores, LB and UB are lower and upper bounds on the cost of an optimal solution to the input CNF F, and μ is a feasible solution, represented as a sequence of literals over $\mathit{atoms}(F)$, with $\mathit{cost}(\mu) = UB$. Let **A-MaxSMT** be the transition system defined by the rules in Table 3 along with the rules **A-Smt**.[3] The initial state of **A-MaxSMT** is always the state $IS = (0, \infty, \mathit{undef}) \mid \emptyset \mid \langle \{\ell \mid (\ell) \in \mathit{soft}(F)\} \mid \emptyset \mid \mathit{hard}(F) \rangle$, i.e., we start with valid lower and upper bounds, an empty set of cores, an initial assumption that all soft clauses are satisfied, and all of the hard clauses of F.

The calculus computes a growing set of cores K, each obtained from assumption based SMT solving, and uses the two subroutines, $\mathit{minHS}(K)$ which returns a minimum cost hitting set of K, and $\mathit{HS}(K)$ which returns an arbitrary hitting set of K. It can be noted that the assumptions (initially and after the rules **HS** or **MinHS**) are always asserting that some subset of the soft clauses along with the hard clauses are satisfied. Hence, as explained above, the subset of $\mathit{soft}(F)$

[3] IHS MaxSAT solvers can be obtained by using the **A-Sat** rules and replacing $\mathit{T\text{-}SAT}(AM, F)$ in **ImprovedSolution** with $\mathit{SAT}(AM, F)$.

identified by the returned conflict and added to K by rule **Core** must be a core. Furthermore, the assumptions always specify that all soft clauses *except* those in some hitting set η of K are true. Thus, the returned conflict must identify a *new* core κ that cannot already be in K. In particular, κ is a subset of $\mathit{soft}(F) - \eta$ (it is a subset of the assumed true soft clauses) but no $s \in K$ is a subset of $\mathit{soft}(F) - \eta$ since s contains a non-empty subset $s \cap \eta$ not in $\mathit{soft}(F) - \eta$.

We say that $S_1 \Longrightarrow \cdots \Longrightarrow S_n$ is a **progressing subsequence** if (a) S_1 is the result of applying the **MinHS** rule, (b) all transitions in the sequence arise from applying one of the **A-Smt** rules (i.e. are **SAT/SMT-Transition** steps), and (c) S_n is final with respect to the rules of **A-Smt** (i.e., no **SAT/SMT-Transition** is applicable).

Theorem 3 (Termination). *If $\mathit{hard}(F)$ is T-satisfiable then any derivation $\mathit{IS} \Rightarrow S_1 \Rightarrow \cdots$ of **A-MaxSMT** is finite if it satisfies the following conditions:*
1. *contains no infinite subsequence of rules from the set $\{T$-**Learn**,T-**Forget**$\}$*
2. *contains no infinite subsequence not containing a progressing subsequence*
3. *always applies the transitions **OptimalSolution**, **ImprovedSolution** and **Core** whenever they are applicable (with **OptimalSolution** being applied first).*

Theorem 4 (Soundness). *If $\mathit{hard}(F)$ is T-satisfiable, $\mathit{IS} \Rightarrow \cdots \Rightarrow S_n$ is a finite sequence of transitions in **A-MaxSMT**, and S_n is final in **A-MaxSMT**, then S_n is $\mathit{optSoln}(\mu)$ and μ is an optimal solution of F.*

Theorem 4 is immediate from the fact that (a) $\mathit{optSoln}(\mu)$ is the only final state in **A-MaxSMT**, (b) LB and UB are always valid bounds, and (c) $\mathit{cost}(\mu) = \mathit{UB}$.

Hence, the main result is that the calculus terminates under the conditions of Theorem 3. A sketch of the proof follows. First, from Theorem 1 it can be seen that all progressing subsequences must be finite, and thus any infinite sequence of transitions must contain an infinite number of progressing subsequences. Theorem 2 shows that every progressing subsequence must reach either a T-SAT or a *conflict* final state. If a *conflict* state is reached, then **Core** must be applied next. As explained above this must add a *new* core to K. Each core is a subset of $\mathit{soft}(F)$ so only a finite number of cores exist. Hence, only a finite number of progressing subsequences can end in *conflict*. Otherwise, the progressing subsequence reaches T-SAT. But this can happen only once since the feasible solution found, AM, must be an optimal solution. AM satisfies all clauses except those in a minimum cost hitting set η of K (obtained from **MinHS**), and hence $\mathit{cost}(\mathit{AM}) \leq \mathit{cost}(\eta)$. Every feasible solution π satisfies $\mathit{hard}(F)$ and every core is unsatisfiable when added to $\mathit{hard}(F)$. Hence, π must falsify at least one soft clause in every core; i.e., the set of clauses falsified by π is a hitting set of K. So by the definition of *cost* and the minimality of η, $\mathit{cost}(\eta) \leq \mathit{cost}(\pi)$, and thus $\mathit{cost}(\mathit{AM}) \leq \mathit{cost}(\pi)$ for every feasible solution π. Once AM is found, **ImprovedSolution** must be applicable and the condition $\mathit{cost}(\mathit{AM}) = \mathit{UB} \leq \mathit{cost}(\eta) \leq \mathit{LB}$ is achieved (**MinHS** ensures $\mathit{cost}(\eta) \leq \mathit{LB}$). Then **OptimalSolution** must be applied and the derivation terminates. In sum, under the stated conditions only a finite number of progressing subsequences can be executed and so the derivation must be finite.

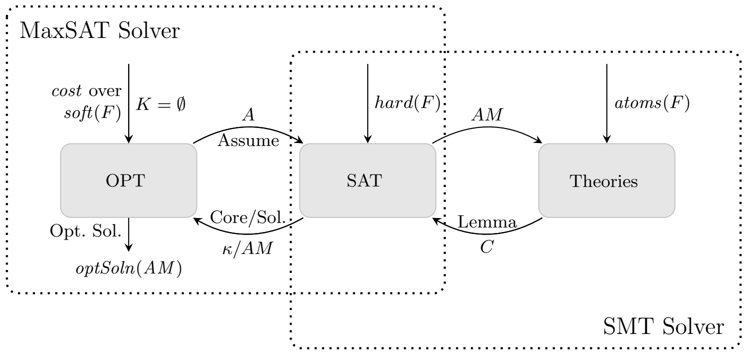

Fig. 1: General architecture for an IHS based MaxSMT solver

4 Generic Hitting Set based MaxSMT

Here we present a general framework that realizes the previously introduced ideas for IHS based MaxSMT solving. Following the desiderata presented in our introduction, we decompose the problem of MaxSMT into three sub-problems: optimization (over Boolean atoms), Boolean satisfiability, and theory reasoning. Although modern SMT solvers are equipped with efficient engines for arithmetic reasoning, in MaxSMT the optimization problem depends purely on the Boolean abstraction of the formula and thus delegating the task of optimization to a specialized solver can be more efficient [8]. Figure 1 shows a general architecture to solve MaxSMT as an implicit hitting set problem [18, 19]. The method combines three components that are responsible for our three subtasks: OPT, an optimizer for hitting set computation; SAT, a SAT solver for Boolean reasoning; and Theories, a set of theory solvers to perform theory reasoning. The framework expects as input a MaxSMT formula (F) with a satisfiable set of hard clauses. The SAT and Theory solvers consider only the hard clauses, while the soft (unit) clauses and their costs are only considered by the optimizer. Note that we can initially check the hard clauses for satisfiability. If they are unsatisfiable there is no optimal solution.

The evaluation starts with OPT, which computes a (potentially optimal) hitting set η of the current set of unsatisfiable cores (K). This is translated into a set of assumptions (noted as A in Fig. 1) that requires the satisfaction of all soft clauses not in η (see **HS** and **MinHS** steps of **A-MaxSMT**).

The SAT solver can then decide if there exists a feasible solution satisfying $\mathit{hard}(F)$ and the assumptions. Theory solvers can be invoked at various points to check the T-consistency of the SAT solver's current partial assignment AM and to perform T-learning. As in [12] there are a range of flexible (e.g., more eager or more lazy) strategies for deciding when theory reasoning should be invoked.

For a given conjunction of theory literals, a theory solver might return a subset that is T-unsatisfiable forming a **conflict clause** after negation, or additional theory clauses for T-learning. In both cases the returned clauses are valid **lemmas** of the theory (C in Fig. 1). The SAT and theory solvers continue their collaboration under the assumption of A until either a theory consistent model

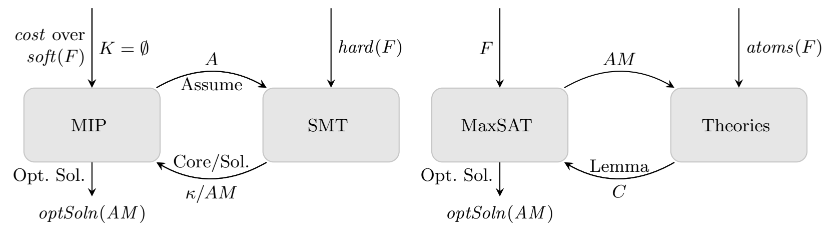

Fig. 2: Example merged (i.e. single-SAT) instantiations of our framework

of $hard(F) \wedge A$ is found (i.e. state $T\text{-}SAT(AM, F)$ is reached), or $hard(F) \wedge A$ is found to be unsatisfiable (i.e. state $conflict(F, C)$ is reached). In the latter case, the SAT solver constructs an unsatisfiable core (κ in Fig. 1) that consists of a subset of the soft clauses assumed to be satisfied in A. After that, the optimizer can compute a new hitting set that hits κ as well. Note that the new hitting set need not be of minimum cost. From the new hitting set, a new A is constructed and a new iteration starts. Any theory consistent model that is found for $hard(F) \wedge A$ is a feasible solution of the MaxSMT problem. The optimality of these solutions can be decided by the optimizer component based on their costs. In case the found solution is not optimal, a new hitting set is computed in order to find a better solution. Otherwise, the model is returned as a final optimal solution.

4.1 Possible Instantiations

A practical tool following our proposed general architecture can be achieved in various ways. Based on Fig. 1, one could combine a hitting set calculator with a SAT solver and a set of theory solvers. However, this implementation would not automatically benefit from the advanced techniques implemented in MaxSAT and SMT solvers nor from any future improvements to such solvers. So a more practical question is how to combine already existing tools to obtain a MaxSMT solver. Here we consider mixed integer programming solvers (MIP), for example CPLEX, for solving the minimum hitting set problem since they are widely available and display state of the art performance on a range of instances.

As Fig. 1 hints, some MaxSAT solvers already implement efficient collaboration between MIP and SAT solvers, while SMT solvers combine SAT and theory solvers. Combining these solvers as black-boxes results in an engine that contains two SAT solvers, while merging these engines results in a tool with a single SAT solver. Figure 2 shows two possible instances of the latter case. On the left, we keep an SMT solver as a black-box and combine it with a MIP solver that is responsible for the hitting sets (and so the assumptions) in each iteration. A benefit of this instance is efficient SMT solving and the ability to use the full power of the MIP solver to express complex objective functions (e.g., multi-objective optimization). One disadvantage is the lack of MaxSAT preprocessing and simplifications. A tighter combination could replace the SAT solver in an IHS based MaxSAT solver with an SMT solver. However, in IHS MaxSAT solving SAT

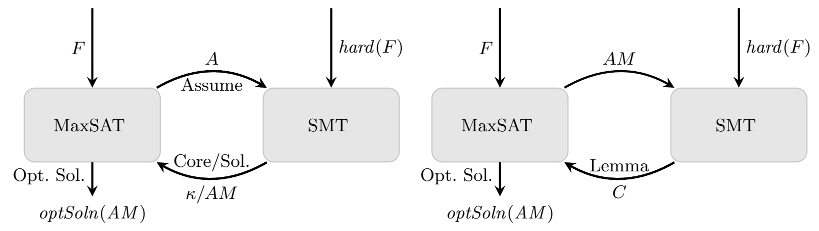

Fig. 3: Example combined (i.e. double-SAT) instantiations of our framework

calls are considered relatively cheap compared to MIP calls [20], but SMT calls can be more expensive, so the tradeoffs of some techniques would have to be reevaluated. The instance on the right side of Fig. 2 considers a (not necessarily IHS based) MaxSAT solver as a black-box to find an optimal solution for the abstraction of the problem, and forms a lazy lemmas on demand structure with a set of theory solvers for theory consistency checks. The benefit here is efficient optimization solving, but the disadvantage is delayed theory support.

Instantiations in Fig. 3 present possibilities for combining black-box (i.e. not necessarily IHS based) MaxSAT and SMT solvers, providing the advantage of efficient optimization and SMT solving at the same time. These combinations contain multiple SAT solvers where the connecting interface determines the work distribution among them. On the left side, the solvers communicate via assumptions and cores or solutions. Whenever the MaxSAT solver finds an optimal propositional model for its current problem, the SMT solver has to verify that the soft clauses satisfied in that model are also T-satisfiable (via a set of assumptions that forces their satisfaction). If not, it returns a new core to refine the MaxSAT problem. In practice, the effectiveness of this instance would be compromised if many iterations are needed to refine the MaxSAT model. Another possible disadvantage of this instance is that the SMT solver could learn lemmas that would be useful to the MaxSAT engine but are never passed to it.

An alternate instance (right side in Fig. 3) involves the MaxSAT solver giving to the SMT solver the complete propositional model it found (the optimal model for its current problem). If that model is not theory consistent the SMT solver can return any number of lemmas to refine the MaxSAT problem. In this instance the MaxSAT solver can learn theory related constraints from the SMT solver beyond unsatisfiable cores. The approach introduced in [3] can be seen as a combination of the instances in Fig. 3. There the MaxSAT optimal model is used to provide assumptions to the SMT solver (as in the left-hand instance), but the SMT solver can return many lemmas to the MaxSAT solver not just cores (as in the right-hand instance). A potential drawback of that approach is that the returned lemmas might or might not be useful to the MaxSAT solver, and there is a risk of overloading the MaxSAT solver.

Based on these instances, it appears that support of assumption based incremental solving and efficient extraction of small cores are important features of the involved tools. Thus techniques that improve these aspects of solvers (e.g., [21]) have the potential to improve modular MaxSMT solvers as well. Further, note that our calculus allows the interruption of SMT calls in certain cases (see conditions in Theorem 3), which may be worth considering in practice.

5 Related Work

As argued in the introduction the focus of this paper is on the important class of MaxSMT solvers. Thus this section will concentrate on the closest related approaches. Additional experimental results are provided in the next section.

We modify and extend a general DPLL(T) framework introduced in [12] to formalize our MaxSMT solving approach. Another extension of DPLL(T) by Nieuwenhuis and Oliveras in [2] represents the optimization task explicitly as a set of theory constraints and progressively strengthens this theory by deriving tighter bounds. Our extension of DPLL(T) focuses only on MaxSMT problems and separates the optimization task from theory reasoning.

A modular approach was proposed by Cimatti et al. in [3] where MaxSAT and SMT solvers are employed as black-boxes for MaxSMT solving. As we showed in Section 4.1, our framework includes this approach. In Section 6 we present some empirical results comparing their approach with other instantiations.

In the context of core-guided MaxSAT solving, SMT solvers have been used instead of SAT, e.g., [22], to handle cardinality constraints more efficiently. We focus on IHS based MaxSMT solving in which no cardinality constraints are introduced into the SMT sub-problems.

Manolios et al. introduced the theoretical underpinnings of a Branch and Cut Modulo Theories framework and developed an optimization procedure where integer linear programming (ILP) and stably-infinite theories are combined [7]. Our approach delegates Boolean reasoning to a SAT solver, while in their construction this is done by the ILP solver.

6 Experimental Evaluation

We implemented two instantiations of our framework. Both use MathSAT5 [23] version 5.5.1 as the SMT component. Our first implementation **maxhs-msat** follows the architecture proposed on the left side of Fig. 3. It combines maxHS 3.0 as the optimizer with MathSAT5. To evaluate the potential of lifting theory lemmas to the MaxSAT level, as proposed in [3] and described in Section 4.1 as a combination of the instances in Fig. 3, the configuration **maxhs-msat-ll** lifts and adds all used theory lemmas to the MaxSAT solver in addition to unsatisfiable cores. Our second solver **cplex-msat** implements the architecture shown on the left of Fig. 2 which combines MathSAT5 directly with a hitting set solver (CPLEX 12.7 as in maxHS 3.0) as the optimizer. In this implementation the components interface only with assumptions and unsatisfiable cores.

In both solvers the optimizers compute an optimal hitting set η. In **maxhs-msat** maxHS computes an optimal solution to its current Boolean abstraction, but the clauses falsified by that solution form an optimal hitting set. The SMT solver then tests if the other soft clauses ($\mathit{soft}(F) - \eta$) are T-satisfiable. If not, a new core is added to the optimizer (along with additional theory lemmas in **maxhs-msat-ll**). Following [20], rather than calling the optimizer in each iteration we allow non-optimal hitting sets. In particular, the new SMT core can be added to the previous hitting set (**-djnt**), or a single minimum weight clause from the new core can be added to the hitting set (**-min**). In both cases we obtain a new (non-minimum) hitting set covering the new core. For **cplex-msat** only, we can also use CPLEX to compute a linear programming solution of the hitting set problem which when rounded up yields a new hitting set (**-lp**). In these cases we continue to use non-minimum hitting sets η' until $\mathit{soft}(F) - \eta'$ becomes T-satisfiable, and then we again use the optimizer to compute a hitting set with minimum cost.

We compare against two state-of-the-art MaxSMT solvers. OptiMathSAT (version 1.4.5) [6] is a general purpose Optimization Modulo Theories (OMT) solver that we use in two different configurations. The default configuration is denoted by **optimathsat-omt**, while **optimathsat-maxres** employs the maximum resolution approach of [14]. We also compare against z3 (version 4.6.0) with two different MaxSAT engine configurations (**z3-maxres** and **z3-wmax**). Note, that the hitting set based engine in z3 has been deprecated and was removed.

We considered three sets of benchmarks from three different sources. The *LL-benchmark* set consists of all 398 quantifier free MaxSMT benchmarks used in [3] with annotations replaced by soft assertions, split into 212 benchmarks over the theory of linear integer arithmetic and 186 benchmarks over linear real arithmetic. For each theory, half the instances have **U**nit weight for soft assertions, while the other half contains **R**andom weights in the interval of 1 and 100. The runtime limit on these instances was set to 20 minutes.

Our second benchmark set *LEX-benchmark*, consisting of equalities over propositional atoms, are lexicographically-optimum realization problems used in [8]. We only considered the 6098 instances where three groups of soft assertions (**T**ime, **C**ost and **W**eight) have different priorities and the objective is to lexicographically minimize the sum of the falsified assertions with respect to a given priority order of **T**, **C**, **W**). The time limit was set to 100 seconds.

Finally, in order to further exercise the strengths of the different approaches, we generated a set of scaled problems from one (arbitrarily chosen) QF-LIA SMT-LIB benchmark family (Bofill-scheduling waste water treatment scheduling problems from [24]). The original family contained 156 randomly generated (referred as *rand-wwtp*) and 251 industrial (*ind-wwtp*) satisfiable SMT problems. We derived instances from these SMT problems by adding randomly chosen theory atoms with random polarity as unit soft clauses. The four groups of derived instances introduced four different percentages (10%, 25%, 50% and 100%) of the atoms in the original problem as soft assertions. All instances were generated once with unit weights and once more with random weights between 1 and the total number of atoms. Due to space constraints, we only present results on instances derived from *rand-wwtp* problems, where we observed an interesting pattern. The time limit was set to 5 minutes.

Table 4: Results of various solvers and configurations on LL-benchmarks from [3].

Solver	LIA(212)		LRA(186)		Total	SMT%	OPT%
	U	R	U	R			
cplex-msat	82	90	85	85	342	99.22%	0.13%
cplex-msat-djnt	85	**91**	85	85	346	98.83%	0.33%
cplex-msat-min	83	86	85	85	339	99.22%	0.04%
cplex-msat-lp	84	89	85	85	343	98.26%	0.97%
maxhs-msat	85	87	85	85	342	88.15%	11.20%
maxhs-msat-djnt	86	89	85	85	345	83.85%	15.36%
maxhs-msat-min	84	89	85	85	343	92.31%	7.04%
maxhs-msat-ll	80	84	83	78	325	82.57%	15.45%
maxhs-msat-ll-djnt	78	84	83	77	322	87.97%	10.37%
maxhs-msat-ll-min	79	86	82	85	332	80.13%	17.03%
optimathsat-maxres	**87**	90	85	86	**348**	–	–
optimathsat-omt	75	72	85	85	317	–	–
z3-maxres	73	79	86	85	323	–	–
z3-wmax	69	77	**88**	**88**	322	–	–

The experiments were performed on a cluster in which each computing node consisted of two Intel(R) Xeon(R) E5-2620 v4 @ 2.10GHz CPUs and 128 GB of main memory. We limited memory usage of each tool to 7GB on each instance and used different time limits as described above.

Table 4 presents results on the LL-benchmarks. For each solver configuration the first two columns list the number of solved instances with linear integer arithmetic as background theory, where the soft assertions have **U**nit weights in the first column and **R**andom weights in the second. Analogously, the next two columns present results in linear real arithmetic. The fifth column contains the total sum of solved instances in the previous four columns. The last two columns show the percentage of time spent in the SMT and in the optimization component (considering only solved instances). The optimization component in **cplex-msat** is CPLEX, while in **maxhs-msat** it is maxHS.

It turns out that **optimathsat-maxres** outperforms the other tools and configurations on these instances, but our implementations remain competitive. Furthermore, lemma lifting (**maxhs-msat-ll** and its different configurations) reduces the percentage time spent in SMT solving, but has a negative effect with respect to the number of solved instances compared to **maxhs-msat** and its different configurations. None of the involved tools appears to be sensitive to the type of weights (**U**niform vs. **R**andom). Although **cplex-msat** does not contain any MaxSAT preprocessing or simplification technique, the results of that tool in this experiment are similar to **maxhs-msat**.

Results on the LEX-benchmark are shown in Table 5. The 6098 problems contained two groups of problems. The first group of 3699 instances used the lexicographic preference ordering **C**ost, **T**ime and then **W**eight, and are shown in the first two columns which list the number of solved instances and the total run time used to solve them. The second group of 2399 instances used the reversed lexicographic preference and are shown in the next two columns. For our tools we also give the total number of unsatisfiable cores and of optimal hitting set calculations (considering again only solved instances) in the last two columns.

Table 5: Results of various solvers and configurations on LEX-benchmarks from [8].

Solver	CTW	Time[s]	WTC	Time[s]	cores	opt. HS
cplex-msat	3499	27825	2399	1942	1610031	1615150
cplex-msat-djnt	3687	5936	2399	1455	920387	137339
cplex-msat-min	3699	2479	2399	1391	909828	27245
cplex-msat-lp	3699	4564	2399	1493	1260683	19056
maxhs-msat	3699	2401	2399	1367	0	5319
maxhs-msat-djnt	3699	**2224**	2399	**1359**	0	5319
maxhs-msat-min	3699	2451	2399	1409	0	5319
maxhs-msat-ll	3699	2302	2399	1518	0	5319
maxhs-msat-ll-djnt	3699	2394	2399	1406	0	5319
maxhs-msat-ll-min	3699	2441	2399	1437	0	5319
optimathsat-maxres	3410	13851	1850	10209	–	–
optimathsat-omt	3481	9710	2068	10483	–	–
z3-maxres	3699	4555	2399	2231	–	–
z3-wmax	3651	5566	2295	9513	–	–

On these instances most versions of our approach solve at least as many problems as the state-of-the-art tools and in significantly less time. Due to the background theory of these instances it is enough to find a propositional model, i.e., solve a MaxSAT problem, since every propositional solution also happens to be T-satisfiable. This is reflected in the last two columns, where the two instantiations of our framework show different behaviour. For **maxhs-msat**, which combines maxHS with the SMT solver, the number of iterations is always one (in all 5319 satisfiable instances). In this case maxHS finds an optimal Boolean model (through several iterations of its *internal* SAT solver), which the SMT solver then verifies to be theory consistent in one call. In case of **cplex-msat** there is no additional SAT solver between the SMT and the optimization components. Therefore it has to learn all the necessary transitivity properties of the equalities in form of cores from the SMT solver. Thus the number of unsatisfiable cores is higher for **cplex-msat**, which can significantly increase solving time depending on the type of hitting sets used.

These benchmarks in essence allow us to compare the effectiveness of the optimization components independently of the SMT component. This benefits our hitting set based methods, while other solvers rely on alternative approaches. Another important difference is that our prototypes solve lexicographic problems as single objective functions in one run by aggregating the cost functions [25].

The last table (Table 6) presents results for the randomized rand-wwtp benchmarks on which **cplex-msat** performs better than **maxhs-msat**. Using non-minimum hitting sets measurably reduces the performance of both implementations on these instances. From the last two columns we can deduce that the best performing methods are those where more time was spent within the optimization component. Although lemma lifting does result in significant more time spent in maxHS calls, some part of it is spent in the SAT solver, and not in actual optimization. This might explain its bad performance.

To summarize, the experiments support the need for a generic framework for MaxSMT. More concretely we make the following three observations. First, there is no overall best configuration. Performance depends on the distribution

Table 6: Results of considered solvers and configurations on rand-wwtp family with 10%-100% random unit soft clauses. Each percentage group consists of 312 problems.

Solver	10%	25%	50%	100%	Total	SMT%	OPT%
cplex-msat	289	271	**203**	4	**767**	60.85%	38.46%
cplex-msat-djnt	286	247	114	2	649	97.35%	1.96%
cplex-msat-min	282	244	142	**16**	684	91.46%	7.68%
cplex-msat-lp	287	262	184	13	746	83.4%	15.27%
maxhs-msat	288	270	179	0	737	42.28%	57.31%
maxhs-msat-djnt	289	249	112	1	651	93.91%	5.69%
maxhs-msat-min	281	242	132	15	670	87.99%	11.59%
maxhs-msat-ll	266	166	16	0	448	7.69%	84.93%
maxhs-msat-ll-djnt	266	161	9	0	436	11.30%	77.59%
maxhs-msat-ll-min	263	166	27	0	456	11.36%	68.11%
optimathsat-maxres	291	258	123	0	672	–	–
optimathsat-omt	240	130	0	0	370	–	–
z3-maxres	280	224	103	0	607	–	–
z3-wmax	**304**	**288**	4	0	596	–	–

of the workload among the involved components, since in general the difficulty of the optimization and SMT problems differ. For instance, improved MaxSAT performance does not necessarily translate into improved MaxSMT performance, simply because of different relative costs between SMT calls and SAT calls. Accordingly, non-minimum hitting sets (like disjoint cores or LP relaxation) usually reduce the workload of the optimizer but put more stress on the SMT solver.

Second, the number of extracted unsatisfiable cores or calculated optimal hitting sets is not always an expedient metric to measure the performance of MaxSMT. Finally, most of the time, lemma lifting does not improve but actually seems to reduce performance of a modular MaxSMT solver, particularly with an implicit hitting set based approach.

All of our experimental results as well as the evaluated benchmarks are available at http://fmv.jku.at/maxsmt/.

7 Conclusion

We have proposed an abstract framework to gain a unifying view of how optimization, propositional reasoning, and theory reasoning can be combined in IHS based MaxSMT solving. Our framework is very flexible supporting a rich space of possible implementation architectures all of which are provably sound. Our empirical results show that different architectures yield quite different performance on different problems sets. This implies that there is considerable potential in more fully exploiting the flexibility of our framework to obtain improved and more robust performance in MaxSMT solvers.

Acknowledgments This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23.

References

1. Li, Y., Albarghouthi, A., Kincaid, Z., Gurfinkel, A., Chechik, M.: Symbolic optimization with SMT solvers. In: POPL, ACM (2014) 607–618
2. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: SAT. Volume 4121 of LNCS., Springer (2006) 156–169
3. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: A modular approach to MaxSAT modulo theories. In: SAT. Volume 7962 of LNCS., Springer (2013)
4. Sebastiani, R., Tomasi, S.: Optimization modulo theories with linear rational costs. ACM Trans. Comput. Log. **16**(2) (2015) 12:1–12:43
5. Bjørner, N., Phan, A., Fleckenstein, L.: νZ - an optimizing SMT solver. In: TACAS. Volume 9035 of LNCS., Springer (2015) 194–199
6. Sebastiani, R., Trentin, P.: OptiMathSAT: A tool for optimization modulo theories. In: CAV. (2015) 447–454
7. Manolios, P., Pais, J., Papavasileiou, V.: The Inez mathematical programming modulo theories framework. In: CAV. (2015) 53–69
8. Sebastiani, R., Trentin, P.: On optimization modulo theories, MaxSMT and sorting networks. In: TACAS. Volume 10206 of LNCS. (2017) 231–248
9. Ansótegui, C., Bacchus, F., Järvisalo, M., Martins, R.: MaxSAT Evaluation 2017 (2017) http://mse17.cs.helsinki.fi/.
10. Bacchus, F., Järvisalo, M.: Algorithms for maximum satisfiability with applications to AI. AAAI-16 Tutoral https://www.cs.helsinki.fi/group/coreo/aaai16-tutorial/
11. Davies, J., Bacchus, F.: Solving MaxSAT by solving a sequence of simpler SAT instances. In: CP. Volume 6876 of LNCS., Springer (2011) 225–239
12. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6) (2006) 937–977
13. Sebastiani, R.: Lazy satisability modulo theories. JSAT **3**(3-4) (2007) 141–224
14. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: AAAI, AAAI Press (2014) 2717–2723
15. Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: CP. Volume 8656 of LNCS., Springer (2014) 531–548
16. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196** (2013) 77–105
17. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT. (2003) 502–518
18. Chandrasekaran, K., Karp, R.M., Moreno-Centeno, E., Vempala, S.: Algorithms for implicit hitting set problems. In: SODA, SIAM (2011) 614–629
19. Saikko, P., Wallner, J.P., Järvisalo, M.: Implicit hitting set algorithms for reasoning beyond NP. In: KR, AAAI Press (2016) 104–113
20. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Principles and Practice of Constraint Programming. (2013) 247–262
21. Lagniez, J., Biere, A.: Factoring out assumptions to speed up MUS extraction. In: SAT. Volume 7962 of LNCS., Springer (2013) 276–292
22. Ansótegui, C., Gabàs, J., Levy, J.: Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J. Heuristics **22**(1) (2016) 1–53
23. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: TACAS. Volume 7795 of LNCS., Springer (2013) 93–107
24. Bofill, M., Muñoz, V., Murillo, J.: Solving the wastewater treatment plant problem with SMT. CoRR **abs/1609.05367** (2016)
25. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. AI **62**(3-4) (2011) 317–343