Satisfiability Modulo the Theory of Costs: Foundations and Applications ⋆

Alessandro Cimatti[1], Anders Franzén[1], Alberto Griggio[2], Roberto Sebastiani[2], and Cristian Stenico[2]

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

Abstract. We extend the setting of Satisfiability Modulo Theories (SMT) by introducing a theory of costs $\mathcal{C}$, where it is possible to model and reason about resource consumption and multiple cost functions, e.g., battery, time, and space. We define a decision procedure that has all the features required for the integration withint the lazy SMT schema: incrementality, backtrackability, construction of conflict sets, and deduction. This naturally results in an SMT solver for the disjoint union of $\mathcal{C}$ and any other theory $\mathcal{T}$.
This framework has two important applications. First, we tackle the problem of *Optimization Modulo Theories*: rather than checking the existence of a satisfying assignment, as in SMT, we require a satisfying assignment that minimizes a given cost function. We build on the decision problem for SMT with costs, i.e., finding a satisfying assigniment with cost within an admissibility range, and propose two algorithms for optimization. Second, we use multiple cost functions to deal with *PseudoBoolean constraints*. Within the SMT($\mathcal{C}$) framework, the effectively PseudoBoolean constraints are dealt with by the cost solver, while the other constraints are reduced to pure boolean reasoning.
We implemented the proposed approach within the MathSAT SMT solver, and we experimentally evaluated it on a large set of benchmarks, also from industrial applications. The results clearly demonstrate the potential of the approach.

1 Motivations and goals

Important verification problems are naturally encoded as Satisfiability Modulo Theory (SMT) problems, i.e. as satisfiability problems for decidable fragments of first order logic. Efficient SMT solvers have been developed, that combine the power of SAT solvers with dedicated decision procedures for several theories of practical interest.

In many practical domains, problems require modeling and reasoning about resource consumption and multiple cost functions, e.g., battery, time, and space. In this paper, we extend SMT by introducing a *theory of costs* $\mathcal{C}$. The language of the theory of costs is very expressive: it allows for multiple cost functions and, for each of these, arbitrary Boolean conditions may be stated to result in a given cost increase; costs may be

⋆ A. Cimatti is supported in part by the European Commission FP7-2007-IST-1-217069 COCONUT. A. Franzén, A. Griggio and R. Sebastiani are supported in part by SRC under GRC Custom Research Project 2009-TJ-1880 WOLFLING, and by MIUR under PRIN project 20079E5KM8_002.

be both lower- and upper-bounded, also depending on Boolean conditions. In the current paper, we concentrate on the case of Boolean cost functions, where costs are bound by and function of Boolean atoms (including relations over individual variables). For the theory of costs, it is possible to define a decision procedure that has all the features required for the integration within the lazy SMT schema: incrementality, backtrackability, construction of conflict sets, and deduction. This naturally results in a solver for SMT($\mathcal{C}$), and, given the assumption of Boolean cost functions, also in a solver for the disjoint union of $\mathcal{C}$ and any other theory $\mathcal{T}$.

Based on the theory of costs, we propose two additional contributions. First, we extend SMT by tackling the problem of *Optimization Modulo Theories*: rather than checking the existence of a satisfying assignment, as in SMT, we require a satisfying assignment that minimizes a given cost function. We build on the decision problem for SMT with costs, i.e. finding a satisfying assignment with cost within an admissibility range. The optimization problem is then tackled as a sequence of decision problems, where the admissibility range is adapted from one problem to the next. We propose two algorithms: one is based on branch-and-bound, where the admissibility range is increasingly tightened, based on the best valued solution so far. The other one, based on binary search, proceeds by bisecting the admissible interval, and leveraging under-approximations as well as over-approximations.

Second, we show how to exploit the feature of multiple cost functions to deal with the well known problem of *PseudoBoolean (PB) constraints*. The approach can be seen as dealing with PB problems as in an SMT paradigm. The PB constraints are dealt with by the cost solver (as if it were a solver for the theory of PB constraints), while the other constraints are reduced to pure Boolean reasoning. The approach is enabled by the ability of the cost theory to deal with multiple cost functions. The resulting solution is very elegant and extremely simple to implement.

We implemented the proposed approach within the MathSAT SMT solver. We experimentally evaluate it on a wide set of benchmarks, including artificial benchmarks (obtained by adding cost functions to the problems in SMT-LIB), real-world (from two different industrial domains) benchmarks, and benchmarks from the PB solver competition. The results show that the approach, despite its simplicity, is very effective: it is able to solve complex case studies in SMT($\mathcal{T} \cup \mathcal{C}$) and, despite its simplicity, it shows surprising efficiency in some Boolean and PB optimization problems, outperforming the winners of the most recent comptetition.

This paper is structured as follows. In §2 we present some background. The theory of costs and the decision procedure are presented in §3. In §4 we show how to tackle optimization problems. In §5 we show how to encode PseudoBoolean constraints as SMT($\mathcal{C}$). In §6 we experimentally evaluate our approach. In §7 we discuss some related approaches. In §8 we draw some conclusions and outline directions for future research. The proofs and some possibly-useful material are reported in [8].

2 Background on SMT and SMT solving

Satisfiability Modulo (the) Theory $\mathcal{T}$, *SMT*($\mathcal{T}$), is the problem of deciding the satisfiability of (typically) ground formulas under a background theory $\mathcal{T}$. (Notice that $\mathcal{T}$ can

also be a combination of simpler theories: $\mathcal{T} \stackrel{\text{def}}{=} \bigcup_i \mathcal{T}_i$.) We call an *SMT($\mathcal{T}$) solver* any tool able to decide SMT($\mathcal{T}$). We call a *theory solver for $\mathcal{T}$*, *$\mathcal{T}$-solver*, any tool able to decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of ground atomic formulas and their negations (*$\mathcal{T}$-literals*). If the input set of $\mathcal{T}$-literals μ is $\mathcal{T}$-unsatisfiable, then *$\mathcal{T}$-solver* returns unsat and the subset η of $\mathcal{T}$-literals in μ which was found $\mathcal{T}$-unsatisfiable; (η is hereafter called a *$\mathcal{T}$-conflict set*, and $\neg\eta$ a *$\mathcal{T}$-conflict clause*.) if μ is $\mathcal{T}$-satisfiable, then *$\mathcal{T}$-solver* returns sat; it may also be able to return some unassigned $\mathcal{T}$-literal l s.t. $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, where $\{l_1, ..., l_n\} \subseteq \mu$. We call this process $\mathcal{T}$-deduction and $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ a *$\mathcal{T}$-deduction clause*. Notice that both $\mathcal{T}$- and $\mathcal{C}$-conflict and $\mathcal{T}$-deduction clauses are valid in $\mathcal{T}$. We call them *$\mathcal{T}$-lemmas*.

We adopt the following terminology and notation. The bijective function $\mathcal{T}2\mathcal{B}$ ("$\mathcal{T}$-to-Boolean"), called *Boolean abstraction*, maps propositional variables into themselves, ground $\mathcal{T}$-atoms into fresh propositional variables, and is homomorphic w.r.t. Boolean operators and set inclusion. The symbols φ, ψ, ϕ denote $\mathcal{T}$-formulas, and μ, η denote sets of $\mathcal{T}$-literals. If $\mathcal{T}2\mathcal{B}(\mu) \models \mathcal{T}2\mathcal{B}(\varphi)$, then we say that μ *propositionally satisfies* φ written $\mu \models_p \varphi$. With a little abuse of terminology, we will often omit specifying "the Boolean abstraction of" when referring to propositional reasoning steps, as if these steps were referred to the ground $\mathcal{T}$-formula/assignment/clause rather than to their Boolean abstraction. (E.g., we say "φ is given in input to DPLL" rather "$\mathcal{T}2\mathcal{B}(\varphi)$ is..." or "μ is a truth assignment for φ" rather than "$\mathcal{T}2\mathcal{B}(\mu)$ is a truth assignment for $\mathcal{T}2\mathcal{B}(\varphi)$".) This is done w.l.o.g. since $\mathcal{T}2\mathcal{B}$ is bijective.

In a lazy SMT($\mathcal{T}$) solver the truth assignments for φ are enumerated and checked for $\mathcal{T}$-satisfiability, returning either sat if one $\mathcal{T}$-satisfiable truth assignment is found, unsat otherwise. In practical implementations, φ is given as input to a modified version of DPLL, and when an assignment μ is found s.t. $\mu \models_p \varphi$ μ is fed to the *$\mathcal{T}$-solver*; if μ is $\mathcal{T}$-consistent, then φ is $\mathcal{T}$-consistent; otherwise, *$\mathcal{T}$-solver* returns the conflict set η causing the inconsistency. Then the $\mathcal{T}$-conflict clause $\neg\eta$ is fed to the backjumping and learning mechanism of DPLL (*$\mathcal{T}$-backjumping* and *$\mathcal{T}$-learning*).

Important optimizations are *early pruning* and *$\mathcal{T}$-propagation*: the $\mathcal{T}$-solver is invoked also on an intermediate assignment μ: if it is $\mathcal{T}$-unsatisfiable, then the procedure can backtrack; if not, and if the *$\mathcal{T}$-solver* performs 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. 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., [6], for details and further references.

3 Satisfiability Modulo the Theory of Costs

3.1 Modeling cost functions

We extend the SMT framework by adding *cost functions*. Let $\mathcal{T}$ be a first-order theory. We consider a pair $\langle \varphi, \underline{costs} \rangle$, s.t. $\underline{costs} \stackrel{\text{def}}{=} \{cost^i\}_{i=1}^{M}$ is an array of M integer cost functions over $\mathcal{T}$ and φ is a Boolean combination on ground $\mathcal{T}$-atoms and atoms in the form $(cost^i \leq \mathsf{c})$ s.t. c is some integer value. [3] We focus on problems in which each

[3] Notice that every atom in the form $(cost^i \bowtie \mathsf{c})$ s.t. $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$ can be expressed as a Boolean combination of constraints in the form $(cost^i \leq \mathsf{c})$.

$cost^i$ is a *Boolean cost function* in the form

$$cost^i = \sum_{j=1}^{N_i} ite(\psi_j^i, \mathsf{c}_{j1}^i, \mathsf{c}_{j2}^i), \tag{1}$$

s.t., for every i, ψ_j^i is a formula in $\mathcal{T}$ and c_{j1}^i,c_{j2}^i are integer constants values and ite (term if-then-else) is a function s.t. ite (A_j^i,c_{j1}^i,c_{j2}^i) returns c_{j1}^i if A_j^i holds, c_{j2}^i otherwise. We notice that the problem is very general, and it can express a wide amount of different interesting problems, as it will be made clear in the next sections.

Hereafter w.l.o.g. we can restrict our attention to problems $\langle \varphi, \underline{costs} \rangle$ in which:

$$cost^i = \sum_{j=1}^{N_i} ite(A_j^i, \mathsf{c}_j^i, 0), \tag{2}$$

for every i, s.t. for every j, A_j^i is a Boolean literal and $0 < \mathsf{c}_j^i \leq \mathsf{c}_{j+1}^i$. (Passing from (1) to (2) is straighforward [8].) We denote by bound_{max}^i the value $\sum_{j=1}^{N_i} \mathsf{c}_j^i$, for every i.

We notice that we can easily encode (2) into subformulas in the theory of linear arithmetic over the integers ($\mathcal{LA}(\mathbb{Z})$) and hence the whole problem $\langle \varphi, \underline{costs} \rangle$ into a ground $\mathcal{T} \cup \mathcal{LA}(\mathbb{Z})$-formula, where $\mathcal{T}$ and $\mathcal{LA}(\mathbb{Z})$ are completely-disjoint theories, and have it solved by an SMT solver. Unfortunately this technique is inefficient in practice. (An explanation of this fact is reported in [8].) Instead, we cope with this problem by defining an ad-hoc theory of costs.

3.2 A theory of costs $\mathcal{C}$

We address the problem by introducing a "theory of costs" $\mathcal{C}$ consisting in:

- a collection of M fresh variables $c^1, \ldots, c^M$, denoting the output of the functions $cost^1$,...,$cost^M$;
- a fresh binary predicate BC ("bound cost"), s.t. $\mathsf{BC}(c^i, \mathsf{c})$ mean "$(c^i \leq \mathsf{c})$", c^i and c are one of the cost variables and an integer value respectively;
- a fresh ternary predicate IC ("incur cost"), s.t. $\mathsf{IC}(c^i, j, \mathsf{c}_j^i)$ means "c_j^i is added to c^i as jth element in the sum (2)", c^i, j, and c_j^i being one of the cost variables, an integer value denoting the index in the sum (2), and the corresponding integer value respectively. [4] We introduce exactly $\sum_{i=1}^{M} N_i$ distinct atoms $\mathsf{IC}(c^i, j, \mathsf{c}_j^i)$, one for each c_j^i in (2).

We call *$\mathcal{C}$-atoms* all atoms in the form $\mathsf{BC}(c^i, \mathsf{c})$, $\mathsf{IC}(c^i, j, \mathsf{c}_j^i)$, and *$\mathcal{C}$-literals* all $\mathcal{C}$-atoms and their negations. We call a *$\mathcal{T} \cup \mathcal{C}$-formula* any Boolean combination of ground $\mathcal{T}$- and $\mathcal{C}$-atoms (simply *$\mathcal{C}$-formula* if $\mathcal{T}$ is pure Boolean logic).

Intuitively, the theory of costs allows for modeling domains with multiple costs c^i by means of $\mathcal{C}$- or $\mathcal{T} \cup \mathcal{C}$-formulas. For instance, in the domain of planning for an autonomous rover, different costs may be battery consumption, and elapsed time. In the theory of costs, IC statements can be used to state the cost associated to specific partial configurations. For instance, a specific drilling action may require 5 seconds and 20mAh of battery energy, $Drill \rightarrow (\mathsf{IC}(battery, 1, 5) \wedge \mathsf{IC}(time, 1, 20))$, while moving

[4] Notice that the index j in $\mathsf{IC}(c^i, j, \mathsf{c}_j^i)$ is necessary to avoid using the same predicate for two constants c_j^i and $\mathsf{c}_{j'}^i$ with the same value but different indexes j, j'.

can have different impact on the available resources: $Move \rightarrow (\mathsf{IC}(battery, 2, 20) \wedge \mathsf{IC}(time, 2, 10))$. The BC predicates can be used to state for instance that the achievement of a certain goal G_1 should not require more than a certain amount of energy, while another goal should always be completed within a certain time bound and with a certain energy consumption: $G_1 \rightarrow \mathsf{BC}(battery, 20)$ and $G_2 \rightarrow \mathsf{BC}(battery, 20) \wedge \mathsf{BC}(time, 10)$. Notice that it is also possible to state lower bounds, e.g. the overall plan should never take less than a certain amount of time.

We consider a generic set $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{B}} \cup \mu_{\mathcal{T}} \cup \mu_{\mathcal{C}}$ s.t. $\mu_{\mathcal{B}}$ is a set of Boolean literals, $\mu_{\mathcal{T}}$ is a set of $\mathcal{T}$-literals, and $\mu_{\mathcal{C}} \stackrel{\text{def}}{=} \bigcup_{i=1}^{M} \mu^i_{\mathcal{C}}$ is a set of $\mathcal{C}$-literals s.t., for every i, $\mu^i_{\mathcal{C}}$ is:

$$\mu^i_{\mathcal{C}} \stackrel{\text{def}}{=} \{\mathsf{BC}(c^i, \mathsf{ub}^i_{(k)}) \mid k \in [1, ...K_i]\} \cup \{\neg\mathsf{BC}(c^i, \mathsf{lb}^i_{(m)} - 1) \mid m \in [1, ..., M_i]\} \quad (3)$$

$$\cup \{\mathsf{IC}(c^i, j, \mathsf{c}^i_j) \mid j \in J^{i+}\} \cup \{\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j) \mid j \in J^{i-}\}, \quad (4)$$

where $\mathsf{ub}^i_{(1)}, \ldots, \mathsf{ub}^i_{(K_i)}, \mathsf{lb}^i_{(1)}, \ldots, \mathsf{lb}^i_{(M_i)}$ are positive integer values, J^{i+} and J^{i-} are two sets of indices s.t. $J^{i+} \cap J^{i-} = \emptyset$ and $J^{i+} \cup J^{i-} \subseteq \{1, \ldots, N_i\}$, and no literal occurs both positively and negatively in μ. We say $\mu^i_{\mathcal{C}}$ is *total* if $J^{i+} \cup J^{i-} = \{1, \ldots, N_i\}$, *partial* otherwise. Notice that every truth assignment μ for a SMT($\mathcal{T} \cup \mathcal{C}$) formula φ is in the form $\mu_{\mathcal{B}} \cup \mu_{\mathcal{T}} \cup \mu_{\mathcal{C}}$ described above, s.t. $\mu_{\mathcal{B}}$, $\mu_{\mathcal{T}}$ and $\mu_{\mathcal{C}}$ are the restriction of μ to its Boolean, $\mathcal{T}$- and $\mathcal{C}$-literals respectively.

Let $\mathsf{lb}^i_{max} \stackrel{\text{def}}{=} max(\{\mathsf{lb}^i_{(1)}, \ldots, \mathsf{lb}^i_{(M_i)}\})$ and $\mathsf{ub}^i_{min} \stackrel{\text{def}}{=} min(\{\mathsf{ub}^i_{(1)}, \ldots, \mathsf{lb}^i_{(K_i)}\})$.

Definition 1. *If $\mu^i_{\mathcal{C}}$ is total, we say that $\mu^i_{\mathcal{C}}$ is* $\mathcal{C}$-consistent *if and only if*

$$\mathsf{lb}^i_{max} \leq \textstyle\sum_{j \in J^{i+}} \mathsf{c}^i_j \leq \mathsf{ub}^i_{min}. \quad (5)$$

If $\mu^i_{\mathcal{C}}$ is partial, we say that $\mu^i_{\mathcal{C}}$ is $\mathcal{C}$-consistent *if and only if there exists a total $\mathcal{C}$-consistent superset of $\mu^i_{\mathcal{C}}$, that is, a $\mathcal{C}$-consistent set $\mu^{i*}_{\mathcal{C}}$ in the form*

$$\mu^i_{\mathcal{C}} \cup \{\mathsf{IC}(c^i, j, \mathsf{c}^i_j) \mid j \in K^{i+}\} \cup \{\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j) \mid j \in K^{i-}\} \quad (6)$$

s.t. $(J^{i+} \cup K^{i+}) \cap (J^{i-} \cup K^{i-}) = \emptyset$ and $(J^{i+} \cup K^{i+} \cup J^{i-} \cup K^{i-}) = \{1, \ldots, N_i\}$.

Proposition 1. *If $\mathsf{lb}^i_{max} > \mathsf{ub}^i_{min}$, then $\mu^i_{\mathcal{C}}$ is $\mathcal{C}$-inconsistent.*

Proposition 2. *A partial set $\mu^i_{\mathcal{C}}$* (3)-(4) *is $\mathcal{C}$-consistent only if the following two conditions hold:*

$$\textstyle\sum_{j \in J^{i+}} \mathsf{c}^i_j \leq \mathsf{ub}^i_{min} \quad (7)$$

$$(\mathsf{bound}^i_{max} - \textstyle\sum_{j \in J^{i-}} \mathsf{c}^i_j) \stackrel{def}{=} \textstyle\sum_{j \in \{1,\ldots,N_i\} \setminus J^{i-}} \mathsf{c}^i_j \geq \mathsf{lb}^i_{max}. \quad (8)$$

Intuitively, if $\mu^i_{\mathcal{C}}$ violates (8), it cannot be expanded into a $\mathcal{C}$-consistent set by adding positive or negative $\mathcal{C}$-literals. Notice that if $\mu^i_{\mathcal{C}}$ is total, then $\{1, \ldots, N_i\} \setminus J^{i-}$ is J^{i+}, so that (7) and (8) collapse into the right and left part of (5) respectively. Hereafter we call $\sum_{j \in J^{i+}} \mathsf{c}^i_j$ the *i-th cost* of μ, denoted as $\mathsf{CostOf_i}(\mu)$ or $\mathsf{CostOf_i}(\mu^i_{\mathcal{C}})$, and we call $(\mathsf{bound}^i_{max} - \sum_{j \in J^{i-}} \mathsf{c}^i_j)$ the *maximum possible i-th cost* of μ, denoted as $\mathsf{MCostOf_i}(\mu)$ or $\mathsf{MCostOf_i}(\mu^i_{\mathcal{C}})$.

The notion of $\mathcal{C}$-consistency is extended to the general assignment μ above as follows. We say that $\mu_\mathcal{C} \stackrel{\text{def}}{=} \bigcup_{i=1}^{M} \mu^i_\mathcal{C}$ is $\mathcal{C}$-consistent if and only if $\mu^i_\mathcal{C}$ is $\mathcal{C}$-consistent for every i. We say that $\mu \stackrel{\text{def}}{=} \mu_\mathcal{B} \cup \mu_\mathcal{T} \cup \mu_\mathcal{C}$ is $\mathcal{T} \cup \mathcal{C}$-consistent if and only if $\mu_\mathcal{T}$ is $\mathcal{T}$-consistent and $\mu_\mathcal{C}$ is $\mathcal{C}$-consistent. (Notice that $\mu_\mathcal{B}$ is consistent by definition.) A $\mathcal{T} \cup \mathcal{C}$-formula is $\mathcal{T} \cup \mathcal{C}$-satisfiable if and only if there exists a $\mathcal{T} \cup \mathcal{C}$-satisfiable assignment μ defined as above which propositionally satisfies it.

3.3 A decision procedure for the theory of costs: *$\mathcal{C}$-solver*

We add to the SMT($\mathcal{T}$) solver one theory solver for the theory of costs $\mathcal{C}$ (*$\mathcal{C}$-solver* hereafter). *$\mathcal{C}$-solver* takes as input a truth assignment $\mu \stackrel{\text{def}}{=} \mu_\mathcal{B} \cup \mu_\mathcal{T} \cup \mu_\mathcal{C}$ selecting only the $\mathcal{C}$-relevant part $\mu_\mathcal{C} \stackrel{\text{def}}{=} \bigcup_{i=1}^{M} \mu^i_\mathcal{C}$, and for every i, it checks whether $\mu^i_\mathcal{C}$ is $\mathcal{C}$-satisfiable according to Propositions 1 and 2. This works as follows:

1. if $\mathsf{lb}^i_{max} > \mathsf{ub}^i_{min}$, then *$\mathcal{C}$-solver* returns unsat and the $\mathcal{C}$-conflict clause
$$\mathsf{BC}(c^i, \mathsf{lb}^i_{max} - 1) \vee \neg\mathsf{BC}(c^i, \mathsf{ub}^i_{min}); \tag{9}$$
2. if $\mathsf{CostOf_i}(\mu^i_\mathcal{C}) > \mathsf{ub}^i_{min}$, then *$\mathcal{C}$-solver* returns unsat and the $\mathcal{C}$-conflict clause
$$\neg\mathsf{BC}(c^i, \mathsf{ub}^i_{min}) \vee \bigvee\nolimits_{j \in K^{i+}} \neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j) \tag{10}$$
where K^{i+} is a minimal subset of J^{i+} s.t. $\sum_{j \in K^{i+}} \mathsf{c}^i_j > \mathsf{ub}^i_{min}$;
3. if $\mathsf{MCostOf_i}(\mu^i_\mathcal{C}) < \mathsf{lb}^i_{max}$, then *$\mathcal{C}$-solver* returns unsat and the $\mathcal{C}$-conflict clause
$$\mathsf{BC}(c^i, \mathsf{lb}^i_{max} - 1) \vee \bigvee\nolimits_{j \in K^{i-}} \mathsf{IC}(c^i, j, \mathsf{c}^i_j) \tag{11}$$
where K^{i-} is a minimal subset of J^{i-} s.t. $\sum_{j \in K^{i-}} \mathsf{c}^i_j < \mathsf{lb}^i_{max}$;

If neither condition above is verified for every i, then $\mathcal{C}$-solver returns sat, and the current values of c^i (i.e., $\mathsf{CostOf_i}(\mu)$) for every i. In the latter case, theory propagation for $\mathcal{C}$ ($\mathcal{C}$-propagation hereafter) can be performed as follows:

4. every unassigned literal $\mathsf{BC}(c^i, \mathsf{ub}^i_{(r)})$ s.t. $\mathsf{ub}^i_{(r)} \geq \mathsf{ub}^i_{min}$ and every unassigned literal $\neg\mathsf{BC}(c^i, \mathsf{lb}^i_{(s)} - 1)$ s.t. $\mathsf{ub}^i_{(s)} \leq \mathsf{lb}^i_{max}$ can be returned ("$\mathcal{C}$-deduced"). It is possible to build the corresponding $\mathcal{C}$-deduction clause by applying step 1. to $\mu^i_\mathcal{C} \cup \{\mathsf{BC}(c^i, \mathsf{ub}^i_{(r)})\}$ and $\mu^i_\mathcal{C} \cup \{\neg\mathsf{BC}(c^i, \mathsf{lb}^i_{(s)} - 1)\}$ respectively;
5. if $\mathsf{CostOf_i}(\mu^i_\mathcal{C}) \leq \mathsf{ub}^i_{min}$ but $\mathsf{CostOf_i}(\mu^i_\mathcal{C} \cup \{\mathsf{IC}(c^i, j, \mathsf{c}^i_j)\}) > \mathsf{ub}^i_{min}$ for some $j \notin (J^{i+} \cup J^{i-})$, then $\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ is $\mathcal{C}$-deduced. It is possible to build the corresponding $\mathcal{C}$-deduction clause by applying step 2. to $\mu^i_\mathcal{C} \cup \{\mathsf{IC}(c^i, j, \mathsf{c}^i_j)\}$;
6. if $\mathsf{MCostOf_i}(\mu^i_\mathcal{C}) \geq \mathsf{lb}^i_{max}$ but $\mathsf{MCostOf_i}(\mu^i_\mathcal{C} \cup \{\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j)\}) < \mathsf{lb}^i_{max}$ for some $j \notin (J^{i+} \cup J^{i-})$, then $\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ is $\mathcal{C}$-deduced. It is possible to build the corresponding $\mathcal{C}$-deduction clause by applying step 3. to $\mu^i_\mathcal{C} \cup \{\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j)\}$.

Notice that Propositions 1 and 2 describe conditions which are *necessary* but *not sufficient* to guarantee the $\mathcal{C}$-consistency of a *partial* set $\mu^i_\mathcal{C}$, and that Steps 1.-3. return

unsat if one such condition is violated, sat otherwise. Thus, if $\mathcal{C}$-*solver* returns sat, this does not necessary mean that $\mu^i_{\mathcal{C}}$ is $\mathcal{C}$-consistent, unless $\mu^i_{\mathcal{C}}$ is *total*. To this extent, if $\mu^i_{\mathcal{C}}$ is a partial $\mathcal{C}$-inconsistent set which does not violate the consistency conditions of Propositions 1 and 2, it is even possible that $\mathcal{C}$-*solver* returns sat and $\mathcal{C}$-deduces $\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ by step 5. and $\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ by step 6. for some j, which forces DPLL to detect the inconsistency.

$\mathcal{C}$-*solver* can be easily implemented in order to meet all the standard requirements for integration within the lazy SMT schema. In particular, it can work incrementally, by updating $\mathsf{CostOf_i}(\mu)$ and $\mathsf{MCostOf_i}(\mu)$ each time a literal in the form $\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ or $\neg\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ is added to μ. Detecting the $\mathcal{C}$-inconsistency inside $\mathcal{C}$-*solver* is computationally very cheap, since it consists in performing only one sum and one comparison each time a $\mathcal{C}$-literals is incrementally added to $\mu_{\mathcal{C}}$. (Thus, is O(1) for every incremental call.) The cost of performing $\mathcal{C}$-propagation is linear in the number of literals propagated (in fact, it suffices to scan the $\mathsf{IC}(c^i, j, \mathsf{c}^i_j)$ literals in decreasing order, until none is $\mathcal{C}$-propagated anymore).

3.4 A SMT($\mathcal{T} \cup \mathcal{C}$)-solver

A SMT($\mathcal{T} \cup \mathcal{C}$)-solver can be implemented according to the standard lazy SMT architecture. Since the theories $\mathcal{T}$ and $\mathcal{C}$ have no logic symbol in common ($\mathcal{C}$ does not have equality) so that they do not interfere to each other, $\mathcal{T}$-*solver* and $\mathcal{C}$-*solver* can be run independently. In its basic version, a SMT($\mathcal{T} \cup \mathcal{C}$) solver works as follows: an internal DPLL solver enumerates truth assignments propositionally satisfying $\varphi_{\mathcal{C}}$ and both $\mathcal{C}$-*solver* and $\mathcal{T}$-*solver* are invoked on on each of them. If both return sat, then the problem is satisfiable. (Notice that the algorithm concludes that $\varphi_{\mathcal{C}}$ is $\mathcal{T} \cup \mathcal{C}$-satisfiable only if all $\mu^i_{\mathcal{C}}$'s in the assignment μ are total.) If one of the two solvers returns unsat and a conflict clause C, then C is used as a theory conflict clause by the rest of the SMT($\mathcal{T} \cup \mathcal{C}$) solver for theory-driven backjumping and learning. The correctness an completeness of this process is a direct consequence of that of the standard lazy SMT paradigm, of Proposition 3 and of the definitions of $\mathcal{C}$- and $\mathcal{T} \cup \mathcal{C}$-consistency. As with plain SMT($\mathcal{T}$), the SMT($\mathcal{T} \cup \mathcal{C}$) solver can be enhanced by means of early-pruning calls to $\mathcal{C}$-*solver* and $\mathcal{C}$-propagation.

4 Optimization Modulo Theories via SMT($\mathcal{T} \cup \mathcal{C}$)

In what follows, we consider the problem of finding a satisfying assignment to an SMT($\mathcal{T}$) formula which is subject to some bound constraints to some Boolean cost functions and such that one Boolean cost function is minimized. We refer to this problem as *Boolean Optimization Modulo Theory (BOMT)*. We first address the decision problem (§4.1) and then the minimization problem (§4.2).

4.1 Addressing the SMT($\mathcal{T}$) cost decision problem

An SMT($\mathcal{T}$) cost decision problem is a triple $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ s.t. φ is a $\mathcal{T}$-formula, $\underline{costs}$ are in the form (2), and $\underline{\mathsf{bounds}} \stackrel{\text{def}}{=} \{\langle \mathsf{lb}^i, \mathsf{ub}^i \rangle\}_{i=1}^{M}$, where lb^i, ub^i are integer

```
1. int IncLinearBOMT(φ_C, c^1)
2.     mincost = +∞; bound = ub^1;
3.     do
4.         ⟨status, cost⟩ = IncrementalSMT_{T∪C}(φ_C, BC(c^1, bound));
5.         if (status==sat)
6.         then {
7.             φ_C = φ_C ∧ BC(c^1, bound); //activates learned C-lemmas
8.             mincost = cost;
9.             bound = cost-1; }
10.    while (status==sat);
11.    return mincost;
```

Fig. 1. A BOMT($\mathcal{T}$) algorithm based on linear search and incremental SMT($\mathcal{T} \cup \mathcal{C}$).

values s.t. $0 \leq \mathsf{lb}^i \leq \mathsf{ub}^i \leq \mathsf{bound}^i_{max}$. We call lb^i, ub^i and $[\mathsf{lb}^i, ..., \mathsf{ub}^i]$ the *lower bound*, the *upper bound* and the *range* of $cost^i$ respectively. (If some of the lb^i's and ub^i's are not given, we set w.l.o.g. $\mathsf{lb}^i = 0$ and $\mathsf{ub}^i = \mathsf{bound}^i_{max}$.)

We encode the decision problem $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ into the SMT($\mathcal{T} \cup \mathcal{C}$)-satisfiability problem of the following formula:

$$\varphi_{\mathcal{C}} \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{i=1}^{M} \Big(\mathsf{BC}(c^i, \mathsf{ub}^i) \wedge \neg \mathsf{BC}(c^i, \mathsf{lb}^i - 1) \wedge \bigwedge_{j=1}^{N_i} (A^i_j \leftrightarrow \mathsf{IC}(c^i, j, \mathsf{c}^i_j)) \Big) \quad (12)$$

Proposition 3. *A decision problem $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ has a solution if and only if $\varphi_{\mathcal{C}}$ is $\mathcal{T} \cup \mathcal{C}$-satisfiable. In such a solution, for every i, the value of c^i is $\mathsf{CostOf}_{\mathsf{i}}(\mu)$, μ being the $\mathcal{T} \cup \mathcal{C}$-satisfiable truth assignment satisfying $\varphi_{\mathcal{C}}$.*

4.2 Addressing the SMT($\mathcal{T}$) cost minimization problem

A SMT($\mathcal{T}$) cost minimization problem is a triple $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ s.t. φ, $\underline{costs}$ and $\underline{\mathsf{bounds}}$ are as in §4.1; the problem consists in finding one of the $\mathcal{T}$-models for φ whose value of $cost^1$ is minimum. We call $cost^1$ the *goal cost function*. (That is, we adopt the convention of considering the goal function the first function.) Using SMT($\mathcal{T} \cup \mathcal{C}$), we addressing BOMT($\mathcal{T}$) with two approaches, one based on linear-search/branch&bound, and the other based on binary-search.

A Linear search/branch&bound approach. Consider the $\mathcal{T} \cup \mathcal{C}$-formula $\varphi_{\mathcal{C}}$ in (12). In linear search, if $\varphi_{\mathcal{C}}$ is found $\mathcal{T} \cup \mathcal{C}$-satisfiable with a certain cost $cost$ for the cost variable c^1, we know that the minimum value of c^1 is at most $cost$. We can then conjunct $\mathsf{BC}(c^1, cost - 1)$ to $\varphi_{\mathcal{C}}$ and try again. We repeat this step until the formula is unsatisfiable, and then the last solution found is optimal.

The pseudo-code of the algorithm can be seen in Fig. 1. The procedure receives as input the formula $\varphi_{\mathcal{C}}$ as in (12) and the cost variable c^1 to minimize. Initially, $bound$ is set to the value ub^1 in (12). Each call IncrementalSMT$_{\mathcal{T} \cup \mathcal{C}}(\varphi_{\mathcal{C}}, \mathsf{BC}(c^1, bound))$ is a call to an incremental SMT($\mathcal{T} \cup \mathcal{C}$)-solver, which asserts $\mathsf{BC}(c^1, bound)$ before starting the

```
 1. int IncBinaryBOMT(φ_C, c^1)
 2.     lower = lb^1; upper = ub^1;
 3.     mincost = +∞; guess = ub^1;
 4.     do
 5.         ⟨status, cost⟩ = IncrementalSMT_{T∪C}(φ_C, BC(c^1, guess));
 6.         if (status==sat)
 7.         then {
 8.             φ_C = φ_C ∧ BC(c^1, guess); // activates learned C-lemmas
 9.             mincost = cost;
10.             upper = cost-1; }        // more efficient than guess-1
11.         else {
12.             φ_C = φ_C ∧ ¬BC(c^1, guess); // activates learned C-lemmas
13.             lower = guess +1; }
14.         guess = ⌊(lower + upper)/2⌋;
15.     while (lower ≤ upper);
16.     return mincost;
```

Fig. 2. A BOMT($\mathcal{T}$) algorithm based on binary search and incremental SMT($\mathcal{T} \cup \mathcal{C}$).

search, returning unsat if $\varphi_\mathcal{C} \wedge \mathsf{BC}(c^1, bound)$ is $\mathcal{T} \cup \mathcal{C}$-inconsistent, sat plus the value $cost \stackrel{\text{def}}{=} \mathsf{CostOf}_1(\mu)$, μ being the $\mathcal{T} \cup \mathcal{C}$-consistent satisfying assignment, otherwise. The fact of having an *incremental* SMT($\mathcal{T} \cup \mathcal{C}$) solver is crucial for efficiency, since it can reuse the Boolean, $\mathcal{T}$- and $\mathcal{C}$-lemmas learned in the previous iterations to prune the search. To this extent, the fact of explicitly conjoining $\mathsf{BC}(c^1, bound)$ to $\varphi_\mathcal{C}$ (7.) is not necessary for correctness, but it allows for reusing the $\mathcal{C}$-lemmas (10) from one call to the other to prune the search.

Termination is straightforward, since $mincost$ is a suitable ranking function. The correctness and completeness of the algorithm is also straightforward: in the last iteration we prove that there exist no solution better than the current value of $mincost$, so $mincost$ is optimal. (If no solution exists, then the procedure returns $+\infty$.)

A binary-search approach. A possibly faster way of converging on the optimal solution is binary search over the possible solutions. Instead of tightening the upper bound with the last solution found, we keep track of the interval of all possible solutions $[lower, upper]$, and we proceed bisecting such interval, each time picking a guess as $\lfloor (lower + upper)/2 \rfloor$.

The pseudo-code of this algorithm can be seen in Fig. 2. As before, the procedure receives as input $\varphi_\mathcal{C}$ and c^1. $[lower, upper]$ is initialized to $[\mathsf{lb}^1, \mathsf{ub}^1]$, $mincost$ to $+\infty$ and $guess$ to ub^1; each call IncrementalSMT$_{\mathcal{T}\cup\mathcal{C}}(\varphi_\mathcal{C}, \mathsf{BC}(c^1, bound))$ either returns sat plus the value $cost \stackrel{\text{def}}{=} \mathsf{CostOf}_1(\mu)$, or it returns unsat. In the first case, the range is restricted to $[lower, cost - 1]$ (10.), in the latter to $[guess + 1, upper]$ (13.). (Notice that, unlike with standard binary search, restricting to $[lower, cost - 1]$ rather than to $[lower, guess - 1]$ allows for exploiting the $cost$ information to further restrict the search.) Moreover, in the first case $\mathsf{BC}(c^1, bound)$ is conjoined to $\varphi_\mathcal{C}$ (8.),

$\neg\mathsf{BC}(c^1, bound)$ in the latter (`12.`), which allows for reusing the previously-learned $\mathcal{C}$-lemmas (10) end (11) respectively to prune the search.

Termination is straightforward, since $upper - lower$ is a suitable a ranking function. Correctness and completeness are similarly obvious, since the interval of possible solutions will always contain the optimal solution.

5 PseudoBoolean and MAX-SAT/SMT as SMT($\mathcal{C}$)/SMT($\mathcal{T} \cup \mathcal{C}$)

The PseudoBoolean (PB) problem can be defined as the problem:

$$\text{minimize} \sum_{j=1}^{N_1} \mathsf{c}_j^1 A_j^1 \text{ under the constraints } \{\sum_{j=1}^{N_i} \mathsf{c}_j^i A_j^i \geq \mathsf{lb}^i \mid i \in [2, ..., M]\} \quad (13)$$

where A_j^1 are Boolean atoms, A_j^i Boolean literals, and $\mathsf{c}_j, \mathsf{c}_j^i, \mathsf{lb}^i$ positive integer values. This is an extension of the SAT problem which can efficiently express many problems of practical interest.

The $SMT(\mathcal{C})$ problem is closely related to the PB problem, in fact they are equally expressive. First, we notice that $\sum_j \mathsf{c}_j^i A_j^i$ can be rewritten as $\sum_j ite(A_j^i, \mathsf{c}_j^i, 0)$, s.t. we immediately see that the PB problem (13) is a subcase of the BOMT($\mathcal{T}$) problem of §4.2 where $\mathcal{T}$ is plain Boolean logic, and as such it can be solved using the SMT($\mathcal{T} \cup \mathcal{C}$) encoding in (12) and the SMT($\mathcal{T} \cup \mathcal{C}$)-based procedures in §4.2.

For solving PB problems by translation into $SMT(\mathcal{C})$ in practice, the above translation can be improved. As an example, PB constraints of the form $\sum_j A_j^i \geq 1$ can be translated into the single propositional clause $\bigvee_j A_j^i$. In general, it may be advantageous to translate PB constraints into propositional clauses when the number of resulting clauses is low. See for instance [10] for some possibilities.

Proposition 4. *For every $SMT(\mathcal{C})$ instance, there exists a polynomial-time translation into an equivalent instance of the PB problem*

In the *Weighted Partial Max-SMT($\mathcal{T}$)* problem, in a CNF $\mathcal{T}$-formula $\phi \stackrel{\text{def}}{=} \phi_h \wedge \phi_s$ each clause C_j in ϕ_s is tagged with a positive cost value c_j, and the problem consists in finding a $\mathcal{T}$-consistent assignment μ which propositionally satisfies ϕ_h and maximizes the sum $\sum_{j\ s.t. \mu \models_p C_j} \mathsf{c}_j$ (that is, minimizes $\sum_{j\ s.t. \mu \not\models_p C_j} \mathsf{c}_j$). The problem is not "Weighted" iff $\mathsf{c}_j^i = 1$ for every j, and it is not "Partial" iff ϕ_h is the empty set of clauses; the [Weighted] [Partial] Max-SAT problem is the [Weighted] [Partial] Max-SMT($\mathcal{T}$) problem where $\mathcal{T}$ is plain Boolean logic.

A Weighted Partial Max-SMT($\mathcal{T}$) problem (and hence all its subcases described above) can be encoded into a SMT($\mathcal{T}$) cost minimization problem $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ s.t. $\varphi \stackrel{\text{def}}{=} \phi_h \wedge \bigwedge_j (C_j \vee A_j^i)$, $\underline{costs} \stackrel{\text{def}}{=} \{cost^1\} = \{\sum_j ite(A_j^i, \mathsf{c}_j^i, 0)\}$ and $\underline{\mathsf{bounds}} \stackrel{\text{def}}{=} \{\langle 0, \sum_j \mathsf{c}_j^i \rangle\}$, which can be addressed as described in §4.2.

Vice versa, a SMT($\mathcal{T}$) cost minimization problem $\langle \varphi, \underline{costs}, \underline{\mathsf{bounds}} \rangle$ s.t. $\underline{costs} \stackrel{\text{def}}{=} \{cost^1\} = \{\sum_j ite(A_j^1, \mathsf{c}_j^i, 0)\}$ and $\underline{\mathsf{bounds}} \stackrel{\text{def}}{=} \{\}$, can be encoded into a Weighted Partial Max-SMT($\mathcal{T}$) problem $\phi \stackrel{\text{def}}{=} \phi_h \wedge \phi_s$ where $\phi_h \stackrel{\text{def}}{=} \varphi$ and $\phi_s \stackrel{\text{def}}{=} \bigwedge_j (\neg A_j^1)$ s.t. each unit-clause $(\neg A_j^1)$ is tagged with the cost c_j^i.

6 Empirical evaluation

The algorithms described in the previous sections have been implemented within the MATHSAT SMT solver. In order to demonstrate the versatility and the efficiency of our approach, we have tested MATHSAT in several different scenarios: BOMT($\mathcal{T}$), Max-SMT, Max-SAT, and PseudoBoolean optimization.

6.1 Results on Max-SMT

In the first part of our experiments, we evaluate the behaviour of MATHSAT on problems requiring the use of a combination of $\mathcal{C}$ and another theory $\mathcal{T}$. For this evaluation, we have collected two kinds of benchmarks. First, we have randomly-generated some Max-SMT problems, [5] starting from standard SMT problems taken from the SMT-LIB. The second group of benchmarks comes from two real-world industrial case studies. These are the case studies that actually prompted us towards this research, because a plain encoding in SMT without costs resulted in unacceptable performance. Interestingly, although the application domains are very different, all the problems can be thought of as trying to find optimal displacement for some components in space. Unfortunately, we can not disclose any further details.

As regards the comparison with other systems, to the best of our knowledge there are two other SMT solvers that support Max-SMT, namely YICES [9] and BARCELOGIC [13], which were therefore the natural candidates for comparison. Unfortunately however, it was not possible to obtain from the authors a version of BARCELOGIC with support for optimization, so we had to exclude it from our analysis.

We have performed experiments on weigthed Max-SMT and partial weighted Max-SMT problems. For weighted Max-SMT, we have generated benchmarks by combining n independent unsatisfiable CNF formulas in the SMT-LIB (for $n = 2$ and $n = 3$) and assigning random weights to each clause. In order to obtain partial weighted Max-SMT instances, instead, we have first generated random BOMT($\mathcal{T}$) problems by assigning random costs to a subset of the atoms (both Boolean and $\mathcal{T}$-atoms) of some satisfiable formulas in the SMT-LIB, and then encoded the BOMT($\mathcal{T}$) problems into partial weighted Max-SMT ones, as descrbed in §5. The same encoding into partial weighted Max-SMT was used also to convert the BOMT($\mathcal{T}$) instances coming from the industrial case studies.

We ran MATHSAT using both binary and linear search for optimization, and compared it with YICES. All the experiments have been performed on 2.66Ghz Intel Xeon machines with 6Mb of cache, running Linux. The time limit was set to 300 seconds, and the memory limit to 2Gb.

The results for problems generated from SMT-LIB instances are reported in Table 1. For each solver, the table lists the number of instances for which the optimal solution was found (the total number of instances was 200), the number of instances for which the given solver was the only one to find the optimal solution, and the total and average execution times on the solved instances. From the results, we can see that binary search

[5] As observed in §5 BOMT($\mathcal{T}$) with a single cost function is equivalent to weighted partial Max-SMT, and therefore we only refer to Max-SMT here.

Category	Solver	Optimum	Unique	Time	Mean	Median
Weighted	MATHSAT-binary	56	6	4886.59	87.26	68.38
Max-SMT	YICES	47	3	5260.67	111.92	86.21
	MATHSAT-linear	23	0	4777.45	207.71	251.00
Weighted partial	MATHSAT-binary	206	1	1462.98	7.10	2.45
Max-SMT	MATHSAT-linear	206	1	2228.39	10.81	4.02
(BOMT($\mathcal{T}$))	YICES	195	0	3559.53	18.25	3.19

Table 1. Performance on Max-SMT and BOMT($\mathcal{T}$) problems. For each category, the solvers are sorted from "best" to "worst". Optimum is the number of instances where the optimum was found, and Unique is the number of optimal solutions found by a the given solver only. Time is the total execution time in seconds for all instances where an optimum was found. Mean and median is the mean and median of those times.

outperforms linear search for this kind of problems. This is true in particular on the first group of benchmarks, where binary search can find the optimum for more than twice as many instances as linear search. In both cases, moreover, MATHSAT outperforms also YICES, both in number of optimal solutions found and in execution time.

We also measured the overhead of performing optimization on these instances compared to solving the decision problem given the known optimal bound. For partial Max-SMT and binary search the mean was 9.5, the median 4.1 and the maximal ratio 49.6, meaning that solving the optimization problem took on average 9.5 times as long as determining that the optimal solution is indeed a solution. Similarly, for partial Max-SMT and linear search the mean of the ratio was 45.8, the median 24.3 and the maximal 222.3 showing that the overhead in linear search can be considerable. In the weighted partial Max-SMT problem the overhead was slightly lower. Using binary search the mean was 2.6, the median 3.4 and the maximal 22.1. Using linear search we get a mean of 4.4, a median of 5.9 and a maximal of 54.

Finally, we compared MATHSAT-linear, MATHSAT-binary and YICES on the two industrial case studies we had. In the first one, all three solvers could find the optimum on all the 7 instances of the set. YICES turned out to be the fastest, with a median run time of 0.5 seconds. The median time for MATHSAT-binary was of 1.54 seconds, and that of MATHSAT-linear of 64.84 seconds. In the second case study, composed of two instances, however, the outcome was the opposite: MATHSAT-linear and MATHSAT-binary could compute the optimum for both instances in approximately the same time, with the former being slightly faster (about 35 seconds for the easiest problem for both solvers, about 370 and 405 seconds respectively for the hardest). Yices, instead, could not compute the optimum for the hardest problem even with a timeout of 30 minutes (taking about 11 seconds on the easiest instead).

6.2 Results on Max-SAT

We have also performed comparisons with several Max-SAT solvers from the 2009 Max-SAT Evaluation [1]. For each of the three industrial categories containing pure

Category	Solver	Optimum	Sat	Time	Mean	Median
Max-SAT	MsUncore	83	0	2191.17	26.40	6.94
	Yices	56	0	1919.79	34.28	8.16
	SAT4J	30	50	1039.07	34.64	12.54
	MATHSAT-binary	16	71	1017.87	63.62	20.41
	Clone	15	0	2561.06	170.74	129.06
	MATHSAT-linear	5	82	466.91	93.38	72.05
Partial Max-SAT	Yices	71	0	1643.60	23.15	0.23
	SAT4J	67	31	1943.81	29.01	1.48
	MATHSAT-binary	55	43	248.00	4.51	0.07
	MATHSAT-linear	53	45	611.52	11.54	0.10
	MsUncore	46	0	353.84	7.69	0.20
	Clone	44	29	1743.54	39.63	6.59
Weighted partial Max-SAT	MATHSAT-binary	80	0	110.49	1.38	1.23
	SAT4J	80	0	271.86	3.40	3.26
	MsUncore	80	0	579.20	7.24	7.09
	MATHSAT-linear	79	1	1104.10	13.97	8.95
	Clone	0	0	0.00	N/A	N/A

Table 2. Performance on Max-SAT problems. For each category, the solvers are sorted from "best" to "worst". Optimum is the number of instances where the optimum was found, and Sat is the number of instances where some non-optimal solution was found. Time is the total execution time in seconds for all instances where either an optimum or unsat was found. Mean and median is the mean and median of those times.

Max-SAT, partial Max-SAT and partial weighted Max-SAT respectively we have chosen 100 instances randomly (in the case of partial weigted Max-SAT, we chose all 80 instances). We chose 3 solvers (MsUncore [11], SAT4J [4], and Clone [14]) participating in the 2009 competition that were readily available together with the YICES SMT solver [9], and ran each of them on all instances supported by that particular solver. We run MATHSAT using both binary and linear search. All solvers were run with a timeout of 300 seconds, and a memory limit of 2 GB. The results are summarized in table 2. We count both the number of optimal solutions found and the number of non-optimal solutions found. We report also the total execution time taken to find all optimal solutions and unsatisfiable answers as well as the mean and median of these times.

We can see that for pure Max-SAT, MATHSAT is not competitive in finding optimal solutions, although it can find many solutions. This can be attributed to the encoding; All clauses are marked with one IC predicate, and any cost theory conflict is very likely to be extremely large, and not helping prune search effectively. For partial Max-SAT most of the clauses are hard constraints, so the number of IC predicates is more moderate, and performance is noticeably better. This is also true for weighted partial Max-SAT, where MATHSAT using binary search outperforms the winner of the 2009 Max-SAT Evaluation, SAT4J.

Overall, we can notice that binary search seems to outperform linear search in the number of optimal solutions found, although both binary and linear search can find

Category	Solver	Optimum	Unsat	Sat	Time	Mean	Median
SMALLINT	SCIP	98	8	62	3078.88	29.04	3.49
	BSOLO	88	7	110	1754.31	18.46	0.43
	PBCLASP	67	7	127	869.66	11.75	0.05
	MATHSAT-linear	63	7	132	1699.69	24.28	0.21
	MATHSAT-binary	63	7	132	2119.07	30.27	0.22
	SAT4J	59	6	127	1149.96	17.69	1.34
BIGINT	MATHSAT-binary	52	13	45	2373.35	36.51	15.54
	MATHSAT-linear	48	13	49	1610.04	26.39	13.40
	SAT4J	19	18	51	759.15	20.51	3.55

Table 3. Performance on PB problems. For each category, the solvers are sorted from "best" to "worst". Optimum is the number of instances where the optimum was found, Sat is the number of instances where some non-optimal solution was found and Unsat is the number of instances that were found unsatisfiable. Time is the total execution time in seconds for all instances where either an optimum or unsat was found. Mean and median is the mean and median of those times.

some solution for the same number of instances as expected given that the first iteration in both algorithms are identical.

6.3 Results on PseudoBoolean solving

Finally, we tested the performance of MATHSAT on PseudoBoolean (PB) optimization problems. We compared MATHSAT, using both linear and binary search, with several PB solvers from the 2009 PB Evaluation [3], namely SCIP [7] (the winner in the OPT-SMALLINT category), BSOLO [12], PBCLASP [2] and SAT4J [4] (the winner in the OPT-BIGINT category). We selected a subset of the instances used in the 2009 PB Evaluation in the categories OPT-SMALLINT (optimization with small coefficients) and OPT-BIGINT (optimization with large coefficients, requiring multi-precision arithmetic), and ran all the solvers in the categories they supported.

The results are summarized in table 3. They show that, although MATHSAT is not competitive with the two best PB solvers currently available in the SMALLINT category, its performace is comparable to that of PBCLASP, which got the third place in the 2009 PB Evaluation. Moreover, MATHSAT (with both binary and linear search) outperforms the winner of the BIGINT category, solving more than twice as many problems as SAT4J within the timeout. It is worth observing that these results were obtained without using any specific heuristic for improving performance of MATHSAT.

Finally, we observe that also in this case binary search seems to be better than linear search. For PB problems it has been reported [5] that linear search is more effective than binary search. In our case, the opposite appears to be the case. A possible explanation is that, since our solver is still very basic, it does not find a very good initial solution. For linear search we often need a large number of iterations to locate the optimum, and this search appears to be short-circuited by the binary search algorithm. This happens not only on PB problems, but also on Max-SAT and Max-SMT problems.

7 Related Work

The closest work to ours is the work presented in [13], where the idea of optimization in SMT was introduced, in particular wrt the Max-SMT problem, in the setting of SMT with increasingly- strong theories. There are however several differences wrt [13]. The first one is that our approach is more general, since we allow for multiple cost functions. Consequently, we can handle more expressive problems (e.g. the rover domain) with multiple cost functions, and PB constraints. The second one is that there is no need to change the framework to deal with increments in the theory. In fact, this has also the advantage that the extension of a theory is not "permanent". Thus, differently from the approach in [13], our framework can also deal with binary search, while theirs can not (once inconsistency is reached, the framewrok does not support changes in the theory).

Optimization problems are also supported by Yices, but we could obtain no information about the algorithm being used.

8 Conclusions and Future Work

In this paper we have addressed the problem of Satisfiability Modulo the Theory of Costs. We have shown that dealing with costs in a dedicated manner allows to tackle significant SMT problems. Furthermore, the SMT($\mathcal{C}$) solver provides a very effective framework to deal with optimization problems. Our solver shows decent performance even in Boolean and PseudoBoolean optimization problems, providing an answer (albeit suboptimal) more often than other solvers. In a couple of categories, our MathSAT outperforms the highly tuned solvers winners of the most recent competitions.

In the future, we expect to experiment in several application domains that require reasoning about resources (e.g. planning, scheduling, WCET). We also plan to investigate applications to minimization in bounded model checking, for instance to provide more user-friendly counter-examples, and in error localization and debugging. From the technological point of view, we will investigate whether it is possible to borrow effective techniques from PseudoBoolean solvers, given the similarities with the theory of costs. Finally, we will address the problem of minimization in the case costs are a function of individual (rather than Boolean) variables.

References

1. Max-SAT 2009 Evaluation. `http://www.maxsat.udl.cat/09/`.
2. PBclasp. `http://potassco.sourceforge.net/labs.html`.
3. Pseudo-Boolean Competition 2009. `http://www.cril.univ-artois.fr/PB09/`.
4. SAT4J. `http://www.sat4j.org/`.
5. F. A. Aloul, A. Ramani, K. A. Sakallah, and I. L. Markov. Solution and optimization of systems of pseudo-boolean constraints. *IEEE Transactions on Computers*, 56(10), 2007.
6. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*. IOS Press, 2009.
7. T. Berthold, S. Heinz, and M. E. Pfetsch. Solving Pseudo-Boolean Problems with SCIP. Technical Report ZIB-Report 08-12, K. Zuse Zentrum für Informationdtechnik Berlin, 2009.

8. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability Modulo the Theory of Costs: Foundations and Applications. (Extended version). Technical Report DISI-10-001, 2010. `http://disi.unitn.it/~rseba/tacas10_extended.pdf`.
9. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Computer Aided Verification*, volume 4144 of *LNCS*. Springer, 2006.
10. N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2(1-4), 2006.
11. V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for Weighted Boolean Optimization. In *Proc. SAT*, LNCS. Springer, 2009.
12. V. M. Manquinho and J. Marques-Silva. Effective Lower Bounding Techniques for Pseudo-Boolean Optimization. In *Proc. DATE*. IEEE Computer Society, 2005.
13. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *SAT*, volume 4121 of *LNCS*. Springer, 2006.
14. K. Pipatsrisawat, A. Palyan, M. Chavira, A. Choi, and A. Darwiche. Solving Weighted Max-SAT Problems in a Reduced Search Space: A Performance Analysis. *JSAT*, 4, 2008.