Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions [*]

Roberto Sebastiani and Patrick Trentin

DISI, University of Trento, Italy

NOTE
This is an extended version of a paper published at TACAS 2015 [25].
Latest update: Wednesday 21st January, 2015

Abstract. In the last decade we have witnessed an impressive progress in the expressiveness and efficiency of Satisfiability Modulo Theories (SMT) solving techniques. This has brought previously-intractable problems at the reach of state-of-the-art SMT solvers, in particular in the domain of SW and HW verification. Many SMT-encodable problems of interest, however, require also the capability of finding models that are *optimal* wrt. some cost functions. In previous work, namely *Optimization Modulo Theory with Linear Rational Cost Functions* – *OMT*$(\mathcal{LRA} \cup \mathcal{T})$, we have leveraged SMT solving to handle the *minimization* of cost functions on linear arithmetic over the rationals, by means of a combination of SMT and LP minimization techniques.

In this paper we push the envelope of our OMT approach along three directions: first, we extend it to work with linear arithmetic on the mixed integer/rational domain, by means of a combination of SMT, LP and ILP minimization techniques; second, we develop a *multi-objective* version of OMT, so that to handle many cost functions simultaneously or lexicographically; third, we develop an *incremental* version of OMT, so that to exploit the incrementality of some OMT-encodable problems. An empirical evaluation performed on OMT-encoded verification problems demonstrates the usefulness and efficiency of these extensions.

1 Introduction

In many contexts including automated reasoning (AR) and formal verification (FV) important *decision* problems are effectively encoded into and solved as Satisfiability Modulo Theories (SMT) problems. In the last decade efficient SMT solvers have been developed, that combine the power of modern conflict-driven clause-learning (CDCL) SAT solvers [18] with the expressiveness of dedicated decision procedures ($\mathcal{T}$*-solvers*) for several first-order theories of practical interest like, e.g., those of linear arithmetic over the rationals ($\mathcal{LRA}$) or the integers ($\mathcal{LIA}$) or their combination ($\mathcal{LRIA}$), those of non-linear arithmetic over the reals ($\mathcal{NLRA}$) or the integers ($\mathcal{NLIA}$), of arrays ($\mathcal{AR}$), of bit-vectors ($\mathcal{BV}$), and their combinations. (See [20, 22, 3] for an overview.) This has

[*] This work is supported by Semiconductor Research Corporation (SRC) under GRC Research Project 2012-TJ-2266 WOLF. We thank Alberto Griggio for support with MATHSAT5 code.

brought previously-intractable problems at the reach of state-of-the-art SMT solvers, in particular in the domain of software (SW) and hardware (HW) verification.

Many SMT-encodable problems of interest, however, may require also the capability of finding models that are *optimal* wrt. some cost function over arithmetical variables. (See e.g. [24, 16, 23] for a rich list of such applications.) For instance, in SMT-based *model checking with timed or hybrid systems* (e.g. [2, 1]) you may want to find executions which optimize the value of some parameter (e.g., a clock timeout value, or the total elapsed time) while fulfilling/violating some property (e.g., find the minimum time interval for a rail-crossing causing a safety violation).

Surprisingly, only few works extending SMT to deal with *optimization* problems have been presented in the literature [19, 8, 24, 11, 17, 9, 23, 16, 15, 5] –most of which handle problems which are different to that addressed in this paper [19, 8, 11, 17, 9], see related work below.

Sebastiani and Tomasi [24, 23] presented two procedures for adding to SMT($\mathcal{LRA} \cup \mathcal{T}$) the functionality of finding models minimizing some $\mathcal{LRA}$ cost variable –$\mathcal{T}$ being some (possibly empty) stably-infinite theory s.t. $\mathcal{T}$ and $\mathcal{LRA}$ are signature-disjoint. This problem is referred to as *Optimization Modulo Theories with linear cost functions on the rationals*, OMT($\mathcal{LRA} \cup \mathcal{T}$). (If $\mathcal{T}$ is the empty theory, then we refer to it as OMT($\mathcal{LRA}$).) [1] These procedures combine standard SMT and LP minimization techniques: the first, called *offline*, is much simpler to implement, since it uses an incremental SMT solver as a black-box, whilst the second, called *inline*, embeds the search for optimum within the CDCL loop schema, and as such it is more sophisticate and efficient, but it requires modifying the code of the SMT solver. In [24, 23] these procedures have been implemented on top of the MATHSAT5 SMT solver [10] into a tool called OPTIMATHSAT, and an extensive empirical evaluation is presented.

Li et al. [16] extended the OMT($\mathcal{LRA}$) problem by considering *contemporarily* many cost functions for the input formula φ, namely $\{cost_1, ..., cost_k\}$, so that the problem consists in enumerating k independent models for φ, each minimizing one specific $cost_i$. [2] (Intuitively, enumerating such models is in general more efficient than solving one optimization problem at the time, because it allows for sharing the SMT search steps among different cost objectives.) In [16] they presented a novel offline algorithm for OMT($\mathcal{LRA}$), and implemented it into the tool SYMBA. Unlike with the procedures in [24, 23], the algorithm described in [16] does not use a LP minimization procedure: rather, a sequence of blackbox calls to an underlying SMT solver (Z3) allows for finding progressively-better solutions along some objective direction, either forcing discrete jumps to some bounds induced by the inequalities in the problem, or proving such objective is unbounded. SYMBA is used as backend engine of the SW model checker UFO. [3] An empirical evaluation on problems derived from SW verification shows the usefulness of this multiple-cost approach.

[1] Importantly, both MaxSMT ([19, 8, 9]) and SMT with pseudo-Boolean constraints and costs [8] are straightforwardly encoded into OMT [24, 23].

[2] More precisely, in [16] the set of objectives $k_1, k_2, ...$ must be *maximized*, but the problem can be converted into a minimization problem by setting $cost_i = -k_i$. As in [16], we remark also that this is *not* Pareto-optimality, where a single model optimizing all objectives is searched.

[3] https://bitbucket.org/arieg/ufo/

Larraz et al. [15] present incomplete SMT($\mathcal{NLIA}$) and MaxSMT($\mathcal{NLIA}$) procedures, which use an OMT($\mathcal{LIA}$) tool as an internal component. The latter procedure, called BCLT, is described neither in [15] nor in any previous publication; however, it has been kindly made available to us by their authors upon request, together with a link to the master student's thesis describing it. [4]

Finally, we have been informed by a reviewer of an invited presentation given by Bjørner and Phan two months after the submission of this paper [5], describing general algorithms for optimization in SMT, including MaxSMT, incremental, multi-objective and lexicographic OMT, Pareto-optimality, which are implemented into the tool νZ on top of Z3. Remarkably, [5] presents specialized procedures for MaxSMT, and enriches the offline OMT schema of [24, 23] with specialized algorithms for unbound-solution detection and for bound-tightening.

We are not aware of any other OMT tool currently available.

We remark a few facts about the OMT tools in [24, 23, 16, 15]. First, none of them has an *incremental* interface, allowing for pushing and popping subformulas (including definitions of novel cost functions) so that to reuse previous search from one call to the other; in a FV context this limitation is relevant, because often SMT backends are called incrementally (e.g., in the previously-mentioned example of SMT-based bounded model checking of timed&hybrid systems). Second, none of the above tools supports mixed integer/real optimization, OMT($\mathcal{LRIA}$). Third, none of the above tools supports *both* multi-objective optimization and integer optimization. Finally, neither SYMBA nor BCLT currently handle combined theories.

In this paper we push the envelope of the OMT($\mathcal{LRA} \cup \mathcal{T}$) approach of [24, 23] along three directions: (i) we extend it to work also with linear arithmetic on the mixed integer/rational domain, OMT($\mathcal{LRIA} \cup \mathcal{T}$), by means of a combination of SMT, LP and ILP minimization techniques; (ii) we develop a *multi-objective* version of OMT, so that to handle many cost functions simultaneously or lexicographically; (iii) we develop an *incremental* version of OMT, so that to exploit the incrementality of some OMT-encodable problems. We have implement these novel functionalities in OPTIMATHSAT. An empirical evaluation performed on OMT-encoded formal verification problems demonstrates the usefulness and efficiency of these extensions.

Content. The paper is organized as follows: in §2 we provide the necessary background knowledge on SMT and OMT; in §3 we introduce and discuss the above-mentioned novel extensions of OMT; in §4 we perform an empirical evaluation of such procedures.

Other Related Work The idea of optimization in SMT was first introduced by Nieuwenhuis & Oliveras [19], who presented an abstract logical framework of "SMT with progressively stronger theories" (e.g., where the theory is progressively strengthened by every new approximation of the minimum cost), and present implementations for MaxSMT based on this framework. Cimatti et al. [8] introduced the notion of "Theory of Costs" $\mathcal{C}$ to handle PB cost functions and constraints by an ad-hoc and independent "$\mathcal{C}$-solver" in the standard lazy SMT schema, and implemented a variant of

[4] `http://upcommons.upc.edu/pfc/handle/2099.1/14204?locale=en`.

MathSAT tool able to handle SMT with PB constraints and to minimize PB cost functions. Cimatti et al. [9] presented a "modular" approach for MaxSMT, combining a lazy SMT solver with a MaxSAT solver, which can be used as blackboxes. We recall that SMT with PB functions and MaxSMT can be encoded into each other, and that both are strictly less general than the OMT($\mathcal{LRA} \cup \mathcal{T}$) problems (see [24, 23]).

Two other forms of optimization in SMT, which are quite different from the one presented in our work, have been proposed in the literature. Dillig et al. [11] addressed the problem of finding *partial* models for quantified first-order formulas modulo theories, which minimize the number of free variables which are assigned a value from the domain.Quoting an example from [11], given the formula $\varphi \overset{\text{def}}{=} (x + y + w > 0) \vee (x + y + z + w < 5)$, the partial assignment $\{z = 0\}$ satisfies φ because every total assignment extending it satisfies φ and is minimum because there is no assignment satisfying φ which assigns less then one variable. They proposed a general procedure addressing the problem for every theory $\mathcal{T}$ admitting quantifier elimination, and implemented a version for $\mathcal{LIA}$ and $\mathcal{EUF}$ into the MISTRAL tool. Manolios and Papavasileiou [17] proposed the "ILP Modulo Theories" framework as an alternative to SAT Modulo Theories, which allows for combining Integer Linear Programming with decision procedures for signature-disjoint stably-infinite theories $\mathcal{T}$; they presented a general algorithm by integrating the Branch&Cut ILP method with $\mathcal{T}$-specific decision procedures, and implemented it into the INEZ tool. Notice that the approach of [17] cannot combine ILP with $\mathcal{LRA}$, since $\mathcal{LIA}$ and $\mathcal{LRA}$ are not signature-disjoint. (See Definition 2 in [17].) Also, the objective function is defined on the Integer domain. We understand that neither of the above-mentioned works can handle the problem addressed in this paper, and vice versa. (See [23] for a discussion on this topic.)

2 Background

In this section we provide the necessary background on SMT and OMT.

2.1 Satisfiability Modulo Theories

We assume a basic background knowledge on first-order logic and on CDCL SAT solving [18]. We consider some first-order theory $\mathcal{T}$, and we restrict our interest to *ground* formulas/literals/atoms in the language of $\mathcal{T}$ ($\mathcal{T}$-formulas/literals/atoms hereafter).

A *theory solver for* $\mathcal{T}$, $\mathcal{T}$*-solver*, is a procedure able to decide the $\mathcal{T}$-satisfiability of a conjunction/set μ of $\mathcal{T}$-literals. If μ is $\mathcal{T}$-unsatisfiable, then $\mathcal{T}$*-solver* returns UNSAT and a set/conjunction η of $\mathcal{T}$-literals in μ which was found $\mathcal{T}$-unsatisfiable; η is 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 $l \not\in \mu$ from a set of all available $\mathcal{T}$-literals, 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 $\mathcal{T}$-conflict and $\mathcal{T}$-deduction clauses are valid in $\mathcal{T}$. We call them $\mathcal{T}$*-lemmas*.

Given a $\mathcal{T}$-formula φ, the formula φ^p obtained by rewriting each $\mathcal{T}$-atom in φ into a fresh atomic proposition is the *Boolean abstraction* of φ, and φ is the *refinement* of φ^p. Notationally, we indicate by φ^p and μ^p the Boolean abstraction of φ and μ, and by φ

and μ the refinements of φ^p and μ^p respectively. With a little abuse of notation, we say that μ^p is $\mathcal{T}$-(un)satisfiable iff μ is $\mathcal{T}$-(un)satisfiable. We say that the truth assignment μ *propositionally satisfies* the formula φ, written $\mu \models_p \varphi$, if $\mu^p \models \varphi^p$.

In a lazy SMT($\mathcal{T}$) solver, the Boolean abstraction φ^p of the input formula φ is given as input to a CDCL SAT solver, and whenever a satisfying assignment μ^p is found s.t. $\mu^p \models \varphi^p$, the corresponding set of $\mathcal{T}$-literals μ is fed to the $\mathcal{T}$*-solver*; if μ is found $\mathcal{T}$-consistent, then φ is $\mathcal{T}$-consistent; otherwise, $\mathcal{T}$*-solver* returns a $\mathcal{T}$-conflict set η causing the inconsistency, so that the clause $\neg\eta^p$ is used to drive the backjumping and learning mechanism of the SAT solver. The process proceeds until either a $\mathcal{T}$-consistent assignment μ is found (φ is $\mathcal{T}$-satisfiable), or no more assignments are available (φ is $\mathcal{T}$-unsatisfiable).

Important optimizations are *early pruning* and $\mathcal{T}$*-propagation*. The $\mathcal{T}$*-solver* is invoked also when an assignment μ is still under construction: if it is $\mathcal{T}$-unsatisfiable, then the procedure backtracks, without exploring the (possibly many) extensions of μ; if it is $\mathcal{T}$-satisfiable, and if the $\mathcal{T}$*-solver* is able to perform a $\mathcal{T}$-deduction $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, then l can be unit-propagated, and the $\mathcal{T}$-deduction clause $(\bigvee_{i=1}^n \neg l_i \vee l)$ can be used in backjumping and learning. To this extent, in order to maximize the efficiency, most $\mathcal{T}$-solvers are *incremental* and *backtrackable*, that is, they are called via a push&pop interface, maintaining and reusing the status of the search from one call and the other.

Another optimization is *pure-literal filtering*: if some $\mathcal{LRA}$-atoms occur only positively [resp. negatively] in the original formula (learned clauses are ignored), then we can safely drop every negative [resp. positive] occurrence of them from the assignment μ to be checked by the $\mathcal{T}$*-solver* [22]. Intuitively, since such occurrences play no role in satisfying the formula, the resulting partial assignment $\mu^{p\prime}$ still satisfies φ^p. The benefits of this action are twofold: (i) it reduces the workload for the $\mathcal{T}$*-solver* by feeding to it smaller sets; (ii) it increases the chance of finding a $\mathcal{T}$-consistent satisfying assignment by removing "useless" $\mathcal{T}$-literals which may cause the $\mathcal{T}$-inconsistency of μ.

The above schema is a coarse abstraction of the procedures underlying most state-of-the-art SMT tools. The interested reader is pointed to, e.g., [20, 22, 3] for details.

2.2 Optimization Modulo Theories

We recall the basic ideas about OMT($\mathcal{LRA} \cup \mathcal{T}$) and about the inline procedure in [24, 23]. In what follows, $\mathcal{T}$ is some stably-infinite theory with equality s.t. $\mathcal{LRA}$ and $\mathcal{T}$ are signature-disjoint. ($\mathcal{T}$ can be a combination of theories.) We call an *Optimization Modulo $\mathcal{LRA} \cup \mathcal{T}$ problem, OMT($\mathcal{LRA} \cup \mathcal{T}$)*, a pair $\langle \varphi, cost \rangle$ such that φ is an SMT($\mathcal{LRA} \cup \mathcal{T}$) formula and *cost* is an $\mathcal{LRA}$ variable occurring in φ, representing the cost to be minimized. The problem consists in finding a $\mathcal{LRA}$-model $\mathcal{M}$ for φ (if any) whose value of *cost* is minimum. We call an *Optimization Modulo $\mathcal{LRA}$ problem (OMT($\mathcal{LRA}$))* an OMT($\mathcal{LRA} \cup \mathcal{T}$) problem where $\mathcal{T}$ is empty. If φ is in the form $\varphi' \wedge (cost < c)$ [resp. $\varphi' \wedge \neg(cost < c)$] for some value $c \in \mathbb{Q}$, then we call c an *upper bound* [resp. *lower bound*] for $cost$. If ub [resp. lb] is the minimum upper bound [resp. the maximum lower bound] for φ, we also call the interval $[\mathsf{lb}, \mathsf{ub}[$ the *range* of $cost$.

Remark 1. [24, 23] explain a general technique to encode an OMT($\mathcal{LRA}$) problem into OMT($\mathcal{LRA} \cup \mathcal{T}$) by exploiting the Delayed Theory Combination technique [6] implemented in MATHSAT5. It is easy to see that this holds also for $\mathcal{LIA}$ and $\mathcal{LRIA}$. Therefore, for the sake of brevity and readability, hereafter we consider the case where $\mathcal{T}$ is the empty theory (OMT($\mathcal{LRA}$), OMT($\mathcal{LIA}$) or OMT($\mathcal{LRIA}$)), referring the reader to [24, 23] for a detailed explanation about how to handle the general case.

In the inline OMT($\mathcal{LRA}$) schema, the procedure takes as input a pair $\langle \varphi, cost \rangle$, plus optionally values for lb and ub (which are implicitly considered to be $-\infty$ and $+\infty$ if not present), and returns the model $\mathcal{M}$ of minimum cost and its cost $\mathsf{u} \stackrel{\text{def}}{=} \mathcal{M}(cost)$; it returns the value ub and an empty model if φ is $\mathcal{LRA}$-inconsistent. Notice that by providing a lower bound lb [resp. an upper bound ub] the user implicitly assumes the responsibility of asserting there is no model whose cost is lower than lb [there is a model whose cost is ub]. The standard CDCL-based schema of the SMT solver is modified as follows.

Initialization. the variables l, u (defining the current range) are initialized to lb and ub respectively, the variable pivot (defining the pivot in binary search) is not initialized, the $\mathcal{LRA}$-atom PIV is initialized to $\top$ and the output model $\mathcal{M}$ is initialized to be an empty model.

Range Updating & Pivoting. Every time the search of the CDCL SAT solver gets back to decision level 0, the range $[\mathsf{l}, \mathsf{u}[$ is updated s.t. u [resp. l] is assigned the lowest [resp. highest] value u_i [resp. l_i] such that the atom $(cost < \mathsf{u}_i)$ [resp. $\neg(cost < \mathsf{l}_i)$] is currently assigned at level 0. Then the heuristic function $\mathsf{BinSearchMode}()$ is invoked, which decides whether to run the current step in binary- or in linear-search mode: in the first case (which can occur only if $\mathsf{l} > -\infty$ and $\mathsf{u} < \infty$) a value $\mathsf{pivot} \in]\mathsf{l}, \mathsf{u}[$ is computed (e.g. $\mathsf{pivot} = (\mathsf{l} + \mathsf{u})/2$), and the (possibly new) atom $\mathsf{PIV} \stackrel{\text{def}}{=} (cost < \mathsf{pivot})$ is decided to be true (level 1) by the SAT solver. This temporarily restricts the cost range to $[\mathsf{l}, \mathsf{pivot}[$. Then the CDCL solver proceeds its search, as in §2.1.

Decreasing the Upper Bound. When an assignment μ is generated s.t. $\mu^p \models \varphi^p$ and which is found $\mathcal{LRA}$-consistent by $\mathcal{LRA}$-Solver, μ is also fed to $\mathcal{LRA}$-$\mathsf{Minimize}$, returning the minimum cost min of μ; then the unit clause $C_\mu \stackrel{\text{def}}{=} (cost < \mathsf{min})$ is learned and fed to the backjumping mechanism, which forces the SAT solver to backjump to level 0, then unit-propagating $(cost < \mathsf{min})$. This restricts the cost range to $[\mathsf{l}, \mathsf{min}[$. $\mathcal{LRA}$-$\mathsf{Minimize}$ is embedded within $\mathcal{LRA}$-Solver –it is a simple extension of the LP algorithm in [12]– so that it is called incrementally after it, without restarting its search from scratch. Notice that the clauses C_μ ensure progress in the minimization every time that a new $\mathcal{LRA}$-consistent assignment is generated.

Termination. The procedure terminates when the embedded SMT-solving algorithm reveals an inconsistency, returning the current values of u and $\mathcal{M}$.

As a result of these modifications, we also have the following typical scenario (see Figure 1).

Increasing the Lower Bound. In binary-search mode, when a conflict occurs and the conflict analysis of the SAT solver produces a conflict clause in the form $\neg\mathsf{PIV} \vee \neg\eta'$ s.t. all literals in η' are assigned true at level 0 (i.e., $\varphi \wedge \mathsf{PIV}$ is $\mathcal{LRA}$-inconsistent), then the SAT solver backtracks to level 0, unit-propagating $\neg\mathsf{PIV}$. This case permanently restricts the cost range to $[\mathsf{pivot}, \mathsf{u}[$.

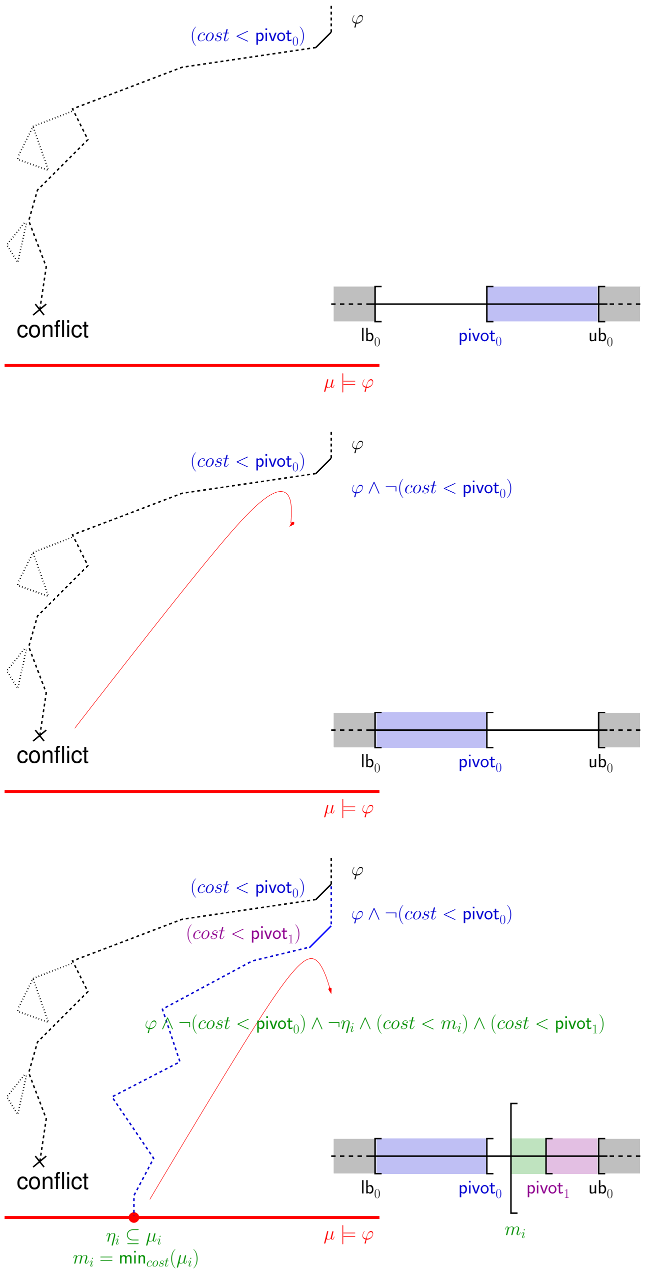

φ
$(cost < \mathsf{pivot}_0)$
conflict
lb_0
pivot_0
ub_0
$\mu \models \varphi$
φ
$(cost < \mathsf{pivot}_0)$
$\varphi \wedge \neg(cost < \mathsf{pivot}_0)$
conflict
lb_0
pivot_0
ub_0
$\mu \models \varphi$
φ
$(cost < \mathsf{pivot}_0)$
$\varphi \wedge \neg(cost < \mathsf{pivot}_0)$
$(cost < \mathsf{pivot}_1)$
$\varphi \wedge \neg(cost < \mathsf{pivot}_0) \wedge \neg\eta_i \wedge (cost < m_i) \wedge (cost < \mathsf{pivot}_1)$
conflict
lb_0
pivot_0
pivot_1
ub_0
m_i
$\eta_i \subseteq \mu_i$
$m_i = \mathsf{min}_{cost}(\mu_i)$
$\mu \models \varphi$

Fig. 1. One piece of possible execution of an inline procedure. (i) Pivoting on $(cost < \mathsf{pivot}_0)$. (ii) Increasing the lower bound to pivot_0. (iii) Decreasing the upper bound to $\mathsf{min}_{cost}(\mu_i)$.

Notice that, to guarantee termination, binary-search steps must be interleaved with linear-search ones infinitely often. We refer the reader to [24, 23] for details and for a description of further improvements to the basic inline procedure.

OMT($\mathcal{LRA}\cup\mathcal{T}$) vs. SMT with PseudoBoolean constraints & costs and MaxSMT. We recall from [23] that both SMT($\mathcal{T}$) with Pseudo-Boolean constraints (see e.g. [21]) and MaxSMT($\mathcal{T}$) are strict subcases of OMT($\mathcal{LRA} \cup \mathcal{T}$), since they can be straightforwardly encoded into it, [5] but the vice-versa does not hold.

Pseudo-Boolean (PB) constraints in the form $(\sum_i \mathbf{a}_i X^i \leq b)$ s.t. X^i are Boolean atoms and $\mathbf{a}_i$ constant values in $\mathbb{Q}$, and cost functions $cost = \sum_i \mathbf{a}_i X^i$, are encoded into OMT($\mathcal{LRA} \cup \mathcal{T}$) by rewriting each PB-term $\sum_i \mathbf{a}_i X^i$ into the $\mathcal{LRA}$-term $\sum_i \mathbf{x}_i$, $\mathbf{x}$ being an array of fresh $\mathcal{LRA}$ variables, and by conjoining to φ the formula:

$$\bigwedge_i((X^i \rightarrow (\mathbf{x}_i = \mathbf{a}_i)) \wedge (\neg X^i \rightarrow (\mathbf{x}_i = 0)) \wedge (\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)\ ). \tag{1}$$

A (partial weighted) MaxSMT($\mathcal{T}$) problem (see [19, 8, 9]) is a pair $\langle \varphi_h, \varphi_s \rangle$ where φ_h is a set of "hard" $\mathcal{T}$-clauses and φ_s is a set of weighted "soft" $\mathcal{T}$-clauses, s.t. a positive weight $\mathbf{a}_i$ is associated to each soft $\mathcal{T}$-clause $C_i \in \varphi_s$; the problem consists in finding a maximum-weight set of soft $\mathcal{T}$-clauses ψ_s s.t. $\psi_s \subseteq \varphi_s$ and $\varphi_h \cup \psi_s$ is $\mathcal{T}$-satisfiable. A MaxSMT($\mathcal{T}$) problem $\langle \varphi_h, \varphi_s \rangle$ can be encoded straightforwardly into an OMT($\mathcal{LRA} \cup \mathcal{T}$) problem $\langle \varphi, cost \rangle$, φ being the following formula:

$$\varphi_h \wedge (cost = \sum_{\mathcal{C}_j \in \varphi_s} \mathbf{x}_i) \wedge \bigwedge_{\mathcal{C}_j \in \varphi_s} \left(\begin{array}{ll} (X^j \vee \mathcal{C}_j) & \wedge \\ (X^j \rightarrow (\mathbf{x}_j = \mathbf{a}_j)) \wedge (\neg X^j \rightarrow (\mathbf{x}_j = 0)) & \wedge \\ (\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i). & \end{array} \right) \tag{2}$$

Notice that the sub-formula "$(\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)$" in both (1) and (2) is not strictly necessary, but it may improve the performances of the OMT($\mathcal{LRA}\cup\mathcal{T}$) solver, because it allows for exploiting the early-pruning technique of the underlying SMT solver by providing a range for the values of the $\mathbf{x}_i$'s before the respective X^i's are assigned. For instance, let $\mathbf{a}_1 = 3$, $\mathbf{a}_2 = 6$, $\mathbf{a}_3 = 2$, let the current upper bound for $cost$ be 5 and the current partial assignment μ assign X^2 to true, forcing $(\mathbf{x}_2 = 6)$ to true. Then no assignment extending μ can be $\mathcal{LRA}$-consistent, because $(\mathbf{x}_2 = 6)$ alone forces $cost$ to exceed the upper bound. In this situation, the constraints $(\mathbf{x}_1 \geq 0) \wedge (\mathbf{x}_1 \leq 3)$ and $(\mathbf{x}_3 \geq 0) \wedge (\mathbf{x}_3 \leq 1)$ allow an early-pruning call to the $\mathcal{LRA}$-solver to detect the $\mathcal{LRA}$-inconsistency of μ before assigning a truth value also to X^1 and X^3, because they force μ to contain also $(\mathbf{x}_1 \geq 0)$ and $(\mathbf{x}_3 \geq 0)$:

$$\mu = \{(cost = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3), (cost < 5), (\mathbf{x}_1 \geq 0), (\mathbf{x}_2 = 6), (\mathbf{x}_3 \geq 0), ...\},$$

so that the SMT solver backtracks, pruning the search space. A specular situation happens, e.g., if the current lower bound for $cost$ is 6 and μ assigns X^2 to false.

[5] Provided $\mathcal{T}$ and $\mathcal{LRA}$ are Nelson-Oppen theories, as in the OMT($\mathcal{LRA} \cup \mathcal{T}$) definition [23].

3 Pushing the envelope of OMT

3.1 From OMT($\mathcal{LRA}$) to OMT($\mathcal{LRIA}$)

We start from the observation that the only $\mathcal{LRA}$-specific components of the inline OMT($\mathcal{LRA}$) schema of §2.2 are the $\mathcal{T}$-solving and minimizing procedures. Thus, under the assumption of having an efficient $\mathcal{LRIA}$-Solver already implemented inside the embedded SMT solver –like we have in MATHSAT5 [14]– the schema in §2.2 can be adapted to $\mathcal{LRIA}$ by invoking an $\mathcal{LRIA}$-specific minimizing procedure each time a truth-assignment μ s.t. $\mu^p \models \varphi^p$ is generated.

Remark 2. Notice that in principle in $\mathcal{LIA}$ the minimization step is not strictly necessary if the input problem is lower bounded. In fact, to find the optimum *cost* value it would be sufficient to iteratively enumerate and remove each solution found by the standard implementation of the $\mathcal{LIA}$-Solver, because each step guarantees an improvement of at least 1. Minimizing the *cost* value at each iteration of the SMT engine, however, allows for speeding up the optimization search by preventing the current truth assignment μ from being generated more than once. In addition, the availability of a specialized $\mathcal{LIA}$-Minimize procedure is essential to recognize unbounded problems.

The problem of implementing an efficient OMT($\mathcal{LRIA}$) tool reduces thus to that of implementing an efficient minimizer in $\mathcal{LRIA}$, namely $\mathcal{LRIA}$-Minimize, *which exploits and cooperates in synergy with the other components of the SMT solver*. In particular, it is advisable that $\mathcal{LRIA}$-Minimize is embedded into the $\mathcal{LRIA}$-Solver, so that it is called incrementally after the latter has checked the $\mathcal{LRIA}$-consistency of the current assignment μ. (Notice that, e.g., embedding into $\mathcal{LRIA}$-Minimize a MILP tool from the shelf would not match these requirements.) To this extent, we have investigated both theoretically and empirically three different schemas of Branch&Bound $\mathcal{LRIA}$-Minimize procedure, which we call *basic*, *advanced* and *truncated*.

The first step performed by $\mathcal{LRIA}$-Minimize is to check whether *cost* is lower bounded. Since a *feasible MILP* problem is unbounded if and only if its corresponding continuous relaxation is unbounded [7],[6] we run $\mathcal{LRA}$-Minimize on the relaxation of μ. If the relaxed problem if unbounded, then $\mathcal{LIA}$-Minimize returns $-\infty$; otherwise, $\mathcal{LRA}$-Minimize returns the minimum value of *cost* in the relaxed problem, which we set as the current *lower bound* lb for *cost* in the original problem. We also initialize the *upper bound* ub for *cost* to the value $\mathcal{M}(cost)$, where $\mathcal{M}$ is the model returned by the most recent call to the $\mathcal{LRIA}$-Solver on μ.

Then we explore the solution space by means of an LP-based Branch&Bound procedure that reduces the original MILP problem to a sequence of smaller sub-problems, which are solved separately.

Basic Branch&Bound. We describe first a naive version of the Branch&Bound minimization procedure. (Since it is very inefficient, we present it only as a baseline for

[6] As in [7], by "continuous relaxation" –henceforth simply "relaxation"– we mean that the integrality constraints on the integer variables are relaxed, so that they can take fractional values.

the other approaches.) We first invoke $\mathcal{LRA}$-Minimize on the relaxation of the current $\mathcal{LRIA}$ problem. If the relaxation is found $\mathcal{LRA}$-unsatisfiable, then also the original problem is $\mathcal{LRIA}$-unsatisfiable, and the procedure backtracks. Otherwise, $\mathcal{LRA}$-Minimize returns a minimum-cost model $\mathcal{M}$ of cost min. If such solution is $\mathcal{LRIA}$-compliant, then we can return $\mathcal{M}$ and min, setting $\mathsf{ub} = \mathsf{min}$. (By "$\mathcal{LRIA}$-compliant solution" here we mean that the integer variables are all given integer values, whilst rational variables can be given fractional values.)

Otherwise, we select an integer variable x_j which is given a fractional value x_j^* in $\mathcal{M}$ as *branching variable*, and split the current problem into a pair of complementary sub-problems, by augmenting them respectively with the linear cuts $(x_j \leq \lfloor x_j^* \rfloor)$ and $(x_j \geq \lceil x_j^* \rceil)$. Then, we separately explore each of these two sub-problems in a recursive fashion, and we return the best of the two minimum values of $cost$ which is found in the two branches, with the relative model.

In order to make this exploration more efficient, as the recursive Branch&Bound search proceeds, we keep updating the upper bound ub to the current best value of $cost$ corresponding to an $\mathcal{LRIA}$-compliant solution. Then, we can prune all sub-problems in which the $\mathcal{LRA}$ optimum $cost$ value is greater or equal than ub, as they cannot contain any better solution.

Advanced Branch&Bound. Unlike the basic scheme, the advanced Branch&Bound is built on top of the $\mathcal{LRIA}$-Solver of MathSAT5 and takes advantage of all the advanced features for performance optimization that are already implemented there [14]. In particular, we re-use its very-efficient internal Branch&Bound procedure for $\mathcal{LRIA}$-solving, which exploits historical information to drive the search and achieves higher pruning by *back-jumping* within the Branch&Bound search tree, driven by the analysis of unsatisfiable cores. (We refer the reader to [14] for details.)

We adapt the $\mathcal{LRIA}$-solving algorithm of [14] to minimization as follows. As before, the minimization algorithm starts by setting $\mathsf{ub} = \mathcal{M}(cost)$, $\mathcal{M}$ being the model for μ which was returned by the most recent call to the $\mathcal{LRIA}$-Solver. Then the linear cut $(cost < \mathsf{ub})$ is pushed on top of the constraint stack of the $\mathcal{LRIA}$-Solver, which forces the search to look for a better $\mathcal{LRIA}$-compliant solution than the current one.

Then, we use the internal Branch&Bound component of the $\mathcal{LRIA}$-Solver to seek for a new $\mathcal{LRIA}$-compliant solution. The first key modification is that we invoke $\mathcal{LRA}$-Minimize on each node of Branch&Bound search tree to ensure that x_{LP}^* is optimal in the $\mathcal{LRA}$ domain. The second modification is that, every time a new solution is found –whose cost ub improves the previous upper bound by construction– we empty the stack of $\mathcal{LRIA}$-Solver, push there a new cut in the form $(cost < \mathsf{ub})$ and restart the search. Since the problem is known to be bounded, there are only a finite number of $\mathcal{LRIA}$-compliant solutions possible that can be removed from the search space. Therefore, the set of constraints is guaranteed to eventually become unsatisfiable, and at that point ub is returned as optimum $cost$ value in μ to the SMT solver, which learns the unit clause $C_\mu \overset{\text{def}}{=} (cost < \mathsf{ub})$.

Truncated Branch&Bound. We have empirically observed that in most cases the above scheme is effective enough that a single loop of advanced Branch&Bound is sufficient to find the optimal solution for the current truth assignment μ. However, the advanced Branch&Bound procedure still performs an additional loop iteration to prove that such solution is indeed optimal, which causes additional unnecessary overhead. Another drawback of advanced B&B is that for degenerate problems the Branch&Bound technique is very inefficient. In such cases, it is more convenient to interrupt the B&B search and simply return ub to the SMT solver, s.t. the unit clause $C_\mu \stackrel{\text{def}}{=} (cost < \mathsf{ub})$ is learned; in fact, in this way we can easily re-use the entire stack of $\mathcal{LRIA}$-Solver routines in MATHSAT5 to find an improved solution more efficiently.

Therefore, we have implemented a "sub-optimum" variant of $\mathcal{LRIA}$-Minimize in which the inner $\mathcal{LRIA}$-Solver minimization procedure stops as soon as either it finds its first solution or it reaches a certain limit on the number of branching steps. The drawback of this variant is that, in some cases, it analyzes a truth assignment μ (augmented with the extra constraint $(cost < \mathsf{ub})$) more than once.

3.2 Multiple-objective OMT

We generalize the OMT($\mathcal{LRIA}$) problem to multiple cost functions as follows. (As with plain OMT($\mathcal{LRIA}$), the extension to OMT($\mathcal{LRIA} \cup \mathcal{T}$) follows the technique described in [24, 23].) A *multiple-cost OMT($\mathcal{LRIA}$) problem* is a pair $\langle\varphi, \mathcal{C}\rangle$ s.t $\mathcal{C} \stackrel{\text{def}}{=} \{cost_1, ..., cost_k\}$ is a set of $\mathcal{LRIA}$-variables occurring in φ, and consists in finding a set of $\mathcal{LRIA}$-models $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$ s.t. each $\mathcal{M}_i$ makes $cost_i$ minimum. We extend the OMT($\mathcal{LRA}$) [OMT($\mathcal{LRIA}$)] procedures of §2.2 and §3.1 to handle multiple-cost problems. The procedure works in linear-search mode only.

Remark 3. Since the linear-search versions of the procedures in §2.2 and §3.1 differ only for the fact that they invoke $\mathcal{LRA}$-Minimize and $\mathcal{LRIA}$-Minimize respectively, here we do not distinguish between them. We only implicitly make the assumption that the $\mathcal{LRIA}$-Minimize does not work in truncated mode, so that it is guaranteed to find a minimum in one run. Such assumption is not strictly necessary, but it makes the explanation easier.

It takes as input a pair $\langle\varphi, \mathcal{C}\rangle$ and returns a list of minimum-cost models $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, plus the corresponding list of minimum values $\{\mathsf{u}_1, ..., \mathsf{u}_k\}$. (If φ is $\mathcal{LRIA}$-inconsistent, it returns $\mathsf{u}_i = +\infty$ for every i.)

Initialization. First, we set $\mathsf{u}_i = +\infty$ for every i, and we set $\mathcal{C}^* = \mathcal{C}$, s.t. $\mathcal{C}^*$ is the list of currently-active cost functions.

Decreasing the Upper Bound. When an assignment μ is generated s.t. $\mu^p \models \varphi^p$ and which is found $\mathcal{LRIA}$-consistent by $\mathcal{LRIA}$-Solver, μ is also fed to $\mathcal{LRIA}$-Minimize. For each $cost_i \in \mathcal{C}^*$:

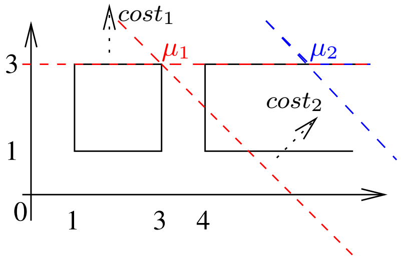
(i) $\mathcal{LRIA}$-Minimize finds an $\mathcal{LRIA}$-model $\mathcal{M}$ for μ of minimum cost min_i;

(ii) if min_i is $-\infty$, then there is no more reason to investigate $cost_i$, so that we set $\mathsf{u}_i = -\infty$ and $\mathcal{M}_i = \mathcal{M}$, and $cost_i$ is dropped from $\mathcal{C}^*$;

(iii) if $\mathsf{min}_i < \mathsf{u}_i$, then we set $\mathsf{u}_i = \mathsf{min}_i$ and $\mathcal{M}_i = \mathcal{M}$.

As with the single-cost versions of §2.2, $\mathcal{LRIA}$-Minimize is embedded within $\mathcal{LRIA}$-Solver, so that it is called incrementally after it, without restarting its search from scratch. After that, the clause

$$C_\mu \stackrel{\text{def}}{=} \bigvee_{cost_i \in \mathcal{C}^*} (cost_i < \mathsf{u}_i) \tag{3}$$



$$
\begin{aligned}
\varphi \overset{\text{def}}{=} & (1 \le y) \wedge (y \le 3) \\
& \wedge\ (((1 \le x) \wedge (x \le 3)) \vee (x \ge 4)) \\
& \wedge\ (cost_1 = -y) \wedge (cost_2 = -x - y)
\end{aligned}
$$

$$
\begin{aligned}
\mu_1 \overset{\text{def}}{=} & \{(1 \le y), (y \le 3), (1 \le x), (x \le 3), \\
& (cost_1 = -y), (cost_2 = -x - y)\} \\
\mu_2 \overset{\text{def}}{=} & \{(cost_1 = -y), (cost_2 = -x - y), \\
& (1 \le y), (y \le 3), (x \ge 4), \\
& (cost_2 < -6)\}
\end{aligned}
$$

Fig. 2. In one possible execution over the $\mathcal{LRA}$-formula φ, the CDCL-based SMT engine finds the truth assignment μ_1 first, which is found $\mathcal{LRA}$-consistent by the $\mathcal{LRA}$-solver. (For the sake of readability, we've removed from the μ_i's the redundant literals like "$\neg(x \ge 4)$" from μ_1.) Then the minimizer finds the minima $\mathsf{min}_1 = -3$, $\mathsf{min}_2 = -6$, the upper bounds are updated to these values, and the clause $(cost_1 < -3) \vee (cost_2 < -6)$ is learned. The next $\mathcal{LRA}$-consistent assignment found is necessarily μ_2, from which the minimizer finds the minima $\mathsf{min}_1 = -3$, $\mathsf{min}_2 = -\infty$. Hence $cost_2$ is dropped from $\mathcal{C}^*$, and the unit clause $(cost_1 < -3)$ is learned, making φ $\mathcal{LRA}$-inconsistent, so that no more assignment is found and the procedure terminates. In a luckier execution $\mu_2 \setminus \{(cost_2 < -6)\}$ is found first, thus the minimizer finds directly the minima $\mathsf{min}_1 = -3$, $\mathsf{min}_2 = -\infty$ s.t. $(cost_1 < -3)$ is learned, and the procedure terminates without generating μ_1.

is learned, and the CDCL-based SMT solving process proceeds its search. Notice that, since by construction $\mu \wedge C_\mu \models_{\mathcal{LRIA}} \bot$, a theory-driven backjumping step [3] will occur as soon as μ is extended to assign to true some literal of C_μ.

Termination. The procedure terminates either when $\mathcal{C}^*$ is empty or when φ is found $\mathcal{LRIA}$-inconsistent. (The former case is a subcase of the latter, because it would cause the generation of an empty clause C_μ (3).)

The clauses C_μ (3) ensure a progress in the minimization of one or more of the $cost_i$'s every time that a new $\mathcal{LRIA}$-consistent assignment is generated. We notice that, by construction, C_μ is such that $\mu \wedge C_\mu \models_{\mathcal{LRIA}} \bot$, so that each μ satisfying the original version of φ can be investigated by the minimizer only once. Since we have only a finite number of such candidate assignments for φ, this guarantees the termination of the procedure. The correctness and completeness is guaranteed by these of $\mathcal{LRIA}$-$\mathsf{Minimize}$, which returns the minimum values for each such assignment.

To illustrate the behaviour of our procedure, and to allow for a direct comparison wrt. the procedure described in [16], in Figure 2 we present its execution on the toy example $\mathcal{LRA}$-problem in [16]. Notice that, unlike the algorithm in [16], our procedure is driven by the Boolean search: each time a novel assignment is generated, it eagerly produces the maximum progress for as many $cost_i$'s as possible. The algorithm described in [16], instead, does not use a LP minimization procedure: rather, a sequence of blackbox calls to an underlying SMT solver (Z3) allows for finding progressively-better solutions along some objective direction, either forcing discrete jumps to some bounds induced by the inequalities in the problem, or proving such objective is unbounded.

The procedure is improved in various ways. First, we notice that the clause C_μ is strictly stronger than the clause $C_{\mu'}$ which was generated with the previous truth assignment μ', so that $C_{\mu'}$ can be safely dropped, keeping only one of such clauses at a time. This is as if we had only one such clause whose literals are progressively strengthened. Second, before step (i), the constraint $(cost_i < \mathsf{u}_i)$ can be temporarily pushed into μ: if $\mathcal{LRIA}$-Minimize returns UNSAT, then there is no chance to improve the current value of u_i, so that the above constraint can be popped from μ and step (ii) and (iii) can be skipped for the current $cost_i$. Third, in case the condition in step (iii) holds, it is possible to learn also the $\mathcal{LRIA}$-valid clause $(cost_i < \mathsf{u}_i) \rightarrow (cost_i < \mathsf{u}'_i)$ s.t. u'_i is the previous value of u_i. This allows for "activating" all previously-learned clauses in the form $\neg(cost_i < \mathsf{u}'_i) \vee C$ as soon as $(cost_i < \mathsf{u}_i)$ is assigned to true.

Lexicographic combination. As in [5], we easily extend our inline procedure to deal with the lexicographic combination of multiple costs $\{cost_1, ..., cost_k\}$. This works as follows. We start by looking for a minimum for $cost_1$: as soon as a minimum u_1 with its model $\mathcal{M}_1$ is found, if $\mathsf{u}_1 = -\infty$ then we stop, otherwise we substitute inside φ the unit clause $(cost_1 < \mathsf{u}_1)$ with $(cost_1 = \mathsf{u}_1)$, we set $\mathsf{u}_2 \overset{\text{def}}{=} \mathcal{M}_1(cost_2)$, and we look for the minimum of $cost_2$ in the resulting formula. This is repeated until all $cost_i$'s have been considered.

Min-max combination. We notice that it is straighforward to extend our inline procedure for OMT($\mathcal{LRIA} \cup \mathcal{T}$) to deal with the min-max optimization of multiple costs $\{cost_1, ..., cost_k\}$, that is, to find a solution which minimizes the maximum value among the $cost_i$'s. It suffices to introduce a fresh cost variable $cost$ and then to solve the OMT($\mathcal{LRIA} \cup \mathcal{T}$) problem $\langle \varphi \wedge \bigwedge_{i=1}^{k} (cost_i \leq cost), cost \rangle$. The encoding for max-min optimization is dual.

3.3 Incremental OMT

Many modern SMT solvers, including MATHSAT5, provide a *stack-based incremental interface* (see e.g. [13]), by which it is possible to push/pop sub-formulas ϕ_i into a stack of formulas $\Phi \overset{\text{def}}{=} \{\phi_1, ..., \phi_k\}$, and then to check incrementally the satisfiability of $\bigwedge_{i=1}^{k} \phi_i$. The interface maintains the *status* of the search from one call to the other, in particular it records the *learned clauses* (plus other information). Consequently, when invoked on Φ, the solver can reuse a clause C which was learned during a previous call on some Φ' if C was derived only from clauses which are still in Φ. [7] In particular, if $\Phi' \subseteq \Phi$, then the solver can reuse all clauses learned while solving Φ'.

In particular, in MATHSAT5 incrementality is achieved by first rewriting Φ into $\{A_1 \rightarrow \phi_1, ..., A_k \rightarrow \phi_k\}$, each A_i being a fresh Boolean variable, and then by running the SMT solver under the assumption of the variables $\{A_1, ..., A_k\}$, in such a way that every learned clause which is derived from some ϕ_i is in the form $\neg A_i \vee C$ [13]. Thus it is possible to safely keep the learned clause from one call to the other because, if ϕ_i is popped from Φ, then A_i is no more assumed, so that the clause $\neg A_i \vee C$ is inactive. (Such clauses can be garbage-collected from time to time to reduce the overhead.)

[7] Provided C was not discharged in the meantime.

Since none of the OMT tools in [24, 23, 16, 15] provides an incremental interface, nor such paper explains how to achieve it, here we address explicitly the problem of making OMT incremental.

We start noticing that if (i) the OMT tool is based on the schema in §2.1 or on its $\mathcal{LRIA}$ and multiple-cost extensions of §3.1 and §3.2, and (ii) the embedded SMT solver has an incremental interface, like that of MATHSAT5, then an OMT tool can be easily made incremental by exploiting the incremental interface of its SMT solver.

In fact, in our OMT schema all learned clauses are either $\mathcal{T}$-lemmas or they are derived from $\mathcal{T}$-lemmas and some of the subformulas ϕ_i's, *with the exception of the clauses* $C_\mu \overset{\mathit{def}}{=} (cost < \mathsf{min})$ *(§2.2) [resp.* $C_\mu \overset{\mathit{def}}{=} (cost < \mathsf{min})$ *(§3.1) and* $C_\mu \overset{\mathit{def}}{=} \bigvee_{cost_i \in \mathcal{C}^*}(cost_i < \mathsf{u}_i)$ *(§3.2),]* which are “artificially” introduced to ensure progress in the minimization steps. (This holds also for the unit clauses (PIV) which are learned in an improved version, see [24, 23].) Thus, in order to handle incrementality, it suffices to drop only these clauses from one OMT call to the other, while preserving all the others, as with incremental SMT.

In a more elegant variant of this technique, which we have used in our implementation, at each incremental call to OMT (namely the k-th call) a fresh Boolean variable $A^{(k)}$ is assumed. Whenever a new minimum min is found, the augmented clause $C^*_\mu \overset{\text{def}}{=} \neg A^{(k)} \vee (cost < \mathsf{min})$ is learned instead of $C_\mu \overset{\text{def}}{=} (cost < \mathsf{min})$. In the subsequent calls to OMT, $A^{(k)}$ is no more assumed, so that the augmented clauses C^*_μ's which have been learned during the k-th call are no more active.

Notice that in this process reusing the clauses that are learned by the underlying SMT-solving steps is not the only benefit. In fact also the learned clauses in the form $\neg(cost < \mathsf{min}) \vee C$ which may be produced after learning $C_\mu \overset{\text{def}}{=} (cost < \mathsf{min})$ are preserved to the next OMT calls. (Same discourse holds for the C_μ's of §3.1 and §3.2.) In the subsequent calls such clauses are initially inactive, but they can be activated as soon as the current minimum, namely min', becomes smaller or equal than min and the novel clause $(cost < \mathsf{min}')$ is learned, so that $(cost < \mathsf{min})$ can be $\mathcal{T}$-propagated or $(\neg(cost < \mathsf{min}') \vee (cost < \mathsf{min}))$ can be $\mathcal{T}$-learned. This allows for reusing lots of previous search.

4 Experimental Evaluation

We have extended OPTIMATHSAT [24, 23] by implementing the advanced and truncated B&B OMT($\mathcal{LRIA} \cup \mathcal{T}$) procedures described in §3.1. On top of that, we have implemented our techniques for multi-objective OMT (§3.2) —including the lexicographic combination— and incremental OMT (§3.3). Then, we have investigated empirically the efficiency of our new procedures by conducing two different experimental evaluations, respectively on OMT($\mathcal{LRIA}$) (§4.1) and on multi-objective and incremental OMT($\mathcal{LRA}$) (§4.2). All tests in this section were executed on two identical *8-core 2.20Ghz Xeon* machines with 64 GB of RAM and running Linux with 3.8-0-29 kernel, with an enforced timeout of 1200 seconds.

For every problem in this evaluation, the correctness of the minimum costs found by OPTIMATHSAT and its competitor tools, namely “min”, have been cross-checked with the SMT solver Z3, by checking both the inconsistency of $\varphi \wedge (cost < \mathsf{min})$ and the consistency of $\varphi \wedge (cost = \mathsf{min})$. In all tests, when terminating, all tools returned the correct results. To make the experiments reproducible, the full-size plots, a Linux binary of OPTIMATHSAT, the input OMT problems, and the results are available. [8]

4.1 Evaluation of OMT($\mathcal{LRIA}$) procedures

Here we consider three different configurations of OPTIMATHSAT based on the search schemas (linear vs. binary vs. adaptive, denoted respectively by "-LIN", "-BIN" and "-ADA") presented in §2.2; the adaptive strategy dynamically switches the search schemas between linear and binary search, based on the heuristic described in [23]. We run OPTIMATHSAT both with the advanced and truncated branch&bound minimization procedures for $\mathcal{LRIA}$ presented in §3.1, denoted respectively by "-ADV" and "-TRN".

In order to have a comparison of OPTIMATHSAT with both νZ and BCLT, in this experimental evaluation we restricted our focus on OMT($\mathcal{LIA}$) only. Here we do not consider SYMBA, since it does not support OMT($\mathcal{LIA}$). We used as benchmarks a set of 544 problems derived from SMT-based Bounded Model Checking and K-Induction on parametric problems, generated via the SAL model checker. [9]

The results of this evaluation are shown in Figure 3. By looking at the table, we observe that the best OPTIMATHSAT configuration on these benchmarks is -TRN-ADA, which uses the truncated branch&bound approach within the $\mathcal{LIA}$-$\mathsf{Minimize}$ procedure with adaptive search scheme. We notice that the differences in performances among the various configurations of OPTIMATHSAT are small on these specific benchmarks.

Comparing the OPTIMATHSAT versions against BCLT and νZ, we notice that OPTIMATHSATand νZ solve all input formulas regardless of their configuration, νZ having better time performances, whilst BCLT timeouts on 44 problems.

4.2 Evaluation of Incremental and Multiple-objective OMT

As mentioned in Section §1, so far BCLT does not feature multi-objective OMT, and neither SYMBA nor BCLT implement incremental OMT. Thus, in order to test the efficiency of our multiple-objective OMT approach, we compared three versions of OPTIMATHSAT against the corresponding versions of νZ and the two best-performing versions of SYMBA presented in [16], namely SYMBA(100) and SYMBA(40)+OPT-Z3.

So far SYMBA handles only OMT($\mathcal{LRA}$), without combinations with other theories. Moreover, it currently does not support strict inequalities inside the input formulas. Therefore for both comparisons we used as benchmarks the multiple-objective problems which were proposed in [16] to evaluate SYMBA, which were generated from a set of C programs used in the 2013 SW Verification Competition. [10] Also, SYMBA computes both the minimum and the maximum value for each *cost* variable, and there is no

[8] `http://disi.unitn.it/~trentin/resources/tacas15.tar.gz`; BCLT is available at `http://www.lsi.upc.edu/~oliveras/bclt.gz`; SYMBA is available at `https://bitbucket.org/arieg/symba/src`; νZ is available at `http://rise4fun.com/z3opt`.

[9] `http://sal.csl.sri.com/`.

[10] `https://bitbucket.org/liyi0630/symba-bench`.

Tool:	#inst.	#solved	#timeout	time
BCLT	544	500	44	93040
νZ	544	544	0	36089
OptiM.-adv-lin	544	544	0	91032
OptiM.-adv-bin	544	544	0	99214
OptiM.-adv-ada	544	544	0	88750
OptiM.-trn-lin	544	544	0	91735
OptiM.-trn-bin	544	544	0	99556
OptiM.-trn-ada	544	544	0	88730

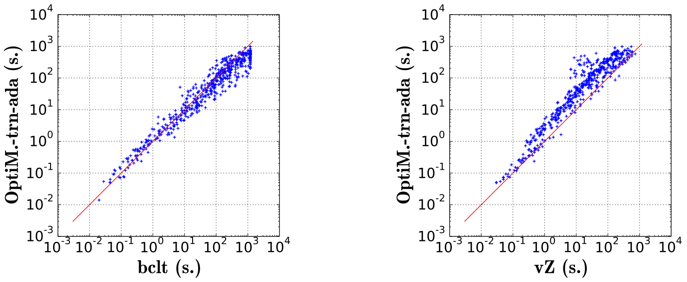


Fig. 3. A table comparing the performances of BCLT, νZ and different configurations of OptiMathSAT on Bounded Model Checking problems. Scatterplots: pairwise comparisons between OptiMathSAT-trn-ada and BCLT (left) and νZ (right).

way of restricting its focus only on one direction. Consequently, in our tests we have forced also OptiMathSATand νZ to both minimize and maximize each objective. (More specifically, they had to minimize both $cost_i$ and $-cost_i$, for each $cost_i$.)

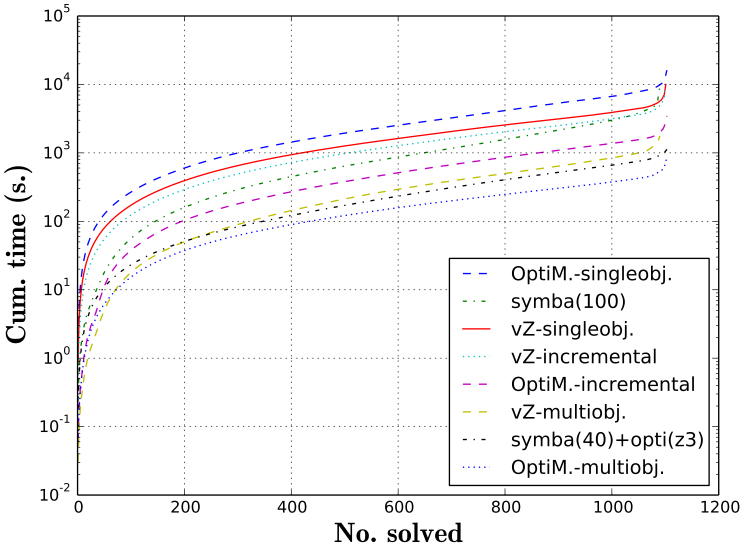
We tested three different configurations of νZ and OptiMathSAT:

- SINGLEOBJECTIVE: each tool is run singularly on the single-objective problems $\langle\varphi, cost_i\rangle$ and $\langle\varphi, -cost_i\rangle$ for each $cost_i$, and the cumulative time is taken;
- INCREMENTAL: as above, using the incremental version of each tool, each time popping the definition of the previous $cost$ and pushing the new one;
- MULTIOBJECTIVE: each tool is run in multi-objective mode with $\bigcup_i \{cost_i, -cost_i\}$.

Figure 4 provides the cumulative plots and the global data of the performance of all procedures under test, whilst Figure 5 reports pairwise comparisons.

We first compare the different versions of OptiMathSAT (see Figure 4 and the first row of Figure 5). By looking at Figure 4 and at the top-left plot in Figure 5, we observe a uniform and relevant speedup when passing from non-incremental to incremental OMT. This is explained by the possibility of reusing learned clauses from one call to the other, saving thus lots of search, as explained in §3.3.

By looking at Figure 4 and at the top-center plot in Figure 5, we observe a uniform and drastic speedup in performance –about one order of magnitude– when passing from single-objective to multiple-objective OMT. We also notice (top-right plot in Figure 5) that this performance is significantly better than that obtained with incremental OMT. Analogous considerations hold for νZ.



Tool:	#inst.	#solved	#timeout	time
Symba(100)	1103	1091	12	10917
Symba(40)+opt-z3	1103	1103	0	1128
νZ-multiobjective	1103	1090	13	1761
νZ-incremental	1103	1100	3	8683
νZ-singleobjective	1103	1101	2	10002
optimathsat-multiobjective	1103	1103	0	901
optimathsat-incremental	1103	1103	0	3477
optimathsat-singleobjective	1103	1103	0	16161

Fig. 4. Comparison of different versions of OptiMathSAT and Symba on the SW verification problems in [16]. (Notice the logarithmic scale of the vertical axis in the cumulative plots.)

We see two main motivations for this improvement in performance with our multiple-objective OMT technique: first, every time a novel truth assignment is generated, the value of many cost functions can be updated, sharing thus lots of Boolean and $\mathcal{LRA}$ search; second, the process of certifying that there is no better solution, which typically requires a significant part of the overall OMT search [23], here is executed only once.

In the second row of Figure 5 we compare the performances of OptiMathSAT-multi-objective against the two versions of Symba and νZ-multi-objective. We observe that multi-objective OptiMathSAT performs much better than the default configuration of Symba, and significantly better than both Symba(40)+opt-z3 and νZ-multi-objective.

We have also wondered how much the relative performances of OptiMathSAT, Symba and νZ depend on the relative efficiency of their underlying SMT solvers: MathSAT5 for OptiMathSAT and Z3 for Symba and νZ. Thus we have run both MathSAT5 and Z3 on the set of problems $\varphi \wedge (cost < \text{min})$ derived from the original benchmarks, and used their timings to divide the respective OptiMathSAT and

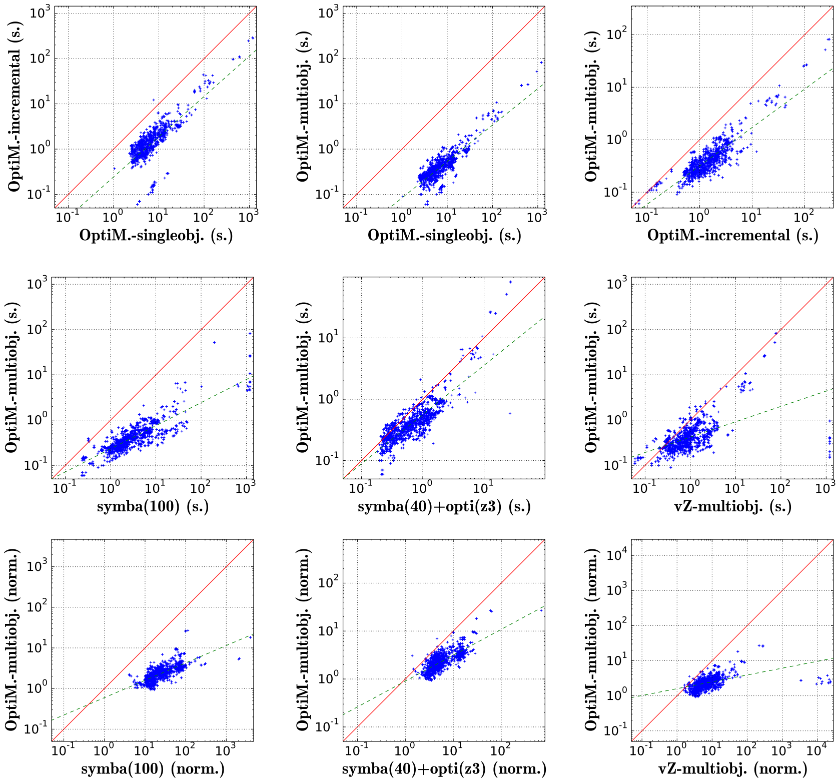


Fig. 5. First row: pairwise comparisons between different versions of OPTIMATHSAT. Second row: pairwise comparisons between OPTIMATHSAT-MULTIOBJECTIVE, the two versions of SYMBA and νZ-MULTIOBJECTIVE. Third row: "normalized" version of the plots in the second row.

SYMBA/νZ execution time values.[11] These "normalized" results, which are shown in the bottom row of Figure 5, seem to suggest that the better performances of OPTIMATHSAT are not due to better performances of the underlying SMT solver.

[11] That is, each value represents the time taken by each OMT tool on $\langle \varphi, cost_i \rangle$ divided by the time taken by its underlying SMT solver to solve $\varphi \wedge (cost < \mathsf{min})$.

References

1. G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proc. BMC 2004*, volume 119 of *ENTCS*. Elsevier, 2005.
2. G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, 2002.
3. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. Volume 185 of Biere et al. [4], February 2009.
4. A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185. IOS Press, February 2009.
5. N. Bjorner and A.-D. Phan. νZ - Maximal Satisfaction with Z3. In *Proc SCSS. Invited presentation.*, Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC). `http://www.easychair.org/publications/?page=862275542`.
6. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10):1493–1525, 2006.
7. R. H. Byrd, A. J. Goldman, and M. Heller. Technical Note– Recognizing Unbounded Integer Programs. *Operations Research*, 35(1), 1987.
8. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.
9. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A Modular Approach to MaxSAT Modulo Theories. In *SAT*, volume 7962 of *LNCS*, July 2013.
10. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT 5 SMT Solver. In *TACAS*, volume 7795 of *LNCS*. Springer, 2013.
11. I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken. Minimum Satisfying Assignments for SMT. In *CAV*, pages 394–409, 2012.
12. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*, 2006.
13. N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
14. A. Griggio. A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT*, 8:1–27, 2012.
15. D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *SAT*, 2014.
16. Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *POPL*. ACM Press., 2014.
17. P. Manolios and V. Papavasileiou. Ilp modulo theories. In *CAV*, pages 662–677, 2013.
18. J. P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. Volume 185 of Biere et al. [4], February 2009.
19. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *SAT*, volume 4121 of *LNCS*. Springer, 2006.
20. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.
21. O. Roussel and V. Manquinho. *Pseudo-Boolean and Cardinality Constraints*, chapter 22, pages 695–733. Volume 185 of Biere et al. [4], February 2009.
22. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3(3-4):141–224, 2007.

23. R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. To appear on ACM Transactions on Computational Logics, TOCL. Available at `http://optimathsat.disi.unitn.it/pages/publications.html`.
24. R. Sebastiani and S. Tomasi. Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012.
25. R. Sebastiani and P. Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In *Proc. TACAS*, LNCS. Springer, 2015.