Optimization Modulo Theories with Linear Rational Costs

ROBERTO SEBASTIANI and SILVIA TOMASI, DISI, University of Trento, Italy

In the contexts of automated reasoning (AR) and formal verification (FV), important *decision* problems are effectively encoded into Satisfiability Modulo Theories (SMT). In the last decade, efficient SMT solvers have been developed for several theories of practical interest (e.g., linear arithmetic, arrays, and bit vectors). Surprisingly, little work has been done to extend SMT to deal with *optimization* problems; in particular, we are not aware of any previous work on SMT solvers able to produce solutions that minimize cost functions over *arithmetical* variables. This is unfortunate, since some problems of interest require this functionality.

In the work described in this article we start filling this gap. We present and discuss two general procedures for leveraging SMT to handle the minimization of linear rational cost functions, combining SMT with standard minimization techniques. We have implemented the procedures within the MathSAT SMT solver. Due to the absence of competitors in the AR, FV, and SMT domains, we have experimentally evaluated our implementation against state-of-the-art tools for the domain of *Linear Generalized Disjunctive Programming (LGDP)*, which is closest in spirit to our domain, on sets of problems that have been previously proposed as benchmarks for the latter tools. The results show that our tool is very competitive with, and often outperforms, these tools on these problems, clearly demonstrating the potential of the approach.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Mechanical theorem proving*

General Terms: Theory, Algorithms, Performance

Additional Key Words and Phrases: Satisfiability modulo theories, automated reasoning, optimization

ACM Reference Format:
Roberto Sebastiani and Silvia Tomasi. 2015. Optimization modulo theories with linear rational costs. ACM Trans. Comput. Logic 16, 2, Article 12 (February 2015), 43 pages.
DOI: http://dx.doi.org/10.1145/2699915

1. INTRODUCTION

In the contexts of 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 with dedicated decision procedures ($\mathcal{T}$-Solvers) for several first-order theories of practical interest like, for example, those of equality with uninterpreted functions ($\mathcal{EUF}$), of linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$) or the integers ($\mathcal{LA}(\mathbb{Z})$), of arrays ($\mathcal{AR}$), of bit vectors ($\mathcal{BV}$), and their combinations. We refer the reader to Sebastiani [2007] and Barrett et al. [2009] for an overview.

Roberto Sebastiani is supported by Semiconductor Research Corporation (SRC) under GRC Research Project 2012-TJ-2266 WOLF.
Authors' addresses: R. Sebastiani and S. Tomasi, DISI, Università di Trento, via Sommarive 9, I-38123 Povo, Trento, Italy; emails: {roberto.sebastiani, silvia.tomasi}@unitn.it.

DOI: http://dx.doi.org/10.1145/2699915

Many SMT-encodable problems of interest, however, may also require the capability of finding models that are *optimal* with respect to some cost function over continuous arithmetical variables. For example, in (SMT-based) *planning with resources* [Wolfman and Weld 1999], a plan for achieving a certain goal must be found that not only fulfills some resource constraints (e.g., on time, gasoline consumption, among others) but that also minimizes the usage of some of such resources; in SMT-based *model checking with timed or hybrid systems* (e.g., Audemard et al. [2002, 2005]) you may want to find executions that minimize some parameter (e.g., elapsed time), or that minimize/maximize the value of some constant parameter (e.g., a clock timeout value), while fulfilling/violating some property (e.g., minimize the closure time interval of a rail crossing while preserving safety). This also involves, as particular subcases, problems that are traditionally addressed as *linear disjunctive programming (LDP)* [Balas 1998] or *linear generalized disjunctive programming (LGDP)* [Raman and Grossmann 1994; Sawaya and Grossmann 2012], or as SAT/SMT with Pseudo-Boolean (PB) constraints and (weighted partial) MaxSAT/SMT problems [Roussel and Manquinho 2009; Li and Manyà 2009; Nieuwenhuis and Oliveras 2006; Cimatti et al. 2010, 2013a]. Notice that the two latter problems can be encoded into each other.

Surprisingly, little work has been done so far to extend SMT to deal with *optimization* problems [Nieuwenhuis and Oliveras 2006; Cimatti et al. 2010; Sebastiani and Tomasi 2012; Dillig et al. 2012; Cimatti et al. 2013a; Manolios and Papavasileiou 2013] (see Section 6). In particular, to the best of our knowledge, most such works aim at minimizing cost functions over *Boolean* variables (i.e., SMT with PB cost functions or MaxSMT), while we are not aware of any previous work on SMT solvers able to produce solutions that minimize cost functions over *arithmetical* variables. Notice that the former can be encoded into the latter, but not vice versa.

In this work, we start filling this gap. We present two general procedures for adding to SMT the functionality of finding models that minimize some $\mathcal{LA}(\mathbb{Q})$ cost variable—$\mathcal{T}$ being some possibly empty, stably infinite theory such that $\mathcal{T}$ and $\mathcal{LA}(\mathbb{Q})$ are signature disjoint. These two procedures combine standard SMT and minimization techniques: the first, called *offline*, is much simpler to implement, since it uses an incremental SMT solver as a black box, while the second, called *inline*, is more sophisticated and efficient, but it requires modifying the code of the SMT solver. This distinction is important, since the source code of many SMT solvers is not publicly available.

We have implemented these procedures within the MathSAT5 SMT solver [Cimatti et al. 2013b]. Due to the absence of competitors from AR, FV, and SMT domains (Section 6), we have experimentally evaluated our implementation against state-of-the-art tools for the domain of LGDP, which is closest in spirit to our domain, on sets of problems that have been previously proposed as benchmarks for the latter tools, and on other problem sets. (Notice that LGDP is limited to plain $\mathcal{LA}(\mathbb{Q})$, so that, e.g., it cannot handle combinations of theories like $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$.) The results show that our tool is very competitive with, and often outperforms, these tools on these problems, clearly demonstrating the potential of the approach.

This article extends a paper presented at the IJCAR 2012 conference [Sebastiani and Tomasi 2012]. Here the content is extended in many ways: First, we provide the theoretical foundations of the procedures, including formal definitions, theorems, and relative proofs; second, we provide a much more detailed description and analysis of the procedures, describing in detail issues that were only hinted at in the conference paper; third, we introduce novel improvements to the procedures; fourth, we provide a much more extended empirical evaluation; and finally, we provide a detailed description of the background and of the related work.

Content. The rest of the article is organized as follows: In Section 2, we provide some background knowledge about SAT, SMT, and LGDP; in Section 3, we formally define the

problem addressed, provide the necessary formal results for its solution, and show how the problem generalizes many known optimization problems; in Section 4, we present our novel procedures; in Section 5, we present an extensive experimental evaluation; in Section 6, we survey the related work; and in Section 7, we briefly conclude and highlight directions for future work. In Appendix A, we provide the proofs of all the theorems presented in the article.

2. BACKGROUND

In this section, we provide the necessary background about SAT (Section 2.1), SMT (Section 2.2), and LGDP (Section 2.3). We assume a basic background knowledge about logic and operational research. We provide a uniform notation for SAT and SMT: We use boldface lowercase letters $\mathbf{a}, \mathbf{y}$ for arrays and boldface uppercase letters $\mathbf{A}, \mathbf{Y}$ for matrices (i.e., two-dimensional arrays), standard lowercase letters a, y for single rational variables/constants or indices and standard uppercase letters A, Y for Boolean atoms and index sets; we use the first five letters in the various forms $\mathbf{a}, \ldots \mathbf{e}, \ldots A, \ldots E$, to denote *constant* values, the last five $\mathbf{v}, \ldots \mathbf{z}, \ldots V, \ldots Z$ to denote *variables*, and the letters i, j, k, I, J, K for indexes and index sets, respectively; subscripts $._j$ denote the jth element of an array or matrix, while superscripts $.^{ij}$ are just indexes, being part of the name of the element. We use lowercase Greek letters $\varphi, \phi, \psi, \mu, \eta$ for denoting formulas and uppercase ones Φ, Ψ for denoting sets of formulas.

Remark 2.1. Although we refer to quantifier-free formulas, as is standard practice in SAT, SMT, CSP, and OR communities, with a little abuse of terminology we call "Boolean variables" the propositional atoms and we call "variables" the free constants x_i in quantifier-free $\mathcal{LA}(\mathbb{Q})$ atoms like "$(3x_1 - 2x_2 + x_3 \leq 3)$."

We assume the standard syntactic and semantic notions of propositional logic. Given a nonempty set of primitive propositions $\mathcal{P} = \{p_1, p_2, \ldots\}$, the language of propositional logic is the least set of formulas containing $\mathcal{P}$ and the primitive constants $\top$ and $\bot$ ("true" and "false"), and closed under the set of standard propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. We call a *propositional atom* every primitive proposition in $\mathcal{P}$, and a *propositional literal* every propositional atom (*positive literal*) or its negation (*negative literal*). We implicitly remove double negations: For example, if l is the negative literal $\neg p_i$, then by $\neg l$ we mean p_i rather than $\neg\neg p_i$. With a little abuse of notation, we represent a truth assignment μ indifferently either as a *set* of literals $\{l_i\}_i$, with the intended meaning that a positive (negative, respectively) literal p_i means that p_i is assigned to true (false, respectively), or as a *conjunction* of literals $\bigwedge_i l_i$; thus, for example, we may say "$l_i \in \mu$" or "$\mu_1 \subseteq \mu_2$," but also "$\neg\mu$" meaning the clause "$\bigvee_i \neg l_i$."

A propositional formula is in *Conjunctive Normal Form (CNF)* if it is written as a conjunction of disjunctions of literals: $\bigwedge_i \bigvee_j l_{ij}$. Each disjunction of literals $\bigvee_j l_{ij}$ is called a *clause*. A *unit clause* is a clause with only one literal.

The previous notation and terminology about (positive/negative) literals, truth assignments, CNF, and (unit) clauses extend straightforwardly to quantifier-free first-order formulas.

2.1. SAT and CDCL SAT Solvers

We present here a brief description on how a CDCL SAT solver works. We refer the reader, for example, to Marques-Silva and Sakallah [1996], Moskewicz et al. [2001], and Marques-Silva et al. [2009] for a detailed description.

We assume the input propositional formula φ is in CNF. (If not, it is first CNF-ized as in Plaisted and Greenbaum [1986].) The assignment μ is initially empty, and it is updated in a stack-based manner. The SAT solver performs an external loop,

alternating three main phases: *Decision*, *Boolean Constraint Propagation (BCP)*, and *Backjumping and Learning*.

During *Decision* an unassigned literal l from φ is selected according to some heuristic criterion, and it is pushed into μ. l is called *decision literal* and the number of decision literals that are contained in μ immediately after deciding l is called the *decision level* of l.

Then BCP iteratively deduces the literals $l_1, l_2, \ldots$ deriving from the current assignment and pushes them into μ. BCP is based on the iterative application of *unit propagation*: If all but one literals in a clause are false, then the only unassigned literal l is added to μ; all negative occurrences of l in other clauses are declared false and all clauses with positive occurrences of l are declared satisfied. Current SAT solvers include rocket-fast implementations of BCP based on the *two-watched-literal scheme* (see Moskewicz et al. [2001] and Marques-Silva et al. [2009]). BCP is repeated until either no more literals can be deduced, so that the loop goes back to another decision step, or no more Boolean variable can be assigned, so that the SAT solver ends returning SAT, or μ falsifies some clause ψ of φ (*conflicting clause*).

In the latter case, *Backjumping and Learning* are performed. A process of *conflict analysis*[1] detects a subset η of μ, which actually caused the falsification of ψ (*conflict set*)[2] and the decision level blevel where to backtrack. Additionally, the *conflict clause* $\psi' \stackrel{\text{def}}{=} \neg\eta$ is added to φ (*Learning*) and the procedure backtracks up to blevel (*Backjumping*), popping out of μ all literals whose decision level is greater than blevel. When two contradictory literals $l, \neg l$ are assigned at level 0, the loop terminates, returning UNSAT.

Notice that CDCL SAT solvers implement "safe" strategies for deleting clauses when no more necessary, which guarantee the use of polynomial space without affecting the termination, correctness, and completeness of the procedure (see, e.g., Marques-Silva et al. [2009] and Nieuwenhuis et al. [2006]).

Many modern CDCL SAT solvers provide a *stack-based incremental interface* (see, e.g., Eén and Sörensson [2004]), by which it is possible to push/pop subformulas ϕ_i into a stack of formulas $\Phi \stackrel{\text{def}}{=} \{\phi_1, \ldots, \phi_k\}$, and check incrementally the satisfiability of $\bigwedge_{i=1}^{k} \phi_i$. The interface maintains most of the information about 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 ψ that was learned during a previous call on some Φ' if ψ was derived only from clauses that are still in Φ—provided ψ was not discharged in the meantime; in particular, if $\Phi' \subseteq \Phi$, then the solver can reuse all clauses learned while solving Φ'.

Another important feature of many incremental CDCL SAT solvers is their capability, when Φ is found unsatisfiable, to return a subset of formulas in Φ that caused the unsatisfiability of Φ. This is related to the problem of finding an *unsatisfiable core* of a formula (see, e.g., Lynce and Marques-Silva [2004]). Notice that such subset is not unique, and it is not necessarily minimal.

2.2. SMT and Lazy SMT Solvers

We assume a basic background knowledge on first-order logic. 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). Notice that, for better readability,

[1]When a clause ψ is falsified by the current assignment, a *conflict clause* ψ' is computed from ψ such that ψ' contains only one literal l_u, which has been assigned at the last decision level. ψ' is computed starting from $\psi' = \psi$ by iteratively resolving ψ' with the clause ψ_l causing the unit propagation of some literal l in ψ' until some stop criterion is met.

[2]That is, η is enough to force the unit propagation of the literals causing the failure of ψ.

with a little abuse of notation we often refer to a theory $\mathcal{T}$ instead of its corresponding signature; also, by "empty theory" we mean the empty theory over the empty signature; finally, by adopting the terminology in Remark 2.1, we say that a *variable* belongs to the signature of a theory $\mathcal{T}$ (or simply that it belongs to a theory $\mathcal{T}$).

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 $l \notin \mu$ from a set of all available $\mathcal{T}$-literals, such that $\{l_1, \ldots, l_n\} \models_{\mathcal{T}} l$, where $\{l_1, \ldots, l_n\} \subseteq \mu$. 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 if and only if 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 such that $\mu^p \models \varphi^p$, 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, or no more assignments are available (φ is $\mathcal{T}$-inconsistent).

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 not, and if the $\mathcal{T}$-Solver is able to perform a $\mathcal{T}$-deduction $\{l_1, \ldots, l_n\} \models_{\mathcal{T}} l$, 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-and-pop interface, maintaining and reusing the status of the search from one call and the other.

Another optimization is *pure-literal filtering*: If some $\mathcal{LA}(\mathbb{Q})$ atoms occur only positively (negatively, respectively) in the original formula (learned clauses are ignored), then we can safely drop every negative (positive, respectively) occurrence of them from the assignment μ to be checked by the $\mathcal{T}$-Solver [Sebastiani 2007]. 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; and (ii) it increases the chance of finding a $\mathcal{T}$-consistent satisfying assignment by removing "useless" $\mathcal{T}$-literals that may cause the $\mathcal{T}$-inconsistency of μ.

The previous schema is a coarse abstraction of the procedures underlying all the state-of-the-art lazy SMT tools. The interested reader is pointed to, for example, Nieuwenhuis et al. [2006], Sebastiani [2007], and Barrett et al. [2009] for details and further references. Importantly, some SMT solvers, including MATHSAT, inherit from their embedded SAT solver the capabilities of working incrementally and of returning the subset of input formulas causing the inconsistency, as described in Section 2.1.

The *theory of linear arithmetic on the rationals* ($\mathcal{LA}(\mathbb{Q})$) and on the integers ($\mathcal{LA}(\mathbb{Z})$) is one of the theories of main interest in SMT. It is a first-order theory whose atoms are of the form $(a_1x_1 + \cdots + a_nx_n \diamond b)$, that is, $(\mathbf{ax} \diamond b)$, such that $\diamond \in \{=, \neq, <, >, \leq, \geq\}$.

Efficient incremental and backtrackable procedures have been conceived in order to decide $\mathcal{LA}(\mathbb{Q})$ [Dutertre and de Moura 2006] and $\mathcal{LA}(\mathbb{Z})$ [Griggio 2012]. In particular, for $\mathcal{LA}(\mathbb{Q})$, most SMT solvers implement variants of the simplex-based algorithm by Dutertre and de Moura [2006], which is specifically designed for integration in a lazy SMT solver, since it is fully incremental and backtrackable and allows for aggressive $\mathcal{T}$-deduction. Another benefit of such algorithm is that it handles *strict inequalities* directly. Its method is based on the fact that a set of $\mathcal{LA}(\mathbb{Q})$ atoms Γ containing strict inequalities $S = \{0 < t_1, \ldots, 0 < t_n\}$ is satisfiable if and only if there exists a rational number $\epsilon > 0$ such that $\Gamma_\epsilon \stackrel{\text{def}}{=} (\Gamma \cup S_\epsilon) \backslash S$ is satisfiable, such that $S_\epsilon \stackrel{\text{def}}{=} \{\epsilon \leq t_1, \ldots, \epsilon \leq t_n\}$. The idea of Dutertre and de Moura [2006] is that of treating the *infinitesimal parameter* ϵ symbolically instead of explicitly computing its value. Strict bounds ($x < b$) are replaced with weak ones ($x \leq b - \epsilon$), and the operations on bounds are adjusted to take ϵ into account. We refer the reader to Dutertre and de Moura [2006] for details.

2.3. Linear Generalized Disjunctive Programming

Mixed Integer Linear Programming (MILP) is an extension of Linear Programming (LP) involving both discrete and continuous variables [Lodi 2009]. MILP problems have the following form:

$$\min\{\mathbf{cx} : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x}_j \in \mathbb{Z}\ \forall j \in I\}, \tag{1}$$

where $\mathbf{A}$ is a matrix, $\mathbf{c}$ and $\mathbf{b}$ are constant vectors, and $\mathbf{x}$ is the variable vector. A large variety of techniques and tools for MILP are available, mostly based on efficient combinations of LP, *branch-and-bound* search mechanism, and *cutting-plane* methods, resulting in a *branch-and-cut* approach (see, e.g., Lodi [2009]). SAT techniques have also been incorporated into these procedures for MILP (see Achterberg et al. [2008]).

The branch-and-bound search iteratively partitions the solution space of the original MILP problem into subproblems and solves their LP relaxation (i.e., a MILP problem where the integrality constraint on the variables $\mathbf{x}_j$, for all $j \in I$, is dropped) until all variables are integral in the optimal solution of the LP relaxation. Cutting planes (e.g., Gomory mixed-integer and lift-and-project cuts [Lodi 2009]) are linear inequalities that can be inferred and added to the original MILP problem and its subproblems in order to cut away noninteger solutions of the LP relaxation and obtain tighter relaxations.

LDP problems are LP problems where linear constraints are connected by the logical operations of conjunction and disjunction (see, e.g., Balas [1998]). The constraint set can be expressed by a disjunction of linear systems (*disjunctive normal form*):

$$\bigvee_{i \in I} (\mathbf{A}^i \mathbf{x} \geq \mathbf{b}^i) \tag{2}$$

or, alternatively, as a conjunction (*conjunctive normal form*):

$$(\mathbf{Ax} \geq \mathbf{b}) \wedge \bigwedge_{j=1}^{t} \bigvee_{k \in I_j} (\mathbf{c}^k \mathbf{x} \geq d^k), \tag{3}$$

or in an intermediate form called *regular form* (see, e.g., Balas [1983]). Notice that (3) can be obtained from (2) by factoring out the common inequalities ($\mathbf{Ax} \geq \mathbf{b}$) and then by applying the distributivity of $\wedge$ and $\vee$, although the latter step can cause a blowup in size. LDP problems are effectively solved by the lift-and-project approach that combines a family of cutting planes, called lift-and-project cuts, and the branch-and-bound schema (see, e.g., Balas and Bonami [2007]).

LGDP is a generalization of LDP, which has been proposed in Raman and Grossmann [1994] as an alternative model to the MILP problem. Unlike MILP, which

is based entirely on algebraic equations and inequalities, the LGDP model allows for combining algebraic and logical equations with Boolean propositions through Boolean operations, providing a much more natural representation of discrete decisions. Current approaches successfully address LGDP by reformulating and solving it as a MILP problem [Raman and Grossmann 1994; Vecchietti and Grossmann 2004; Sawaya and Grossmann 2005, 2012]; these reformulations focus on efficiently encoding disjunctions and logic propositions into MILP, so as to be fed to an efficient MILP solver like CPLEX.

The general formulation of a LGDP problem is the following [Raman and Grossmann 1994]:

$$\begin{aligned} \min \quad & \textstyle\sum_{k\in K} \mathbf{z}_k + \mathbf{d}\mathbf{x}, \\ \text{such that} \quad & \mathbf{B}\mathbf{x} \leq \mathbf{b}, \\ & \bigvee_{j\in J_k} \begin{bmatrix} Y^{jk} \\ \mathbf{A}^{jk}\mathbf{x} \leq \mathbf{a}^{jk} \\ \mathbf{z}_k = c^{jk} \end{bmatrix} \qquad \forall k \in K, \\ & \phi, \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{e}, \\ & \mathbf{z}_k, c^{jk} \in \mathbb{R}^1_+, Y^{jk} \in \{True, False\} \ \ \forall j \in J_k, \forall k \in K, \end{aligned} \tag{4}$$

where $\mathbf{x}$ is a vector of positive rational variables, $\mathbf{d}$ is a vector of positive rational values representing the cost per unit of each variable in $\mathbf{x}$, $\mathbf{z}$ is a vector of positive rational variables representing the cost assigned to each disjunction, c^{jk} are positive constant values, $\mathbf{e}$ is a vector of upper bounds for $\mathbf{x}$, and Y^{jk} are Boolean variables.

The disequalities $\mathbf{B}\mathbf{x} \leq \mathbf{b}$, where $(\mathbf{B}, \mathbf{b})$ is a $m \times (n+1)$ matrix, are the "common" constraints that must always hold.

Each disjunction $k \in K$ consists of at least two disjuncts $j \in J_k$, such that the jkth disjunct contains

(i) the Boolean variable Y^{jk}, representing discrete decisions;
(ii) a set of linear constraints $\mathbf{A}^{jk}\mathbf{x} \leq \mathbf{a}^{jk}$, where $(\mathbf{A}^{jk}, \mathbf{a}^{jk})$ is a $m_{jk} \times (n+1)$ matrix; and
(iii) the equality $\mathbf{z}_k = c^{jk}$, assigning the value c^{jk} to the cost variable $\mathbf{z}_k$.

Each disjunct is true if and only if all three elements (i)–(iii) are true. ϕ is a propositional formula, expressed in CNF, which must contain the "xor" constraints $\bigoplus_{j\in J_k} Y^{jk}$ for each $k \in K$, plus possibly other constraints. Intuitively, for each $k \in K$, the only variable Y^{jk}, which is set to true, selects the set of disequalities $\mathbf{A}^{jk}\mathbf{x} \leq \mathbf{a}^{jk}$, which are enforced, and hence it selects the relative cost c^{jk} of this choice to be assigned to the cost variable $\mathbf{z}_k$.

LGDP problems can be solved using MILP solvers by reformulating the original problem in different ways; big-M (BM) and convex hull (CH) are the two most common reformulations. In BM, the Boolean variables Y^{jk} and the logic constraints ϕ are replaced by binary variables $\mathbf{Y}_{jk}$ and linear inequalities as follows [Raman and Grossmann 1994]:

$$\begin{aligned} \min \quad & \textstyle\sum_{k\in K} \sum_{j\in J_k} c^{jk}\mathbf{Y}_{jk} + \mathbf{d}\mathbf{x}, \\ \text{such that} \quad & \mathbf{B}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{A}^{jk}\mathbf{x} - \mathbf{a}^{jk} \leq \mathbf{M}^{jk}(1 - \mathbf{Y}_{jk}) \qquad \forall j \in J_k, \forall k \in K, \\ & \textstyle\sum_{j\in J_k} \mathbf{Y}_{jk} = 1 \qquad \forall k \in K, \\ & \mathbf{D}\mathbf{Y} \leq \mathbf{D}', \\ & \mathbf{x} \in \mathbb{R}^n \ s.t. \ \mathbf{0} \leq \mathbf{x} \leq \mathbf{e}, \mathbf{Y}_{jk} \in \{0,1\} \ \ \forall j \in J_k, \forall k \in K, \end{aligned} \tag{5}$$

where $\mathbf{M}^{jk}$ are the "big-M" parameters that make redundant the system of constraint $j \in J_k$ in the disjunction $k \in K$ when $\mathbf{Y}_{jk} = 0$, and the constraints $\mathbf{DY} \leq \mathbf{D}'$ are derived from ϕ.

In CH, the Boolean variables Y^{jk} are replaced by binary variables $\mathbf{Y}_{jk}$ and the variables $\mathbf{x} \in \mathbb{R}^n$ are disaggregated into new variables $\mathbf{v} \in \mathbb{R}^n$ in the following way:

$$\begin{array}{rcll} \min & \sum_{k\in K}\sum_{j\in J_k} c^{jk}\mathbf{Y}_{jk} + \mathbf{dx}, & & \\ \text{such that} & \mathbf{Bx} \leq \mathbf{b}, & & \\ & \mathbf{A}^{kj}\mathbf{v}^{jk} \leq \mathbf{a}^{jk}\mathbf{Y}_{jk} & \forall j \in J_k, \forall k \in K, & \\ & \mathbf{x} = \sum_{j\in J_k} \mathbf{v}_{jk} & \forall k \in K, & (6) \\ & \mathbf{v}_{jk} \leq \mathbf{Y}_{jk}\mathbf{e}^{jk} & \forall j \in J_k, \forall k \in K, & \\ & \sum_{j\in J_k} \mathbf{Y}_{jk} = 1 & \forall k \in K, & \\ & \mathbf{DY} \leq \mathbf{D}', & & \\ & \mathbf{x}, \mathbf{v} \in \mathbb{R}^n \text{ such that } \mathbf{0} \leq \mathbf{x}, \mathbf{v},\ \mathbf{Y}_{jk} \in \{0,1\} & \forall j \in J_k, \forall k \in K, & \end{array}$$

where constants $\mathbf{e}^{jk}$ are upper bounds for variables $\mathbf{v}$ chosen to match the upper bounds on the variables $\mathbf{x}$.

Sawaya and Grossmann [2005] observed two facts. First, the relaxation of BM is often weak causing a higher number of nodes examined in the branch-and-bound search. Second, the disaggregated variables and new constraints increase the size of the reformulation leading to a high computational effort. In order to overcome these issues, they proposed a cutting plane method that consists in solving a sequence of BM relaxations with cutting planes that are obtained from CH relaxations. They provided an evaluation of the presented algorithm on three different problems: strip-packing, retrofit planning, and zero-wait job-shop scheduling problems.

3. OPTIMIZATION IN SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)

In this section, we define the problem addressed (Section 3.1), we introduce the formal foundations for its solution (Section 3.2), and we show how it generalizes many known optimization problems from the literature (Section 3.3).

3.1. Basic Definitions and Notation

In this article, we consider only signature-disjoint, stably infinite theories with equality $\mathcal{T}_i$ ("Nelson-Oppen theories" [Nelson and Oppen 1979]) and we focus our interest on $\mathcal{LA}(\mathbb{Q})$. In particular, in what follows, we assume $\mathcal{T}$ to be some stably infinite theory with equality, such that $\mathcal{LA}(\mathbb{Q})$ and $\mathcal{T}$ are signature disjoint. $\mathcal{T}$ can also be a combination of Nelson-Oppen theories.

We assume the standard model of $\mathcal{LA}(\mathbb{Q})$, whose domain is the set of rational numbers $\mathbb{Q}$.

Definition 3.1 (*OMT*($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), *OMT*($\mathcal{LA}(\mathbb{Q})$), *and* $\mathsf{min}_{\mathsf{cost}}$). Let φ be a ground SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) formula, and cost be a $\mathcal{LA}(\mathbb{Q})$ variable occurring in φ. We call an *optimization modulo* $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ *problem*, written *OMT*($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), the problem of finding a model for φ (if any) whose value of cost is minimum. We denote such value as $\mathsf{min}_{\mathsf{cost}}(\varphi)$. If φ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, then $\mathsf{min}_{\mathsf{cost}}(\varphi)$ is $+\infty$; if there is no minimum value for cost, then $\mathsf{min}_{\mathsf{cost}}(\varphi)$ is $-\infty$. We call an *optimization modulo* $\mathcal{LA}(\mathbb{Q})$ problem, written *OMT*($\mathcal{LA}(\mathbb{Q})$), an OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory.

A dual definition where we look for a *maximum* value is easy to formulate.

In order to make the discussion simpler, we assume, without loss of generality, that all $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formulas are *pure* [Nelson and Oppen 1979]. With a little abuse of notation, we say that an atom in a ground $\mathcal{T}_1 \cup \mathcal{T}_2$ formula is $\mathcal{T}_i$-pure if it contains only variables and symbols from the signature of $\mathcal{T}_i$, for every $i \in \{1, 2\}$; a $\mathcal{T}_1 \cup \mathcal{T}_2$ ground formula is pure if and only if all its atoms are either $\mathcal{T}_1$-pure or $\mathcal{T}_2$-pure. Although the purity assumption is not necessary (see Barrett et al. [2002]), it simplifies the explanation, since it allows us to speak about "$\mathcal{LA}(\mathbb{Q})$ atoms" or "$\mathcal{T}$ atoms" without further specifying. Moreover, every nonpure formula can be easily purified [Nelson and Oppen 1979].

We also assume, without loss of generality, that all $\mathcal{LA}(\mathbb{Q})$ atoms containing the variable cost are in the form $(t \bowtie \mathsf{cost})$, such that $\bowtie \in \{=, \leq, \geq, <, >\}$, and cost does not occur in t.

Definition 3.2 (*Bounds and range for* cost). If φ is in the form $\varphi' \wedge (\mathsf{cost} < c)$ ($\varphi' \wedge \neg(\mathsf{cost} < c)$, respectively) for some value $c \in \mathbb{Q}$, then we call c an *upper bound* (*lower bound*, respectively) for cost. If ub (lb , respectively) is the minimum upper bound (the maximum lower bound, respectively) for φ, we also call the interval $[\mathsf{lb}, \mathsf{ub}[$ the *range* of cost.

We adopt the convention of defining upper bounds to be strict and lower bounds to be nonstrict for a practical reason: Typically an upper bound $(\mathsf{cost} < c)$ derives from the fact that a model $\mathcal{I}$ of cost c has been previously found, while a lower bound $\neg(\mathsf{cost} < c)$ derives either from the user's knowledge (e.g., "the cost cannot be lower than zero") or from the fact that the formula $\varphi \wedge (\mathsf{cost} < c)$ has been previously found $\mathcal{T}$-unsatisfiable, while φ has not.

3.2. Theoretical Results

We present here the theoretical foundations of our procedures. The proofs of the novel results are reported in Appendix A.

The following facts follow straightforwardly from Definition 3.1.

PROPOSITION 3.3. *Let $\varphi, \varphi_1, \varphi_2$ be $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formulas and μ_1, μ_2 be truth assignments.*

(*a*) *If $\varphi_1 \models \varphi_2$, then $\mathsf{min}_{\mathsf{cost}}(\varphi_1) \geq \mathsf{min}_{\mathsf{cost}}(\varphi_2)$.*
(*b*) *If $\mu_1 \supseteq \mu_2$, then $\mathsf{min}_{\mathsf{cost}}(\mu_1) \geq \mathsf{min}_{\mathsf{cost}}(\mu_2)$.*
(*c*) *φ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if $\mathsf{min}_{\mathsf{cost}}(\varphi) < +\infty$.*

We recall first some definitions and results from the literature.

Definition 3.4. We say that a collection $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ of (possibly partial) assignments propositionally satisfying φ is *complete* if and only if, for every total assignment η such that $\eta \models_p \varphi$, there exists $\mu_j \in \mathcal{M}$ such that $\mu_j \subseteq \eta$.

THEOREM 3.5 ([SEBASTIANI 2007]). *Let φ be a $\mathcal{T}$ formula and let $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ be a complete collection of (possibly partial) truth assignments propositionally satisfying φ. Then, φ is $\mathcal{T}$-satisfiable if and only if μ_j is $\mathcal{T}$-satisfiable for some $\mu_j \in \mathcal{M}$.*

Theorem 3.5 is the theoretical foundation of the lazy SMT approach described in Section 2.2, where a CDCL SAT solver enumerates a complete collection $\mathcal{M}$ of truth assignments as previously, whose $\mathcal{T}$-satisfiability is checked by a $\mathcal{T}$-Solver. Notice that in Theorem 3.5 the theory $\mathcal{T}$ can be any combination of theories $\mathcal{T}_i$, including $\mathcal{LA}(\mathbb{Q})$.

Here we extend Theorem 3.5 to OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) as follows.

THEOREM 3.6. *Let φ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula and let $\mathcal{M} \stackrel{def}{=} \{\mu_1, \ldots, \mu_n\}$ be a complete collection of (possibly partial) truth assignments that propositionally satisfy φ. Then* $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu)$.

Notice that we implicitly define $min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu) \stackrel{\text{def}}{=} +\infty$ if $\mathcal{M}$ is empty. Since $\mathsf{min}_{\mathsf{cost}}(\mu)$ is $+\infty$ if μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, we can safely restrict the search for minima to the $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable assignments in $\mathcal{M}$.

If $\mathcal{T}$ is the empty theory, then the notion of $\mathsf{min}_{\mathsf{cost}}(\mu)$ is straightforward, since each μ is a conjunction of Boolean literals and of $\mathcal{LA}(\mathbb{Q})$ constraints, so that Theorem 3.6 provides the theoretical foundation for OMT($\mathcal{LA}(\mathbb{Q})$).

If , instead, $\mathcal{T}$ is not the empty theory, then each μ is a set of Boolean literals and of pure $\mathcal{T}$-literals and $\mathcal{LA}(\mathbb{Q})$ constraints sharing variables, so that the notion of $\mathsf{min}_{\mathsf{cost}}(\mu)$ is not straightforward. To cope with this fact, we first recall from the literature some definitions and an important result.

Definition 3.7 (*Interface variables, interface equalities*). Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite theories with equality and disjoint signatures, and let φ be a $\mathcal{T}_1 \cup \mathcal{T}_2$ formula. We call *interface variables* of φ the variables occurring in both $\mathcal{T}_1$-pure and $\mathcal{T}_2$-pure atoms of φ, and *interface equalities* of φ the equalities $(x_i = x_j)$ on the interface variables of φ.

As is common practice in SMT (see, e.g., Tinelli and Harandi [1996]), hereafter we consider only interface equalities modulo reflexivity and symmetry, that is, we implicitly assume some total order $\preceq$ on the interface variables x_i of φ, and we restrict, without loss of generality, the set of interface equalities on φ to $\mathcal{IE}(\varphi) \stackrel{\text{def}}{=} \{(x_i = x_j) \mid x_i \prec x_j\}$, dropping thus uninformative equalities like $(x_i = x_i)$ and considering only the first equality in each pair $\{(x_i = x_j), (x_j = x_i)\}$.

Notationwise, in what follows we use the subscripts e, d, i in "$\mu_{\ldots}$," like in "μ_{ed}," to denote conjunctions of equalities, disequalities, and inequalities between interface variables, respectively.

THEOREM 3.8 ([TINELLI AND HARANDI 1996]). *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite theories with equality and disjoint signatures; let $\mu \stackrel{def}{=} \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ be a conjunction of $\mathcal{T}_1 \cup \mathcal{T}_2$-literals such that each $\mu_{\mathcal{T}_i}$ is pure for $\mathcal{T}_i$. Then, μ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable if and only if there exists an equivalence class $Eq \subseteq \mathcal{IE}(\mu)$ over the interface variables of μ and the corresponding total truth assignment μ_{ed} to the interface equalities over μ:*

$$\mu_{ed} \stackrel{def}{=} \mu_e \wedge \mu_d, \quad \textit{such that } \mu_e \stackrel{def}{=} \bigwedge_{(x_i, x_j) \in Eq} (x_i = x_j), \quad \mu_d \stackrel{def}{=} \bigwedge_{(x_i, x_j) \notin Eq} \neg(x_i = x_j), \tag{7}$$

such that $\mu_{\mathcal{T}_k} \wedge \mu_{ed}$ is $\mathcal{T}_k$-satisfiable for every $k \in \{1, 2\}$.

Theorem 3.8 is the theoretical foundation of, among others, the *delayed theory combination* SMT technique for combined theories [Bozzano et al. 2006], where a CDCL SAT solver enumerates a complete collection of extended assignments $\mu \wedge \mu_{ed}$, which propositionally satisfy the input formula, and dedicated $\mathcal{T}_k$ solvers check independently the $\mathcal{T}_k$-satisfiability of $\mu_{\mathcal{T}_k} \wedge \mu_{ed}$, for each $k \in \{1, 2\}$.

We consider now a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula φ and a (possibly partial) truth assignment μ that propositionally satisfies it. μ can be written as $\mu \stackrel{\text{def}}{=} \mu_{\mathbb{B}} \wedge \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{\mathcal{T}}$, such that $\mu_{\mathbb{B}}$ is a consistent conjunction of Boolean literals, and $\mu_{\mathcal{LA}(\mathbb{Q})}$ and $\mu_{\mathcal{T}}$ are $\mathcal{LA}(\mathbb{Q})$-pure and $\mathcal{T}$-pure conjunctions of literals, respectively. (Notice that the $\mu_{\mathbb{B}}$ component does not affect the $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiability of μ.) Then, the following definitions and theorems show how $\mathsf{min}_{\mathsf{cost}}(\mu)$ can be defined and computed.

Definition 3.9. Let $\mu \stackrel{\text{def}}{=} \mu_{\mathbb{B}} \wedge \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{\mathcal{T}}$ be a truth assignment satisfying some $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ ground formula, such that $\mu_{\mathbb{B}}$ is a consistent conjunction of Boolean literals, and $\mu_{\mathcal{LA}(\mathbb{Q})}$ and $\mu_{\mathcal{T}}$ are $\mathcal{LA}(\mathbb{Q})$-pure and $\mathcal{T}$-pure conjunctions of literals, respectively. We call the *complete set of ed-extensions of* μ the set $\mathcal{EX}_{ed}(\mu) \stackrel{\text{def}}{=} \{\eta_1, \ldots, \eta_n\}$ of all possible assignments in the form $\mu \wedge \mu_{ed}$, where μ_{ed} is in the form (7), for every equivalence class Eq in $\mathcal{IE}(\mu)$.

THEOREM 3.10. *Let μ be as in Definition 3.9. Then,*

(*a*) $\mathsf{min}_{\mathsf{cost}}(\mu) = min_{\eta \in \mathcal{EX}_{ed}(\mu)} \mathsf{min}_{\mathsf{cost}}(\eta)$,
(*b*) *for all* $\eta \in \mathcal{EX}_{ed}(\mu)$,

$$\mathsf{min}_{\mathsf{cost}}(\eta) = \begin{cases} +\infty & \text{if } \mu_{\mathcal{T}} \wedge \mu_{ed} \text{ is } \mathcal{T}\text{-unsatisfiable or} \\ & \text{if } \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed} \text{ is } \mathcal{LA}(\mathbb{Q})\text{-unsatisfiable} \\ \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}) & \text{otherwise.} \end{cases}$$

We notice that, at least in principle, computing $\mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed})$ is an operation that can be performed by standard linear-programming techniques (see Section 4). Thus, by combining Theorems 3.6 and 3.10, we have a general method for computing $\mathsf{min}_{\mathsf{cost}}(\varphi)$ also in the general case of nonempty theory $\mathcal{T}$.

In practice, however, it is often the case that $\mathcal{LA}(\mathbb{Q})$ solvers/optimizers cannot handle efficiently negated equalities like, for example, $\neg(x_i = x_j)$ (see Dutertre and de Moura [2006]). Thus, a technique that is adopted by most SMT solvers is to expand them into the corresponding disjunction of strict inequalities $(x_i < x_j) \vee (x_i > x_j)$. This "case split" is typically efficiently handled directly by the embedded SAT solver.

We notice, however, that such case split may be applied also to interface equalities $(x_i = x_j)$, and that the resulting "interface inequalities" $(x_i < x_j)$ and $(x_i > x_j)$ cannot be handled by the other theory $\mathcal{T}$, because "$<$" and "$>$" are $\mathcal{LA}(\mathbb{Q})$-specific symbols. In order to cope with this fact, some more theoretical discussion is needed.

Definition 3.11. Let μ be as in Definition 3.9. We call the *complete set of edi-extensions of* μ the set $\mathcal{EX}_{edi}(\mu) \stackrel{\text{def}}{=} \{\rho_1, \ldots, \rho_n\}$ of all possible truth assignments in the form $\mu \wedge \mu_{ed} \wedge \mu_i$, where μ_{ed} is as in Definition 3.9 and μ_i is a total truth assignment to the atoms in $\{(x_i < x_j), (x_i > x_j) \,|(x_i = x_j) \in \mathcal{IE}(\mu)\}$ such that $\mu_{ed} \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-consistent.

μ_i assigns both $(x_i < x_j)$ and $(x_i > x_j)$ to false if $(x_i = x_j)$ is true in μ_{ed}, one of them to true and the other to false if $(x_i = x_j)$ is false in μ_{ed}. Intuitively, the presence of each negated interface equalities $\neg(x_i = x_j)$ in μ_{ed} forces the choice of one of the two parts $\langle (x_i < x_j), (x_i > x_j) \rangle$ of the solution space.

THEOREM 3.12. *Let μ be as in Definition 3.9. Then,*

(*a*) *μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if some $\rho \in \mathcal{EX}_{edi}(\mu)$ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable;*
(*b*) $\mathsf{min}_{\mathsf{cost}}(\mu) = min_{\rho \in \mathcal{EX}_{edi}(\mu)} \mathsf{min}_{\mathsf{cost}}(\rho)$;
(*c*) *for all $\rho \in \mathcal{EX}_{edi}(\mu)$, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable; and*
(*d*) *for all $\rho \in \mathcal{EX}_{edi}(\mu)$,*

$$\mathsf{min}_{\mathsf{cost}}(\rho) = \begin{cases} +\infty & \text{if } \mu_{\mathcal{T}} \wedge \mu_{ed} \text{ is } \mathcal{T}\text{-unsatisfiable or} \\ & \text{if } \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i \text{ is } \mathcal{LA}(\mathbb{Q})\text{-unsatisfiable} \\ \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i) & \text{otherwise.} \end{cases}$$

Thus, by combining Theorems 3.6 and 3.12 we have a general method for computing $\mathsf{min}_{\mathsf{cost}}(\varphi)$ in the case of nonempty theory $\mathcal{T}$, which is compliant with an efficient usage of standard $\mathcal{LA}(\mathbb{Q})$ solvers/optimizers.

3.3. OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) with Respect to Other Optimization Problems

In this section, we show that OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) captures many interesting optimization problems.

LP is a particular subcase of OMT($\mathcal{LA}(\mathbb{Q})$) with no Boolean component, such that $\varphi \stackrel{\text{def}}{=} \varphi' \wedge (\mathsf{cost} = \sum_i \mathbf{a}_i \mathbf{x}_i)$ and $\varphi' = \bigwedge_j (\sum_i \mathbf{A}_{ij} \mathbf{x}_i \leq \mathbf{b}_j)$.

LDP can also be encoded into OMT($\mathcal{LA}(\mathbb{Q})$), since (2) and (3) can be written, respectively, as

$$\bigvee_i \bigwedge_j (\mathbf{A}^i_j \mathbf{x} \geq \mathbf{b}^i_j), \tag{8}$$

$$\bigwedge_j (\mathbf{A}_j \mathbf{x} \geq \mathbf{b}_j) \wedge \bigwedge_{j=1}^{t} \bigvee_{k \in I_j} (\mathbf{c}^k \mathbf{x} \geq d^k), \tag{9}$$

where $\mathbf{A}^i_j$ and $\mathbf{A}_j$ are, respectively, the jth row of the matrices $\mathbf{A}^i$ and $\mathbf{A}$, and $\mathbf{b}^i_j$ and $\mathbf{b}_j$ are, respectively, the jth row of the vectors $\mathbf{b}^i$ and $\mathbf{b}$. Since (8) is not in CNF, the CNF-ization process of Plaisted and Greenbaum [1986] is then applied.

LGDP (4) is straightforwardly encoded into a OMT($\mathcal{LA}(\mathbb{Q})$) problem $\langle \varphi, \mathsf{cost} \rangle$:

$$\begin{aligned} \varphi \stackrel{\text{def}}{=} & (\mathsf{cost} = \textstyle\sum_{k \in K} \mathbf{z}_k + \mathbf{dx}) \wedge [[\mathbf{Bx} \leq \mathbf{b}]] \wedge \phi \wedge [[\mathbf{0} \leq \mathbf{x}]] \wedge [[\mathbf{x} \leq \mathbf{e}]] \\ \wedge & \textstyle\bigwedge_{k \in K} \bigvee_{j \in J_k} (Y^{jk} \wedge [[\mathbf{A}^{jk} \mathbf{x} \leq \mathbf{a}^{jk}]] \wedge (\mathbf{z}_k = c^{jk})) \\ \wedge & \textstyle\bigwedge_{k \in K} ((\mathbf{z}_k \geq min_{j \in J_k} c^{jk}) \wedge (\mathbf{z}_k \leq max_{j \in J_k} c^{jk})), \end{aligned} \tag{10}$$

such that $[[\mathbf{x} \bowtie \mathbf{a}]]$ and $[[\mathbf{Ax} \bowtie \mathbf{a}]]$ are abbreviations, respectively, for $\bigwedge_i (\mathbf{x}_i \bowtie \mathbf{a}_i)$ and $\bigwedge_i (\mathbf{A}_{i\cdot} \mathbf{x} \bowtie \mathbf{a}_i)$, $\bowtie \in \{=, \neq \leq, \geq, <, >\}$. The last conjunction "$\bigwedge_{k \in K}((\mathbf{z}_k \geq \dots))$" in (10) is not necessary, but it improves the performances of the SMT($\mathcal{LA}(\mathbb{Q})$) solver, because it allows for exploiting the early-pruning SMT technique (see Section 2.2) by providing a range for the values of the $\mathbf{z}_k$'s before the respective Y^{jk}'s are assigned. Since (10) is not in CNF, the CNF-ization process of Plaisted and Greenbaum [1986] is then applied.

PB constraints (see Roussel and Manquinho [2009]) in the form $(\sum_i \mathbf{a}_i X^i \leq b)$ such that X^i are Boolean atoms and $\mathbf{a}_i$ constant values in $\mathbb{Q}$, and cost functions $\mathsf{cost} = \sum_i \mathbf{a}_i X^i$, are encoded into OMT($\mathcal{LA}(\mathbb{Q})$) by rewriting each PB term $\sum_i \mathbf{a}_i X^i$ into the $\mathcal{LA}(\mathbb{Q})$ term $\sum_i \mathbf{x}_i$, $\mathbf{x}$ being an array of fresh $\mathcal{LA}(\mathbb{Q})$ variables, and by conjoining to φ the formula

$$\textstyle\bigwedge_i ((\neg X^i \vee (\mathbf{x}_i = \mathbf{a}_i)) \wedge (X^i \vee (\mathbf{x}_i = 0)) \wedge (\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)). \tag{11}$$

The term "$(\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)$" in (11) is not necessary, but it improves the performances of the SMT($\mathcal{LA}(\mathbb{Q})$) solver, because it allows for exploiting the early-pruning technique by providing a range for the values of the $\mathbf{x}_i$'s before the respective X^i's are assigned.

A (partial weighted) MaxSMT problem (see Nieuwenhuis and Oliveras [2006], Cimatti et al. [2010, 2013a]) 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, such that a positive weight $\mathbf{a}_i$ is associated with 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, such that $\psi_s \subseteq \varphi_s$ and $\varphi_h \cup \psi_s$ is $\mathcal{T}$-satisfiable. Notice that one can see $\mathbf{a}_i$ as a penalty to pay when the corresponding soft clause is not satisfied. A MaxSMT problem $\langle \varphi_h, \varphi_s \rangle$ can be encoded straightforwardly into an SMT problem with PB cost function $\langle \varphi', \mathsf{cost} \rangle$ by augmenting each soft $\mathcal{T}$ clause C_j with a fresh Boolean variable X^j as follows:

$$\varphi' \stackrel{\text{def}}{=} \varphi_h \cup \textstyle\bigcup_{C_j \in \varphi_s} \{(X^j \vee C_j)\}; \;\; \mathsf{cost} \stackrel{\text{def}}{=} \textstyle\sum_{C_j \in \varphi_s} \mathbf{a}_j X^j. \tag{12}$$

Vice versa, $\langle \varphi', \mathsf{cost} \stackrel{\text{def}}{=} \sum_j \mathbf{a}_j X^j \rangle$ can be encoded into MaxSMT:

$$\varphi_h \stackrel{\text{def}}{=} \varphi'; \quad \varphi_s \stackrel{\text{def}}{=} \bigcup_j \{(\underbrace{\neg X^j}_{\mathbf{a}_j})\}. \tag{13}$$

Thus, combining (11) and (12), optimization problems for SAT with PB constraints and MaxSAT can be encoded into OMT($\mathcal{LA}(\mathbb{Q})$), while those for SMT($\mathcal{T}$) with PB constraints and MaxSMT can be encoded into OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) , under the assumption that $\mathcal{T}$ matches the definition in Section 3.1.

Remark 3.13. We notice the deep difference between OMT($\mathcal{LA}(\mathbb{Q})$)/OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) and the problem of SAT/SMT with PB constraints and cost functions (or MaxSAT/MaxSMT) addressed in Nieuwenhuis and Oliveras [2006], Cimatti et al. [2010, 2013a]. With the latter problems, the value of cost is a deterministic consequence of a truth assignment to the atoms of the formula, so that the search has only a Boolean component, consisting in finding the cheapest truth assignment. With OMT($\mathcal{LA}(\mathbb{Q})$)/OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), instead, for every satisfying assignment μ it is also necessary to find the minimum-cost $\mathcal{LA}(\mathbb{Q})$ model for μ, so that the search has both a Boolean and a $\mathcal{LA}(\mathbb{Q})$ component.

4. PROCEDURES FOR OMT($\mathcal{LA}(\mathbb{Q})$) AND OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)

It might be noticed that very naive OMT($\mathcal{LA}(\mathbb{Q})$) or OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures could be straightforwardly implemented by performing a sequence of calls to an SMT solver on formulas like $\varphi \wedge (\mathsf{cost} \geq \mathsf{l}_i) \wedge (\mathsf{cost} < \mathsf{u}_i)$, each time restricting the range $[\mathsf{l}_i, \mathsf{u}_i[$ according to a linear-search or binary-search schema. With the linear-search schema, every time the SMT solver returns a model of cost c_i, a new constraint $(\mathsf{cost} < c_i)$ would be added to φ, and the solver would be invoked again. However, the SMT solver would repeatedly generate the same $\mathcal{LA}(\mathbb{Q})$-satisfiable truth assignment, each time finding a cheaper model for it. With the binary-search schema, the efficiency should improve. However, an initial lower bound should be necessarily required as input (which is not the case, e.g., of the problems in Section 5.3.)

In this section, we present more sophisticated procedures, based on the combination of SMT and minimization techniques. We first present and discuss an *offline* schema (Section 4.1) and an *inline* schema (Section 4.2) for an OMT($\mathcal{LA}(\mathbb{Q})$) procedure; then we show how to extend them to the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) case (Section 4.3).

4.1. An Offline Schema for OMT($\mathcal{LA}(\mathbb{Q})$)

The general schema for the offline OMT($\mathcal{LA}(\mathbb{Q})$) procedure is displayed in Algorithm 1. It takes as input an instance of the OMT($\mathcal{LA}(\mathbb{Q})$) problem plus optional 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 u (the value ub if φ is $\mathcal{LA}(\mathbb{Q})$-inconsistent). Notice that, by providing a lower bound lb (an upper bound ub, respectively), 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, respectively).

We represent φ as a set of clauses, which may be pushed or popped from the input formula stack of an incremental SMT solver. To this extent, every operation like "$\varphi \leftarrow \varphi \cup \{...\}$" ("$\varphi \leftarrow \varphi \backslash \{...\}$," respectively) in Algorithm 1, "{...}" being a clause set, should be interpreted as "push {...} into φ" ("pop {...} from φ," respectively).

First, the variables l, u defining the current range are initialized to lb and ub, respectively, the atom PIV to $\top$, and $\mathcal{M}$ is initialized to be an empty model. Then, the procedure adds to φ the bound constraints, if present, which restrict the search within the range $[\mathsf{l}, \mathsf{u}[$ (row 2). (Obviously, literals like $\neg(\mathsf{cost} < -\infty)$ and $(\mathsf{cost} < +\infty)$ are not

ALGORITHM 1: Offline OMT($\mathcal{LA}(\mathbb{Q})$) Procedure based on Mixed Linear/Binary Search

Require: $\langle \varphi, \mathsf{cost}, \mathsf{lb}, \mathsf{ub} \rangle$ {ub can be $+\infty$, lb can be $-\infty$}

```
 1: l ← lb; u ← ub; PIV ← ⊤; M ← ∅
 2: φ ← φ ∪ {¬(cost < l), (cost < u)}     // Push bound constraints into φ
 3: while (l < u ) do
 4:   if (BinSearchMode()) then      // Binary-search Mode
 5:     pivot ← ComputePivot(l, u)
 6:     PIV ← (cost < pivot)
 7:     φ ← φ ∪ {PIV}      // Push PIV into φ
 8:     ⟨res, μ⟩ ← SMT.IncrementalSolve(φ)
 9:     if (res = UNSAT) then
10:       η ← SMT.ExtractUnsatCore(φ)
11:     else
12:       η ← ∅
13:     end if
14:   else      // Linear-search Mode
15:     ⟨res, μ⟩ ← SMT.IncrementalSolve(φ)
16:     η ← ∅
17:   end if
18:   if (res = SAT) then
19:     ⟨M, u⟩ ← Minimize(cost, μ)
20:     φ ← φ ∪ {(cost < u)}     // Push new upper-bound constraint into φ
21:   else     // res = UNSAT
22:     if (PIV ∉ η) then
23:       l ← u
24:     else
25:       l ← pivot
26:       φ ← φ \ { PIV}     // Pop PIV from φ
27:       φ ← φ ∪ {¬PIV}     // Push ¬PIV into φ
28:     end if
29:   end if
30: end while
31: return ⟨M, u⟩
```

added.) The solution space is then explored iteratively (rows 3–30), reducing the current range $[\mathsf{l}, \mathsf{u}[$ to explore at each loop, until the range is empty. Then $\langle \mathcal{M}, \mathsf{u} \rangle$ is returned—$\langle \emptyset, \mathsf{ub} \rangle$ if there is no solution in $[\mathsf{lb}, \mathsf{ub}[$—$\mathcal{M}$ being the model of minimum cost u. Each loop may work in either *linear-search* or *binary-search* mode, driven by the heuristic BinSearchMode(). Notice that if $\mathsf{u} = +\infty$ or $\mathsf{l} = -\infty$, then BinSearchMode() returns false.

In *linear-search mode*, steps 5–13 and 25–27 are not executed. First, an incremental SMT($\mathcal{LA}(\mathbb{Q})$) solver is invoked on φ (row 15). (Notice that, given the incrementality of the solver, every operation in the form "$\varphi \leftarrow \varphi \cup \{\phi_i\}$" ($\varphi \leftarrow \varphi \backslash \{\phi_i\}$, respectively) is implemented as a "push" ("pop," respectively) operation on the stack representation of φ (see Section 2.1); it is also very important to recall that during the SMT call, φ is updated with the clauses that are learned during the SMT search.) Then, η is set to be empty, which forces condition 22 to hold.

If φ is $\mathcal{LA}(\mathbb{Q})$-satisfiable, then it is returned $\mathsf{res} = \text{SAT}$ and a $\mathcal{LA}(\mathbb{Q})$-satisfiable truth assignment μ for φ. Thus, Minimize is invoked on (the subset of $\mathcal{LA}(\mathbb{Q})$-literals of) μ,[3]

[3]Possibly after applying pure-literal filtering to μ (see Section 2.2).

returning the model $\mathcal{M}$ for μ of minimum cost u ($-\infty$ if and only if the problem is unbounded). The current solution u becomes the new upper bound, thus the $\mathcal{LA}(\mathbb{Q})$ atom $(\mathsf{cost} < \mathsf{u})$ is added to φ (row 20). Notice that, if the problem is unbounded, then for some μ Minimize will return $-\infty$, forcing condition 3 to be false and the whole process to stop. If φ is $\mathcal{LA}(\mathbb{Q})$-unsatisfiable, then no model in the current cost range $[\mathsf{l}, \mathsf{u}[$ can be found; hence the flag l is set to u, forcing the end of the loop.

In *binary-search mode* at the beginning of the loop (steps 5–13), the value $\mathsf{pivot} \in]\mathsf{l}, \mathsf{u}[$ is computed by the heuristic function ComputePivot (in the simplest form, pivot is $(\mathsf{l} + \mathsf{u})/2$), and the possibly new atom $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is pushed into the formula stack, so as to temporarily restrict the cost range to $[\mathsf{l}, \mathsf{pivot}[$. Then the incremental SMT solver is invoked on φ; if the result is UNSAT, the feature SMT.ExtractUnsatCore is activated, which returns also a subset η of formulas in (the formula stack of) φ, which caused the unsatisfiability of φ (see Section 2.1). This exploits techniques similar to unsat-core extraction [Lynce and Marques-Silva 2004]. (In practical implementations, it is not strictly necessary to explicitly produce the unsat core η; rather, it suffices to check if $\mathsf{PIV} \in \eta$.)

If φ is $\mathcal{LA}(\mathbb{Q})$-satisfiable, then the procedure behaves as in linear-search mode. If, instead, φ is $\mathcal{LA}(\mathbb{Q})$-unsatisfiable, we look at η and distinguish two subcases. If PIV does not occur in η, this means that $\varphi \backslash \{\mathsf{PIV}\}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent, that is, there is no model in the whole cost range $[\mathsf{l}, \mathsf{u}[$. Then the procedure behaves as in linear-search mode, forcing the end of the loop. Otherwise, we can only conclude that there is no model in the cost range $[\mathsf{l}, \mathsf{pivot}[$, so that we still need to explore the cost range $[\mathsf{pivot}, \mathsf{u}[$. Thus, l is set to pivot, PIV is popped from φ, and its negation is pushed into φ. Then the search proceeds, investigating the cost range $[\mathsf{pivot}, \mathsf{u}[$.

We notice an important fact: If BinSearchMode() always returned true, then Algorithm 1 would not necessarily terminate. In fact, an SMT solver invoked on φ may return a set η containing PIV even if $\varphi \backslash \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent.[4] Thus, for example, the procedure might get stuck in a "Zeno"[5] infinite loop, each time halving the cost range right-bound (e.g., $[-1, 0[$, $[-1/2, 0[$, $[-1/4, 0[$, ...).

To cope with this fact, however, it suffices to guarantee that BinSearchMode() returns false after a finite number of such steps, guaranteeing thus that eventually a linear-search loop will be forced, which detects the inconsistency. In our implementations, we have empirically chosen to force one linear-search loop after *every* binary-search loop returning UNSAT, because satisfiable calls are typically much cheaper than unsatisfiable ones. We have empirically verified in previous tests that this was, in general, the best option, since introducing this test caused no significant overhead and prevented the chains of (very expensive) unsatisfiable calls where they used to occur.

Under such hypothesis, as a consequence of Theorem 3.6 of Section 3.2, we have that

(i) Algorithm 1 terminates. In linear-search mode it terminates because there are only a finite number of candidate truth assignments μ to be enumerated, and steps 19–20 guarantee that the same assignment μ will never be returned twice by the SMT solver. In mixed linear/binary-search mode, as previously, it terminates since there can be at most finitely many binary-search loops between two consequent linear-search loops;

[4] A CDCL-based SMT solver implicitly builds a resolution refutation whose leaves are either clauses in φ or $\mathcal{LA}(\mathbb{Q})$ lemmas, and the set η represents the subset of clauses in φ that occur as leaves of such proof (see, e.g., Cimatti et al. [2011] for details). If the SMT solver is invoked on φ, even $\varphi \backslash \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent, then it can "use" PIV and return a proof involving it, even though another PIV-less proof exists.

[5] In the famous Zeno's paradox, Achilles never reaches the tortoise for a similar reason.

(ii) Algorithm 1 returns a model of minimum cost, because it explores the whole search space of candidate truth assignments, and for every suitable assignment μ Minimize finds the minimum-cost model for μ;
(iii) Algorithm 1 requires polynomial space, under the assumption that the underlying CDCL SAT solver adopts a polynomial-size clause-deleting strategy (which is typically the case of SMT solvers, including MATHSAT).

4.1.1. Handling Strict Inequalities. Minimize is a simple extension of the simplex-based $\mathcal{LA}(\mathbb{Q})$-Solver of Dutertre and de Moura [2006], which is invoked after one solution is found, minimizing it by standard simplex techniques. We recall that the algorithm in Dutertre and de Moura [2006] can handle strict inequalities. Thus, if μ contains strict inequalities, then Minimize temporarily relaxes them into nonstrict ones, and then it finds a solution $\mathbf{x}$ of minimum cost min of the relaxed problem, namely, μ_{rel}. (Notice that this could be done also by any standard LP package.) Then,

(1) if such minimum-cost solution $\mathbf{x}$ lays only on nonstrict inequalities, then $\mathbf{x}$ is also a solution of the nonrelaxed problem μ, hence min can be returned;
(2) otherwise, we temporarily add the constraint (cost $\leq$ min) to the nonrelaxed version of μ and then we invoke on it the $\mathcal{LA}(\mathbb{Q})$-solving procedure of Dutertre and de Moura [2006] (without minimization), since such algorithm can handle strict inequalities. Then,
   (i) if the procedure returns SAT, then μ has a model of cost min. If so, then the value min can be returned, and (cost $<$ min) can be pushed into φ;
   (ii) otherwise, μ has no model of cost min. If so, since μ has a convex set of solutions whose cost is strictly greater than min and there is a solution of cost min for the relaxed problem μ_{rel}, then for some $\delta > 0$ and for every cost $c \in]\mathsf{min}, \mathsf{min} + \delta]$ there exists a solution for μ of cost c. (If needed explicitly, such solution can be computed using the techniques for handling strict inequalities described in Dutertre and de Moura [2006].) Thus the value min can be tagged as a nonstrict minimum and returned, so that the constraint (cost $\leq$ min), rather than (cost $<$ min), is pushed into φ.

Notice that situation (2)(i) is very rare in practice, but it is possible in principle, as illustrated in the following example.

Example 4.1. Suppose we have that $\mu = \{(\mathsf{cost} \geq 1), (\mathsf{cost} > y), (\mathsf{cost} > -y)\}$. If we temporarily relax strict inequalities into nonstrict ones, then $\{\mathsf{cost} = 1, y = 1\}$ is a minimum-cost solution that lays on the strict inequality (cost $> y$). Nevertheless, there is a solution of cost 1 for the unrelaxed problem (e.g., $\{\mathsf{cost} = 1, y = 0.9999\}$).

Notice also that (cost $\leq$ min) is pushed into φ only if the minimum cost of the current assignment μ is strictly greater than min, as in situation (2)(ii). This prevents the SMT solver from returning μ again. Therefore, the termination, the correctness, and the completeness of the algorithm are guaranteed also in the case where some truth assignments have strict minimum costs.

4.1.2. Discussion. We note a few facts about this procedure.

(1) If Algorithm 1 is interrupted (e.g., by a timeout device), then u can be returned, representing the best approximation of the minimum cost found so far.
(2) The incrementality of the SMT solver (see Sections 2.1 and 2.2) plays an essential role here, since at every call SMT.IncrementalSolve resumes the status of the search at the end of the previous call, only with tighter cost range constraints. (Notice that at each call here, the solver can reuse all previously learned clauses.) To this extent, one can see the whole process mostly as only one SMT process, which is interrupted

and resumed each time a new model is found, in which cost range constraints are progressively tightened.

(3) In Algorithm 1, all the literals constraining the cost range (i.e., $\neg(\mathsf{cost} < \mathsf{l})$, $(\mathsf{cost} < \mathsf{u})$) are added to φ as unit clauses; thus, inside SMT.IncrementalSolve, they are immediately unit-propagated, becoming part of each truth assignment μ from the very beginning of its construction. As soon as novel $\mathcal{LA}(\mathbb{Q})$-literals are added to μ, which prevent it from having a $\mathcal{LA}(\mathbb{Q})$-model of cost in $[\mathsf{l}, \mathsf{u}[$, the $\mathcal{LA}(\mathbb{Q})$-solver invoked on μ by early-pruning calls (see Section 2.2) returns UNSAT and the $\mathcal{LA}(\mathbb{Q})$-lemma $\neg\eta$ describing the conflict $\eta \subseteq \mu$, triggering theory backjumping and learning. To this extent, SMT.IncrementalSolve implicitly plays a form of *branch and bound*: (i) decide a new literal l and unit- or theory-propagate the literals that derive from l ("branch") and (ii) backtrack as soon as the current branch can no more be expanded into models in the current cost range ("bound").

(4) The unit clause $\neg(\mathsf{cost} < \mathsf{l})$ plays a role even in linear-search mode, since it helps pruning the search inside SMT.IncrementalSolve.

(5) In binary-search mode, the range-partition strategy may be even more aggressive than that of standard binary search, because the minimum cost u returned in row 19 can be smaller than pivot, so that the cost range is more than halved.

(6) Unlike with other domains (e.g., search in sorted arrays), here binary search is not "obviously faster" than linear search, because the unsatisfiable calls to SMT.IncrementalSolve are typically much more expensive than the satisfiable ones, since they must explore the whole Boolean search space rather than only a portion of it—although with a higher pruning power, due to the stronger constraint induced by the presence of pivot. Thus, we have a trade-off between a typically much smaller number of calls plus a stronger pruning power in binary search versus an average much smaller cost of the calls in linear search. To this extent, it is possible, in principle, to use dynamic/adaptive strategies for ComputePivot (see Sellmann and Kadioglu [2008]).

4.2. An Inline Schema for OMT($\mathcal{LA}(\mathbb{Q})$)

With the inline schema, the whole optimization procedure is pushed inside the SMT solver by embedding the range-minimization loop inside the CDCL Boolean-search loop of the standard lazy SMT schema of Section 2.2. The SMT solver, which is thus called only once, is modified as follows.

Initialization. The variables $\mathsf{lb}, \mathsf{ub}, \mathsf{l}, \mathsf{u}, \mathsf{PIV}, \mathsf{pivot}, \mathcal{M}$ are brought inside the SMT solver, and are initialized as in Algorithm 1, steps 1–2.

Range Updating and Pivoting. Every time the search of the CDCL SAT solver gets back to decision level 0, the range $[\mathsf{l}, \mathsf{u}[$ is updated such that u (l, respectively) is assigned the lowest (highest, respectively) value u_i (l_i, respectively) such that the atom $(\mathsf{cost} < \mathsf{u}_i)$ ($\neg(\mathsf{cost} < \mathsf{l}_i)$, respectively) is currently assigned at level 0. (If $\mathsf{u} \leq \mathsf{l}$, or two literals $l, \neg l$ are both assigned at level 0, then the procedure terminates, returning the current value of u.) Then BinSearchMode() is invoked: If it returns true, then ComputePivot computes $\mathsf{pivot} \in]\mathsf{l}, \mathsf{u}[$, and the (possibly new) atom $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is decided to be true (level 1) by the SAT solver. This mimics steps 5–7 in Algorithm 1, temporarily restricting the cost range to $[\mathsf{l}, \mathsf{pivot}[$.

Decreasing the Upper Bound. When an assignment μ propositionally satisfying φ is generated that is found $\mathcal{LA}(\mathbb{Q})$-consistent by $\mathcal{LA}(\mathbb{Q})$-Solver, μ is also fed to Minimize, returning the minimum cost min of μ; then the unit clause $(\mathsf{cost} < \mathsf{min})$ is learned and fed to the backjumping mechanism, which forces the SAT solver to backjump to level 0 and then to unit-propagate $(\mathsf{cost} < \mathsf{min})$. This case mirrors steps 18–20 in Algorithm 1, permanently restricting the cost range to $[\mathsf{l}, \mathsf{min}[$. Minimize is embedded within the

$\mathcal{LA}(\mathbb{Q})$-Solver, so that it is called incrementally after it, without restarting its search from scratch.

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 such that the conflict analysis of the SAT solver produces a conflict clause in the form $\neg\mathsf{PIV} \vee \neg\eta'$ such that all literals in η' are assigned true at level 0 (i.e., $\varphi \wedge \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent), then the SAT solver backtracks to level 0, unit-propagating $\neg\mathsf{PIV}$. This case mirrors steps 25–27 in Algorithm 1, permanently restricting the cost range to [pivot, u[.

Although the modified SMT solver mimics to some extent the behaviour of Algorithm 1, the "control" of the range-restriction process is handled by the standard SMT search. To this extent, notice that also other situations may allow for restricting the cost range: for example, if $\varphi \wedge \neg(\mathsf{cost} < \mathsf{l}) \wedge (\mathsf{cost} < \mathsf{u}) \models (\mathsf{cost} \bowtie \mathsf{m})$ for some atom $(\mathsf{cost} \bowtie \mathsf{m})$ occurring in φ such that $\mathsf{m} \in [\mathsf{l}, \mathsf{u}[$ and $\bowtie \in \{\leq, <, \geq, >\}$, then the SMT solver may backjump to decision level 0 and propagate $(\mathsf{cost} \bowtie \mathsf{m})$, further restricting the cost range.

The same facts (1)–(6) about the offline procedure in Section 4.1 hold for the inline version. The efficiency of the inline procedure can be further improved as follows.

Activating Previously Learned Clauses. In binary-search mode, when an assignment μ with a novel minimum min is found, not only $(\mathsf{cost} < \mathsf{min})$ but also $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is learned as unit clause, although the latter is redundant from the logical perspective because $\mathsf{min} < \mathsf{pivot}$. In fact, the unit clause PIV allows the SAT solver for reusing all the clauses in the form $\neg\mathsf{PIV} \vee C_i$, which have been learned when investigating the cost range [l, pivot[, by unit-resolving them into the corresponding clauses C_i. (In Algorithm 1 this is done implicitly, since PIV is not popped from φ before step 26.) Notice that the previous trick is useful because the algorithm of Dutertre and de Moura [2006] is not "$\mathcal{T}$-deduction complete," that is, it is not guaranteed to $\mathcal{T}$-deduce PIV from $\{\ldots, (\mathsf{cost} < \mathsf{min})\}$.

In addition, the $\mathcal{LA}(\mathbb{Q})$-inconsistent assignment $\mu \wedge (\mathsf{cost} < \mathsf{min})$ may be fed to $\mathcal{LA}(\mathbb{Q})$-Solver and the negation of the returned conflict $\neg\eta \vee \neg(\mathsf{cost} < \mathsf{min})$ such that $\eta \subseteq \mu$, can be learned, preventing the SAT solver from generating any assignment containing η in the future.

Tightening. In binary-search mode, if $\mathcal{LA}(\mathbb{Q})$-Solver returns a conflict set $\eta \cup \{\mathsf{PIV}\}$, then it is further asked to find the maximum value max such that $\eta \cup \{(\mathsf{cost} < \mathsf{max})\}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent. (This is done with a simple modification of the algorithm in Dutertre and de Moura [2006].)

—If $\mathsf{max} \geq \mathsf{u}$, then the clause $C^* \stackrel{\text{def}}{=} \neg\eta \vee \neg(\mathsf{cost} < \mathsf{u})$ is used do drive backjumping and learning instead of $C \stackrel{\text{def}}{=} \neg\eta \vee \neg\mathsf{PIV}$. Since the unit clause $(\mathsf{cost} < \mathsf{u})$ is permanently assigned at level 0, this is equivalent to learning only $\neg\eta$, so that the dependency of the conflict from PIV is removed. Eventually, instead of using C to drive backjumping to level 0 and then to propagate $\neg\mathsf{PIV}$, the SMT solver may use C^* (which is the same as using $\neg\eta$), then forcing the procedure to stop.

—If $\mathsf{u} > \mathsf{max} > \mathsf{pivot}$, then the clauses $C_1 \stackrel{\text{def}}{=} \neg\eta \vee \neg(\mathsf{cost} < \mathsf{max})$ and $C_2 \stackrel{\text{def}}{=} \neg\mathsf{PIV} \vee (\mathsf{cost} < \mathsf{max})$ are used to drive backjumping and learning instead of $C \stackrel{\text{def}}{=} \neg\eta \vee \neg\mathsf{PIV}$. (Notice that C can be inferred by resolving C_1 and C_2.) In particular, C_2 forces backjumping to level 1 and unit-propagating the (possibly fresh) atom $(\mathsf{cost} < \mathsf{max})$; eventually, instead of using C do drive backjumping to level 0 and then to propagate $\neg\mathsf{PIV}$, the SMT solver may use C_1 for backjumping to level 0 and then to propagate $\neg(\mathsf{cost} < \mathsf{max})$, restricting the range to [max, u[rather than to [pivot, u[.

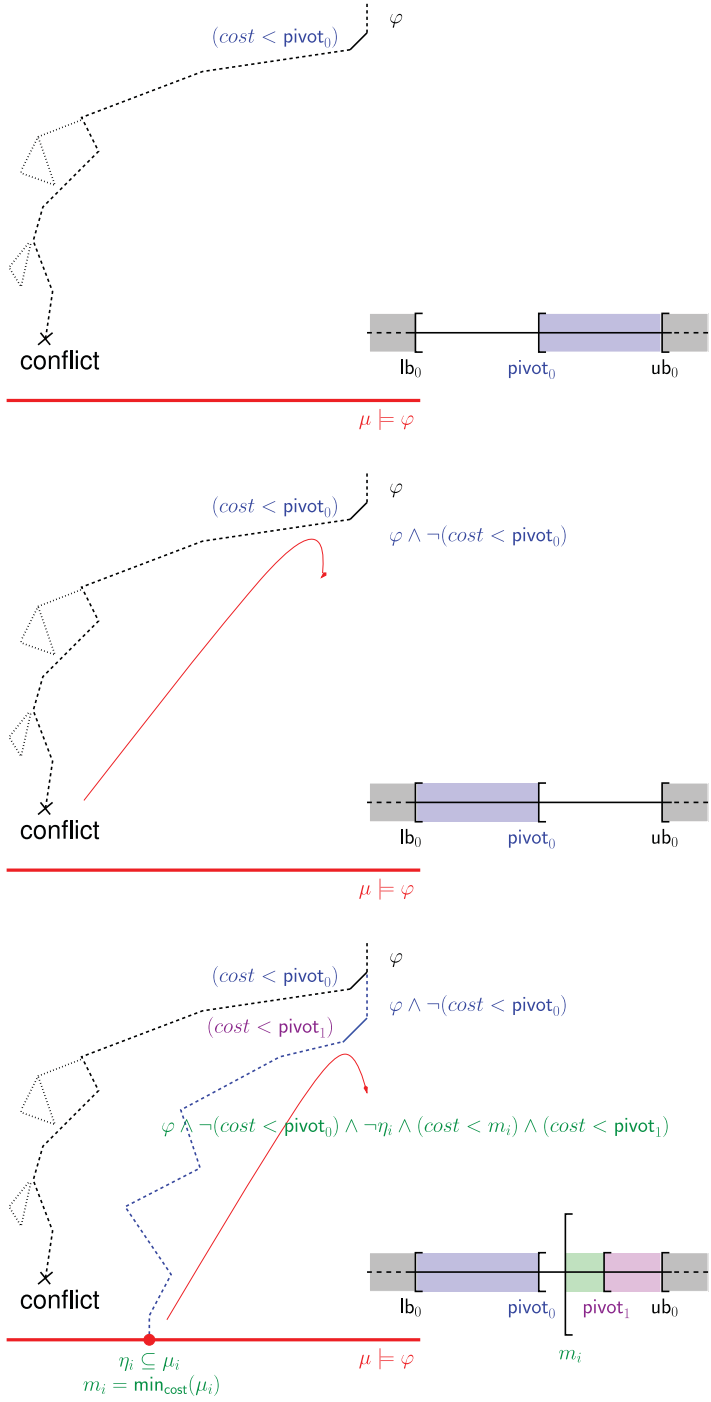

Fig. 1. One peice of possible execution of an inline procedure. (i) Pivoting on $(\mathsf{cost} < \mathsf{pivot}_0)$. (ii) Increasing the lower bound to pivot_0. (iii) Decreasing the upper bound to $\mathsf{min}_{\mathsf{cost}}(\mu_i)$.

Notice that tightening is useful because the algorithm of Dutertre and de Moura [2006] is guaranteed neither to find the "tightest" theory conflict $\eta \cup \{(\mathsf{cost} < \mathsf{max})\}$, nor to $\mathcal{T}$-deduce $(\mathsf{cost} < \mathsf{max})$ from $\{\ldots, \mathsf{PIV}\}$.

Example 4.2. Consider the formula $\varphi \stackrel{\text{def}}{=} \psi \wedge (\mathsf{cost} \geq a + 15) \wedge (a \geq 0)$ for some ψ in the cost range $[0, 16[$. With binary-search deciding $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < 8)$, the $\mathcal{LA}(\mathbb{Q})$-Solver produces the lemma $C \stackrel{\text{def}}{=} \neg(\mathsf{cost} \geq a + 15) \vee \neg(a \geq 0) \vee \neg\mathsf{PIV}$, causing a backjumping step to level 0 on C and the unit-propagation of $\neg\mathsf{PIV}$, restricting the range to $[8, 16[$; it takes a sequence of similar steps to progressively restrict the range to $[12, 16[$, $[14, 16[$, and $[15, 16[$. If, instead, the $\mathcal{LA}(\mathbb{Q})$-Solver produces the lemmas $C_1 \stackrel{\text{def}}{=} \neg(\mathsf{cost} \geq a + 15) \vee \neg(a \geq 0) \vee \neg(\mathsf{cost} < 15)$ and $C_2 \stackrel{\text{def}}{=} \neg\mathsf{PIV} \vee (\mathsf{cost} < 15)$, then this first causes a backjumping step on C_2 to level 1 with the unit propagation of $(\mathsf{cost} < 15)$, and then a backjumping step on C_1 to level zero with the unit propagation of $\neg(\mathsf{cost} < 15)$, which directly restricts the range to $[15, 16[$.

Adaptive Mixed Linear/Binary Search Strategy. An adaptive version of the heuristic BinSearchMode() decides the next search mode according to the ratio between the progress obtained in the latest binary- and linear-search steps and their respective costs. If either ub or lb is not present—or if we are immediately after an UNSAT binary-search step, in compliance with the strategy to avoid infinite "Zeno" sequences described in Section 4.1—then the heuristic selects linear-search mode. Otherwise, it selects binary-search mode if and only if

$$\left|\frac{\Delta\mathsf{ub}_{lin}}{\Delta\#\mathsf{conf}_{lin}}\right| < \left|\frac{\Delta\mathsf{ub}_{bin}}{\Delta\#\mathsf{conf}_{bin}}\right|,$$

where $\Delta\mathsf{ub}_{lin}$ and $\Delta\mathsf{ub}_{bin}$ are, respectively, the variations of the upper bound ub in the latest linear-search and SAT binary-search steps performed, estimating the progress achieved by such steps, while $\Delta\#\mathsf{conf}_{lin}$ and $\Delta\#\mathsf{conf}_{bin}$ are, respectively, the number of conflicts produced in such steps, estimating their expense.

Overall, the inline version described in this section presents some potential computational advantages with respect to the offline version of Algorithm 1. First, despite the incrementality of the calls to the SMT solver, suspending and resuming it may cause some overhead, because at every call the decision stack is popped to decision level 0, so that some extra decisions, unit propagations, and early-pruning calls to the $\mathcal{T}$-Solver may be necessary to get back to the previous search status. Second, in Algorithm 1 the procedure Minimize is invoked from scratch in a nonincremental way, while in the inline version it is embedded inside the $\mathcal{LA}(\mathbb{Q})$-Solver, so that it starts the minimization process from an existing solution rather than from scratch. Third, Algorithm 1 requires computing the unsatisfiable core of φ—or at least checking if PIV belongs to such unsat core—which causes overhead. Notice that the problem of computing efficiently minimal unsat cores in SMT is still ongoing research (see Cimatti et al. [2011]), so that in Algorithm 1 there is a tradeoff between the cost of reducing the size of the cores and the probability of performing useless optimization steps.

4.3. Extensions to OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)

We recall the terminology, assumptions, definitions, and results of Section 3.2. Theorems 3.6, 3.10, and 3.12 allow for extending to the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) case the procedures of Sections 4.1 and 4.2 as follows.

As suggested by Theorem 3.10, straightforward OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) extensions of the procedures for OMT($\mathcal{LA}(\mathbb{Q})$) of Sections 4.1 and 4.2 would be such that the SMT solver

enumerates *ed*-extended satisfying truth assignments $\eta \stackrel{\text{def}}{=} \mu \wedge \mu_{ed}$ as in Definition 3.9, checking the $\mathcal{T}$- and $\mathcal{LA}(\mathbb{Q})$-consistency of its components $\mu_{\mathcal{T}} \wedge \mu_{ed}$ and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$, respectively, and then minimizing the $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$ component. Termination is guaranteed by the fact that each $\mathcal{EX}_{ed}(\mu)$ is a finite set, while correctness and completeness is guaranteed by Theorems 3.6 and 3.10.

Nevertheless, as suggested in Section 3.2, minimizing $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$ efficiently could be problematic due to the presence of negated interface equalities $\neg(x_i = x_j)$. Thus, alternative "asymmetric" procedures, in compliance with the efficient usage of $\mathcal{LA}(\mathbb{Q})$ solvers in SMT, should instead enumerate *edi*-extended satisfying truth assignments $\rho \stackrel{\text{def}}{=} \mu \wedge \mu_{edi}$ as in Definition 3.11, checking the $\mathcal{T}$- and $\mathcal{LA}(\mathbb{Q})$-consistency of its components $\mu_{\mathcal{T}} \wedge \mu_{ed}$ and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ei}$, respectively, and then minimizing the $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ei}$ component. This prevents from passing negated interface equalities to Minimize. As before, termination is guaranteed by the fact that each $\mathcal{EX}_{edi}(\mu)$ is a finite set, whilst correctness and completeness is guaranteed by Theorems 3.6 and 3.12.

This motivates and explains the following OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) variants of the offline and inline procedures of Sections 4.1 and 4.2, respectively.

Algorithm 1 is modified as follows. First, SMT.IncrementalSolve in steps 8 and 15 is asked to return also a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ model $\mathcal{I}$. Then, in step 19, Minimize is invoked instead on $\langle \mathsf{cost}, \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{ei} \rangle$, such that

$$\begin{array}{rll} \mu_{ei} \stackrel{\text{def}}{=} & \{(x_i = x_j), \neg(x_i < x_j), \neg(x_i > x_j) & \mid (x_i = x_j) \in \mathcal{IE}(\mu),\ \mathcal{I} \models (x_i = x_j)\} \\ \cup & \{(x_i < x_j), \neg(x_i > x_j) & \mid (x_i = x_j) \in \mathcal{IE}(\mu),\ \mathcal{I} \models (x_i < x_j)\} \\ \cup & \{(x_i > x_j), \neg(x_i < x_j) & \mid (x_i = x_j) \in \mathcal{IE}(\mu),\ \mathcal{I} \models (x_i > x_j)\}. \end{array}$$

In practice, the negated strict inequalities $\neg(x_i < x_j), \neg(x_i > x_j)$ are omitted from μ_{ei}, because they are entailed by the corresponding non-negated equalities/inequalities.

The implementation of an inline OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedure comes nearly for free once the SMT solver handles $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-solving by *delayed theory combination* [Bozzano et al. 2006], with the strategy of automatically case splitting disequalities $\neg(x_i = x_j)$ into the two inequalities $(x_i < x_j)$ and $(x_i > x_j)$, which is implemented in MATHSAT: The solver enumerates truth assignments in the form $\rho \stackrel{\text{def}}{=} \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{eid} \wedge \mu_{\mathcal{T}}$ as in Definition 3.11, and passes $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ei}$ and $\mu_{\mathcal{T}} \wedge \mu_{ed}$ to the $\mathcal{LA}(\mathbb{Q})$-Solver and $\mathcal{T}$-Solver, respectively. (Notice that this strategy, although not explicitly described in Bozzano et al. [2006], implicitly implements points (*a*) and (*c*) of Theorem 3.12.) If so, then, in accordance with points (*b*) and (*d*) of Theorem 3.12, it suffices to apply Minimize to $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ei}$, then learn (cost $<$ min) and use it for backjumping, as in Section 4.2. As with the offline version, in practice the negated strict inequalities are omitted from μ_{ei}, because they are entailed by the corresponding non-negated equalities/inequalities.

5. EXPERIMENTAL EVALUATION

We have implemented both the offline and inline OMT($\mathcal{LA}(\mathbb{Q})$) procedures and the inline OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures of Section 4 on top of MATHSAT5[6] [Cimatti et al. 2013b]; we refer to them as OPTIMATHSAT. MATHSAT5 is a state-of-the-art SMT solver that supports most of the quantifier-free SMT-LIB theories and their combinations, and provides many other SMT functionalities (like, e.g., unsat-core extraction [Cimatti et al. 2011], interpolation [Cimatti et al. 2010], and All-SMT [Cavada et al. 2007]).

We consider different configurations of OPTIMATHSAT, depending on the approach (offline vs. inline, denoted by "-OF" and "-IN") and on the search schema (linear vs.

[6]http://mathsat.fbk.eu/.

binary vs. adaptive, denoted, respectively, by "-LIN," "-BIN," and "-ADA").[7] For example, the configuration OptiMathSAT-LIN-IN denotes the inline linear-search procedure. We used only five configurations since the "-ADA-OF" were not implemented.

Due to the absence of competitors on OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), we evaluate the performance of our five configurations of OptiMathSAT by comparing them against the commercial LGDP tool GAMS[8] v23.7.1 [Brooke et al. 2011] on OMT($\mathcal{LA}(\mathbb{Q})$) problems. GAMS is a tool for modeling and solving optimization problems, consisting of different language compilers, which translate mathematical problems into representations required by specific solvers, like Cplex [IBM 2010]. GAMS provides two reformulation tools, LogMIP[9] v2.0 and JAMS[10] (a new version of the EMP[11] solver), such that both of them allow for reformulating LGDP models by using either BM or CH methods [Raman and Grossmann 1994; Sawaya and Grossmann 2012]. We use Cplex v12.2 [IBM 2010] (through an OSI/Cplex link) to solve the reformulated MILP models. All the tools were executed using default options, as suggested by the authors [Vecchietti 2011]. We also compared OptiMathSAT against MathSAT augmented by PB optimization [Cimatti et al. 2010] (we call it PB-MathSAT) on MaxSMT problems.

Remark 5.1. Importantly, MathSAT and OptiMathSAT use *infinite-precision arithmetic*, while the GAMS tools and Cplex implement standard *floating-point arithmetic*. Moreover, the former handle strict inequalities natively (see Section 2.2), while the GAMS tools use an approximation with a very-small constant value "eps" ϵ (default $\epsilon \stackrel{\text{def}}{=} 10^{-6}$), so that, for example, "$(x > 0)$ *is internally rewritten into* $(x \geq 10^{-6})$."[12]

The comparison is run on four distinct collections of benchmark problems:

—(Section 5.2) LGDP problems, proposed by LogMIP and JAMS authors [Vecchietti and Grossmann 2004; Sawaya and Grossmann 2005, 2012];
—(Section 5.3) OMT($\mathcal{LA}(\mathbb{Q})$) problems from SMT-LIB;[13]
—(Section 5.4) OMT($\mathcal{LA}(\mathbb{Q})$) problems, coming from encoding parametric verification problems from the SAL[14] model checker;
—(Section 5.5) the MaxSMT problems from Cimatti et al. [2010].

The encodings from LGDP to OMT($\mathcal{LA}(\mathbb{Q})$) and back are described in Section 5.1.

All tests were executed on two identical 2.66GHz Xeon machines with 4GB RAM running Linux, using a timeout of 600s for each run. In order to have a reliable and fair measurement of CPU time, we have run only one process per PC at a time. Overall, the evaluation consisted in $\approx$40,000 solver runs, for a total CPU time of up to 276 CPU days.

The correctness of the minimum costs min found by OptiMathSAT have been cross-checked by another SMT solver, Yices,[15] by checking the inconsistency within the bounds of $\varphi \wedge (\mathsf{cost} < \mathsf{min})$ and the consistency of $\varphi \wedge (\mathsf{cost} = \mathsf{min})$ (if min is nonstrict), or of $\varphi \wedge (\mathsf{cost} \leq \mathsf{min})$ and $\varphi \wedge (\mathsf{cost} = \mathsf{min} + \epsilon)$ (if min is strict), ϵ being some very small value.

[7]Here "-LIN" means that BinSearchMode() always returns false, "-BIN" denotes the mixed linear-binary strategy described in Section 4.1 to ensure termination, while "-ADA" refers to the adaptive strategy illustrated in Section 4.2.
[8]http://www.gams.com.
[9]http://www.logmip.ceride.gov.ar/index.html.
[10]http://www.gams.com/.
[11]http://www.gams.com/dd/docs/solvers/emp.pdf.
[12]GAMS support team, email personal communication, 2012.
[13]http://www.smtlib.org/.
[14]http://sal.csl.sri.com.
[15]http://yices.csl.sri.com/.

All versions of OPTIMATHSAT passed the previously mentioned checks. On the LGDP problems (Section 5.2) all tools agreed on the final results, apart from tiny rounding errors by GAMS tools;[16] on all the other problem collections (Sections 5.3, 5.4, and 5.5) instead, the results of the GAMS tools were affected by errors, which we will discuss there.

In order to make the experiments reproducible, more detailed tables, the full-size plots, the Linux binary of OPTIMATHSAT used in this article, the problems, and the results are made available at the OPTIMATHSAT web page.[17] (We cannot distribute the GAMS tools since they are subject to licencing restrictions (see Brooke et al. [2011]); however, they can be obtained at GAMS url.)

5.1. Encodings

In order to translate LGDP models into OMT($\mathcal{LA}(\mathbb{Q})$) problems, we use the encoding in (10) of Section 3.3, namely, LGDP2SMT. Notice that LGDP models are written in GAMS language, which provides a large number of constructs. Since our encoder supports only base constructs (like equations and disjunctions), before generating the LGDP2SMT encoding, we used the GAMS Converter tool for converting complex GAMS specifications (e.g., containing sets and indexed equations) into simpler specifications. Notice also that in the GAMS language the disjunction of constraints in (10) must be described as nested if-then-elses on the Boolean propositions Y^{jk}, so as to avoid the need of including explicitly in ϕ the "xor" constraints discussed in the explanation of (10). Our encodings in both directions comply with this fact.

In order to translate OMT($\mathcal{LA}(\mathbb{Q})$) problems into LGDP models, we consider two different encodings, namely, SMT2LGDP$_1$ and SMT2LGDP$_2$.

Since GAMS tools do not handle negated equalities and strict inequalities, with both encodings negated equalities $\neg(t_1 = t_2)$ or $(t_1 \neq t_2)$ in the input $\mathcal{LA}(\mathbb{Q})$ formula φ are first replaced by the disjunction of two inequalities $\neg(t_1 \leq t_2) \vee \neg(t_1 \geq t_2)$) and strict inequalities $(t_1 < t_2)$ are rewritten as negated nonstrict inequalities $\neg(t_1 \geq t_2)$.[18] Let φ' be the $\mathcal{LA}(\mathbb{Q})$ formula obtained by φ after these substitutions.

In SMT2LGDP$_1$, which is inspired to the polarity-driven CNF conversion of Plaisted and Greenbaum [1986], we compute the Boolean abstraction φ'^p of φ' (which plays the role of formula ϕ in (4)) and then, for each $\mathcal{LA}$ atom ψ_i occurring positively (negatively, respectively) in φ', we add the disjunction $\neg A_i \vee \psi_i$ ($A_i \vee \neg\psi_i$, respectively), where A_i is the Boolean atom of φ'^p corresponding to the $\mathcal{LA}$ atom ψ_i.

In SMT2LGDP$_2$, first we compute the CNF-ization of φ' using the MATHSAT5 CNF-izer, and then we encode each nonunit clause $(l_{i1} \vee \ldots \vee l_{in}) \in \varphi'$ as a LGDP disjunction $[Y_i^1 \wedge l_{i1}] \vee \cdots \vee [Y_i^n \wedge l_{in}]$, where $Y_i^1, \ldots, Y_i^n$ are fresh Boolean variables.

Remark 5.2. We decided to provide two different encodings for several reasons. SMT2LGDP$_1$ is a straightforward and very-natural encoding. However, we have verified empirically, and some discussion with GAMS support team confirmed it,[19] that some GAMS tools/options often have problems in efficiently and even correctly handling the Boolean structure of the formulas ϕ in (4) (see, e.g., the number of problems terminated with error messages in Sections 5.3– 5.5). Thus, following also the suggestions of the GAMS support team, we have introduced SMT2LGDP$_2$, which eliminates any Boolean

[16]GAMS +CPLEX often gives some errors $\leq 10^{-5}$, which we believe are due to the printing floating-point format: For example, while OPTIMATHSAT reports the value `7728125177/2500000000` with infinite-precision arithmetic, GAMS +CPLEX reports it as its floating-point approximation `3.091250e+00`.

[17]http://optimathsat.disi.unitn.it.

[18]Here, we implicitly assume that the literals $\neg(t_1 = t_2)$, $(t_1 \neq t_2)$, and $(t_1 < t_2)$ occur *positively* in φ; for negative occurrences the encoding is dual.

[19]GAMS support team, email personal communication, 2012.

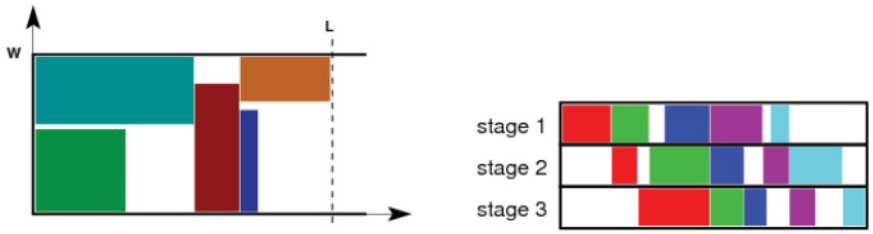

Fig. 2. Graphical representation of the strip-packing (left) and of a zero-walt job shop problem (right).

structure, reducing the encoding substantially to a set of LGDP disjunctions. Notice, however, that SMT2LGDP$_2$ benefits from the CNF encoder of MATHSAT5.

5.2. Comparison on LGDP problems

We have performed the first comparison over two distinct benchmarks, *strip-packing* and *zero-wait job-shop scheduling* problems, which have been previously proposed as benchmarks for LOGMIP and JAMS by their authors [Vecchietti and Grossmann 2004; Sawaya and Grossmann 2005, 2012]. We adopted the encoding of the problems into LGDP given by the authors[20] and gave a corresponding OMT($\mathcal{LA}(\mathbb{Q})$) encoding. We refer to them as "directly generated" benchmarks.

In order to make the results independent from the encoding used, to investigate the correctness and effectiveness of the encodings described in Section 5.1, and to check the robustness of the tools with respect to different encodings, we also generated formulas from "directly generated" benchmarks by applying the encodings SMT2LGDP$_1$, SMT2LGDP$_2$, and LGDP2SMT; we also applied the SMT2LGDP$_1$/SMT2LGDP$_2$ and LGDP2SMT encodings consecutively to SMT formulas. We refer to them as "encoded" benchmarks.

5.2.1. The Strip-Packing Problem. Given a set of N rectangles of different length L_i and height H_i, $i \in 1, \ldots, N$, and a strip of fixed width W but unlimited length, the *strip-packing* problem aims at minimizing the length L of the filled part of the strip while filling the strip with all rectangles, without any overlap and any rotation (see Figure 2, left).

The LGDP model provided by Sawaya and Grossmann [2005] is the following:

$$\begin{array}{rcl}
\min & L, & \\
\text{such that} & L \geq x_i + L_i & \forall i \in N, \\
& \left[\begin{array}{c} Y^1_{ij} \\ x_i + L_i \leq x_j \end{array}\right] \vee \left[\begin{array}{c} Y^2_{ij} \\ x_j + L_j \leq x_i \end{array}\right], & \\
& \vee \left[\begin{array}{c} Y^3_{ij} \\ y_i - H_i \geq y_j \end{array}\right] \vee \left[\begin{array}{c} Y^4_{ij} \\ y_j - H_j \geq y_i \end{array}\right] & \forall i, j \in N, i < j, \\
& x_i \leq \mathsf{ub} - L_i & \forall i \in N. \\
& H_i \leq y_i \leq W & \forall i \in N, \\
& L, x_i, y_i \in \mathbb{R}^1_+, Y^1_{ij}, Y^2_{ij}, Y^3_{ij}, Y^4_{ij} \in \{True, False\}, &
\end{array} \tag{14}$$

where L corresponds to the objective function to minimize and every rectangle $j \in J$ is represented by the constants L_j and H_j (length and height, respectively) and the variables x_j, y_j (the coordinates of the upper left corner in the two-dimensional space).

[20]Examples are available at http://www.logmip.ceride.gov.ar/newer.html and at http://www.gams.com/modlib/modlib.htm.

Every pair of rectangles $i, j \in N, i < j$ is constrained by a disjunction that avoids their overlapping (each disjunct represents the position of rectangle i in relation to rectangle j). The size of the strip limits the position of each rectangle j: The width of the strip W and the upper bound ub on the optimal solution bound the y_j coordinate and the height H_j bounds the x_j coordinate. We express straightforwardly the LGDP model (14) into OMT($\mathcal{LA}(\mathbb{Q})$) as follows:

$$\begin{aligned}\varphi \stackrel{\text{def}}{=}\ & (\mathsf{cost} = L) \wedge \textstyle\bigwedge_{i\in N}(L \geq x_i + L_i)\\ \wedge\ & \textstyle\bigwedge_{i,j\in N, i<j}((x_i + L_i \leq x_j) \vee (x_j + L_j \leq x_i)\\ \vee\ & (y_i - H_i \geq y_j) \vee (y_j - H_j \geq y_i))\\ \wedge\ & \textstyle\bigwedge_{i\in N}(x_i \leq \mathsf{ub} - L_i) \wedge \bigwedge_{i\in N}(x_i \geq 0)\\ \wedge\ & \textstyle\bigwedge_{i\in N}(H_i \leq y_i) \wedge \bigwedge_{i\in N}(W \geq y_i) \wedge \bigwedge_{i\in N}(y_i \geq 0).\end{aligned} \tag{15}$$

We randomly generated instances of the strip-packing problem according to a fixed width W of the strip and a fixed number of rectangles N. For each rectangle $j \in N$, length L_j and height H_j are selected in the interval $]0, 1]$ uniformly at random. The upper bound ub is computed with the same heuristic used by Sawaya and Grossmann [2005], which sorts the rectangles in nonincreasing order of width and fills the strip by placing each rectangle in the bottom-left corner, and the lower bound lb is set to zero. We generated 100 samples each for 9, 10, and 11 rectangles and for two values of the width $\sqrt{N}/2$ and 1. (Notice that with $W = \sqrt{N}/2$ the filled strip looks approximatively like a square, while $W = 1$ is half the average size of one rectangle.)

5.2.2. The Zero-Wait Job-Shop Problem. Consider the scenario where there is a set I of jobs that must be scheduled sequentially on a set J of consecutive stages with zero-wait transfer between them. Each job $i \in I$ has a start time s_i and a processing time t_{ij} in the stage $j \in J_i$, J_i being the set of stages of job i. The goal of the *zero-wait job-shop scheduling* problem is to minimize the makespan, which is the total length of the schedule (see Figure 2, right).

The LGDP model provided by Sawaya and Grossmann [2005] is

$$\begin{aligned}\min\quad & M,\\ \text{such that}\quad & M \geq \textstyle\sum_{j\in J_i} t_{ij} && \forall i \in I,\\ & \begin{bmatrix} Y^1_{ik}\\ s_i + \sum_{m\in J_i, m\leq j} t_{im} \leq s_k + \sum_{m\in J_k, m<j} t_{km}\end{bmatrix} &&\\ & \vee \begin{bmatrix} Y^2_{ik}\\ s_k + \sum_{m\in J_k, m\leq j} t_{km} \leq s_i + \sum_{m\in J_i, m<j} t_{im}\end{bmatrix} && \forall j \in C_{ik}, \forall i, k \in I, i < k,\\ & M, s_i \in \mathbb{R}^1_+, Y^1_{ik}, Y^2_{ik} \in \{True, False\} && \forall i, k \in I, i < k,\end{aligned} \tag{16}$$

where M corresponds to the objective function to minimize and every job $i \in I$ is represented by the variable s_i (its start time) and the constant t_{ij} (its processing time in stage $j \in J_i$). For each pair of jobs $i, k \in I$ and for each stage j with potential clashes (i.e., $j \in C_{ik} = \{J_i \cap J_k\}$), a disjunction ensures that no clash between jobs occurs at any stage at the same time. We encoded the corresponding LGDP model (16) into OMT($\mathcal{LA}(\mathbb{Q})$) as follows:

$$\begin{aligned}\varphi \stackrel{\text{def}}{=}\ & (\mathsf{cost} = M) \wedge \textstyle\bigwedge_{i\in I}\left(M \geq s_i + \sum_{j\in J_i} t_{ij}\right) \wedge \bigwedge_{i\in I}(s_i \geq 0)\\ & \wedge \textstyle\bigwedge_{j\in C_{ik}, \forall i,k\in I, i<k}\Big((s_i + \sum_{m\in J_i, m\leq j} t_{im} \leq s_k + \sum_{m\in J_k, m<j} t_{km})\\ & \vee (s_k + \textstyle\sum_{m\in J_k, m\leq j} t_{km} \leq s_i + \sum_{m\in J_i, m<j} t_{im})\Big).\end{aligned} \tag{17}$$

Procedure	Strip-packing													
	$W = \sqrt{N}/2$						$W = 1$						Total	
	$N = 9$		$N = 12$		$N = 15$		$N = 9$		$N = 12$		$N = 15$			
	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time
Directly Generated Benchmarks														
OM-LIN-OF	100	53	100	605	94	8160	100	749	89	3869	54	5547	537	18983
OM-LIN-IN	100	12	100	144	100	3518	100	173	94	2127	74	6808	**568**	**12782**
OM-BIN-OF	100	50	100	625	89	8346	100	588	89	5253	45	5611	523	20473
OM-BIN-IN	100	14	100	211	98	4880	100	202	94	2985	65	8101	557	16393
OM-ADA-IN	100	13	100	192	99	5574	100	214	94	2675	63	7949	556	16617
JAMS(BM)	100	230	78	10177	12	1180	100	158	91	3878	51	6695	**432**	**22318**
JAMS(CH)	100	2854	27	2393	1	417	100	1906	70	7471	17	4032	315	19073
LogMIP(BM)	100	229	78	10159	12	1192	100	157	91	3866	51	6720	**432**	**22323**
LogMIP(CH)	100	2851	27	2414	1	424	100	1907	70	7440	17	4037	315	19073
LGDP2SMT Encoded Benchmarks														
OM-LIN-IN	100	12	100	144	100	3563	100	183	94	2169	73	6466	567	12537
SMT2LGDP$_1$-LGDP2SMT Encoded Benchmarks														
OM-LIN-IN	100	13	100	166	100	5919	100	195	94	2156	74	7080	568	15529
SMT2LGDP$_2$-LGDP2SMT Encoded Benchmarks														
OM-LIN-IN	100	13	100	141	100	5574	100	172	94	2148	74	6650	568	12618
SMT2LGDP$_1$ Encoded Benchmarks														
JAMS(BM)	100	389	68	8733	12	1934	100	162	89	5565	47	7313	**416**	**24096**
JAMS(CH)	99	980	46	6099	2	769	100	726	72	7454	17	3505	336	19533
LogMIP(BM)	100	390	68	8723	12	1946	100	163	89	5547	47	7299	**416**	**24068**
LogMIP(CH)	99	981	54	5480	12	735	100	725	74	7433	17	3542	346	18896
SMT2LGDP$_2$ Encoded Benchmarks														
JAMS(BM)	100	190	81	8460	11	2066	100	159	89	2960	56	8142	437	21977
JAMS(CH)	98	3799	24	2137	1	292	100	2402	68	7926	16	3429	307	19985
LogMIP(BM)	100	191	81	8462	11	2071	100	159	90	2964	56	8206	**438**	**22053**
LogMIP(CH)	98	3807	24	2133	1	312	100	2388	68	7915	17	4027	308	20582

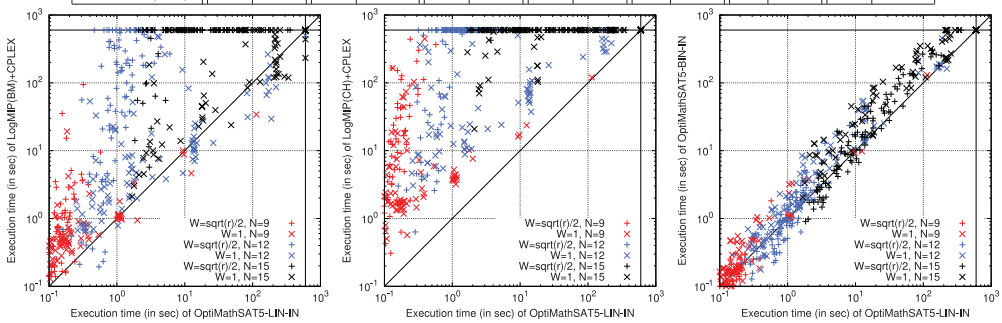

Fig. 3. Table: results (# of solved instances, cumulative time in seconds for solved instances) for OptiMathSAT and GAMS (using LogMIP and JAMS) on 100 random instances (including "directly generated" and "encoded" benchmarks) of each of the strip-packing problems for N rectangles, where $N = 9, 12, 15$, and width $W = \sqrt{N}/2, 1$. (We use "OM" as shortcut for OptiMathSAT and omit "+Cplex" in the labels of GAMS tools.) Values highlighted in **bold** represent best performances. Scatter plots: comparison of the best configuration of OptiMathSAT (OptiMathSAT-LIN-IN) against LogMIP(BM)+CPLEX (left), LogMIP(CH)+CPLEX (center), and OptiMathSAT-BIN-IN (right) on "directly generated" benchmarks.

We randomly generated instances of the zero-wait job-shop problem according to a fixed number of jobs I and a fixed number of stages J. For each job $i \in I$, start time s_i and processing time t_{ij} of every job are selected in the interval $]0, 1]$ uniformly at random. We consider a set of 100 samples each for 9, 10, 11, 12 jobs and 8 stages, and for 11 jobs and 9, 10 stages. We set no value for ub and lb $= 0$.

5.2.3. Discussion. The table of Figure 3 shows the number of solved instances and their cumulative execution time for different configurations of OptiMathSAT and GAMS on "directly generated" and "encoded" benchmarks. The scatter plots of Figure 3 compare the best-performing version of OptiMathSAT, OptiMathSAT-LIN-IN, against LogMIP+CPLEX with BM and CH reformulation (left and center, respectively) and

Procedure	Job-shop													
	$I = 9$, $J = 8$		$I = 10$, $J = 8$		$I = 11$, $J = 8$		$I = 12$, $J = 8$		$I = 11$, $J = 9$		$I = 11$, $J = 10$		Total	
	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time
Directly Generated Benchmarks														
OM5-LIN-OF	100	386	100	1854	97	9396	57	14051	100	9637	99	10670	553	45995
OM5-LIN-IN	100	317	100	1584	100	8100	77	18046	100	7738	100	7433	**577**	**43228**
OM5-BIN-OF	100	726	100	3817	88	13222	38	12529	92	14183	90	13287	508	57764
OM5-BIN-IN	100	602	100	3270	97	12878	54	16234	96	13159	96	12350	543	58493
OM5-ADA-IN	100	596	100	3230	97	12262	53	14810	96	12805	96	12125	542	55828
JAMS(BM)	100	268	100	1113	100	4734	87	17067	100	4941	100	6122	**587**	**34245**
JAMS(CH)	84	23830	4	1596	0	0	0	0	0	0	1	363	89	25789
LogMIP(BM)	100	267	100	1114	100	4718	87	17108	100	4962	100	6174	**587**	**34343**
LogMIP(CH)	84	23871	4	1622	0	0	0	0	0	0	1	338	89	25831
LGDP2SMT Encoded Benchmarks														
OM5-LIN-IN	100	324	100	1571	100	7739	74	16494	100	7175	100	7504	574	40807
SMT2LGDP$_1$-LGDP2SMT Encoded Benchmarks														
OM5-LIN-IN	100	336	100	1578	100	7762	71	16589	100	7726	100	7706	571	41697
SMT2LGDP$_2$-LGDP2SMT Encoded Benchmarks														
OM5-LIN-IN	100	320	100	1533	100	7623	68	15120	100	7216	100	7598	568	39410
SMT2LGDP$_1$ Encoded Benchmarks														
JAMS(BM)	100	239	100	1128	100	5516	84	19949	100	6667	100	4176	**584**	**37675**
JAMS(CH)	100	14527	46	17887	0	0	0	0	1	497	0	0	147	32911
LogMIP(BM)	100	240	100	1122	100	5510	83	19489	100	6684	100	4180	583	37225
LogMIP(CH)	100	14465	47	18206	0	0	0	0	1	495	0	0	148	33166
SMT2LGDP$_2$ Encoded Benchmarks														
JAMS(BM)	100	319	100	1865	100	12470	45	15704	97	13189	96	15773	**538**	**59320**
JAMS(CH)	95	22435	18	8030	2	671	0	0	1	526	3	1043	119	32723
LogMIP(BM)	100	319	100	1871	100	12440	45	15747	98	13661	95	15102	**538**	**59140**
LogMIP(CH)	95	22401	18	7991	1	163	0	0	1	437	3	1020	118	32012

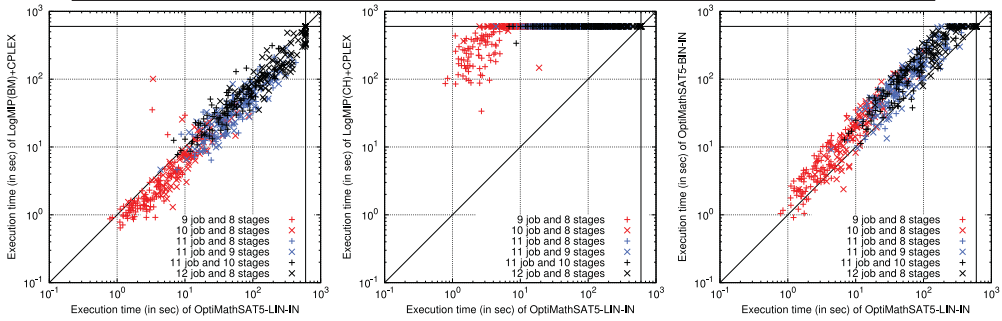

Fig. 4. Table: results (# of solved instances, cumulative time in seconds for solved instances) for OptiMathSAT and GAMS on 100 random samples (including "directly generated" and "encoded" benchmarks) of each of the job-shop problems for $I = 9, 10, 11, 12$ jobs and $J = 8$ stages and for $I = 11$ jobs and $J = 9, 10$ stages. (We use "OM" as a shortcut for OptiMathSAT and omit "+Cplex" in the labels of GAMS tools.) Values highlighted in **bold** represent best performances. Scatter plots: comparison of the best configuration of OptiMathSAT (OptiMathSAT-LIN-IN) against LogMIP(BM)+CPLEX (left), LogMIP(CH)+CPLEX (center), and OptiMathSAT-BIN-IN (right) on "directly generated" benchmarks.

the two inline versions OptiMathSAT-LIN-IN and OptiMathSAT-BIN-IN (right) on "directly generated" benchmarks.

The table of Figure 4 shows the number of solved instances and their cumulative execution time for different configurations of OptiMathSAT and GAMS on "directly generated" and "encoded" benchmarks. The scatter plots of Figure 4 compare, on "directly encoded" benchmarks, the best-performing version of OptiMathSAT, OptiMathSAT-LIN-IN, against LogMIP with BM and CH reformulation (left and center, respectively); the figure also compares the two inline versions OptiMathSAT-LIN-IN and OptiMathSAT-BIN-IN (right).

The results on the LGDP problems in Figures 3 and 4 suggest some considerations.

Comparing the different versions of OptiMathSAT, we notice that

—the inline versions (-IN) behave pairwise uniformly better than the corresponding offline versions (-OF), which is not surprising;
—overall, the -LIN options seem to perform a little better than the corresponding -BIN and -ADA options (although gaps are not dramatic).

Remark 5.3. We notice that with LGDP problems binary search is not "obviously faster" than linear search, in compliance with what is stated in point 6 in Section 4.1. This is further enforced by the fact that in strip-packing (15) (job-shop (17), respectively) encodings, the cost variables $\mathsf{cost} \stackrel{\text{def}}{=} L$ ($\mathsf{cost} \stackrel{\text{def}}{=} M$, respectively) occur only in positive unit clauses in the form ($L \geq \langle term \rangle$) (($M \geq \langle term \rangle$), respectively); thus, learning $\neg(\mathsf{cost} < \mathsf{pivot})$ as a result of the binary-search steps with UNSAT results produces no constraining effect on the variables in $\langle term \rangle$, and hence no substantial extra search-pruning effect due to the early-pruning technique of the SMT solver.

Comparing the different versions of the GAMS tools, we see that LogMIP and JAMS reformulations lead to substantially identical performance on both strip-packing and job-shop instances. For both reformulation tools, the BM versions uniformly outperform the CH ones, often dramatically.

Comparing the performances of the versions of OptiMathSAT against those of the GAMS tools, we notice that

—on *strip-packing problems* all versions of OptiMathSAT outperform all GAMS versions, regardless of the encoding used. For example, the best OptiMathSAT version solved $\approx$30% more formulas than the best GAMS version;
—on *job-shop problems* results are mixed. OptiMathSAT drastically outperforms the CH versions on all encodings and it slightly beats the BM ones on "SMT2LGDP$_2$ encoded" benchmarks, while it is slightly beaten by the BM versions on "directly generated" and "SMT2LGDP$_1$ encoded" benchmarks. For example, the best OptiMathSAT version solved $\approx$2% less formulas than the best GAMS version.

Overall, we can conclude that OptiMathSAT performances on these problems are comparable with, and most often significantly better than, those of GAMS tools.

We may wonder how these results are affected by the different encodings used. (We recall from the beginning of Section 5 that all solvers agreed on the results, regardless of the encoding.) In terms of performances, comparing the effects of the different encodings, we notice the following facts.

—On OptiMathSAT (-LIN-IN) the effects of the different encodings are substantially negligible, on both strip-packing and job-shop problems, since we have only very small variations in the number of solved instances between "directly generated" and "encoded" instances, in the various encoding combinations. From this reason, we conclude that OptiMathSAT is robust with respect to the encodings of these problems.
—On GAMS tools the effects of the different encodings are more relevant, although very heterogeneous: For example, with respect to "directly generated" instances, "SMT2LGDP$_1$ encoded" solved formulas are slightly less with BM options, and up to much more with CH options; "SMT2LGDP$_2$ encoded" solved formulas are slightly more on strip-packing and a little less on job-shop with BM options, and slightly less on strip-packing and much more on job-shop with CH options. For this reason, in the next sections we always report the results with both encodings.

5.2.4. Analysis of OptiMathSAT *Performances.* We want to perform a more fine-grained analysis of the performances of the best version of OptiMathSAT,

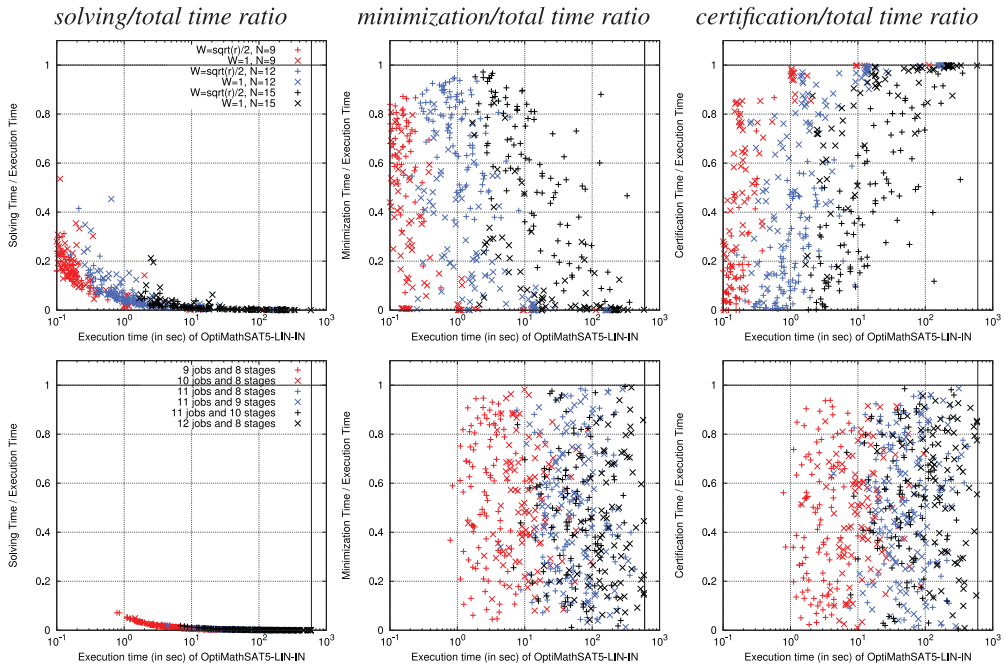

Fig. 5. Scatter plots comparing solving, minimization, and certification time (left, center, and right, respectively) with the execution time of OptiMathSAT-LIN-IN on "directly generated" instances of strip-packing (top) and job-shop (bottom).

OptiMathSAT-LIN-IN. To this extent, we partition the total execution time taken on each problem into three consecutive components:

—*solving time*, that is, the time spent on finding the first suboptimal solution;
—*minimization time*, that is, the time required to search for the optimal solution; and
—*certification time*, that is, the time needed for checking there is no better solution.

Figure 5 reports, for all strip-packing (top) and job-shop (bottom) instances, the ratios of the three previously mentioned components over total execution time. (Notice the log scale of the x axis and the linear scale on the y axis.) We notice a few facts:

—the solving time is nearly negligible; in particular, on hardest problems. This tells us, among other facts, that OMT($\mathcal{LA}(\mathbb{Q})$) on these formulas is a much harder problem than plain SMT($\mathcal{LA}(\mathbb{Q})$) on the same formulas;
—the remaining time, on average, is either evenly shared between the minimization and the certification efforts (job-shop, bottom), or even it is mostly dominated by the latter; in particular, on the hardest problem (strip-packing, top).

Overall, this suggests that on these instances OptiMathSAT-LIN-IN takes on average less than half of the total execution time to find the actual optimal solution, and more than half to prove that there is no better one.

5.3. Comparison on SMT-LIB Problems

As a second comparison, in Figure 6, we compare OptiMathSAT against the GAMS tools on the satisfiable $\mathcal{LA}(\mathbb{Q})$ formulas (QF_LRA) in the SMT-LIB, augmented with randomly selected costs. (Hereafter, we do not consider the -OF versions of OptiMathSAT.) These instances are all classified as "industrial," because they come from the

Procedure	SMT-LIB/QF_LRA formulas						
	#inst.	#term.	#correct.	#err.msg.	#wrong	#unfeas.	time
OPTIMATHSAT5-LIN-IN	194	194	194	0	0	0	1604
OPTIMATHSAT5-BIN-IN	194	194	**194**	0	0	0	**1449**
OPTIMATHSAT5-ADA-IN	194	194	194	0	0	0	1618
LDGP-SMT-Encoded Benchmarks (SMT2LGDP$_1$-LGDP2SMT)							
OPTIMATHSAT5-LIN-IN	194	194	194	0	0	0	1820
LDGP-SMT-Encoded Benchmarks (SMT2LGDP$_2$-LGDP2SMT)							
OPTIMATHSAT5-LIN-IN	194	194	194	0	0	0	1597
LGDP-Encoded Benchmarks (SMT2LGDP$_1$)							
JAMS(BM)+CPLEX	194	171	116	52	0	3	1561
JAMS(CH)+CPLEX	194	193	15	108	0	70	559
LOGMIP(BM)+CPLEX	194	172	**117**	52	0	3	**2152**
LOGMIP(CH)+CPLEX	194	193	15	108	0	70	576
LGDP-Encoded Benchmarks (SMT2LGDP$_2$)							
JAMS(BM)+CPLEX	194	172	**166**	0	4	2	**6839**
JAMS(CH)+CPLEX	194	105	104	0	0	1	9912
LOGMIP(BM)+CPLEX	194	171	165	0	4	2	4103
LOGMIP(CH)+CPLEX	194	105	104	0	0	1	9649

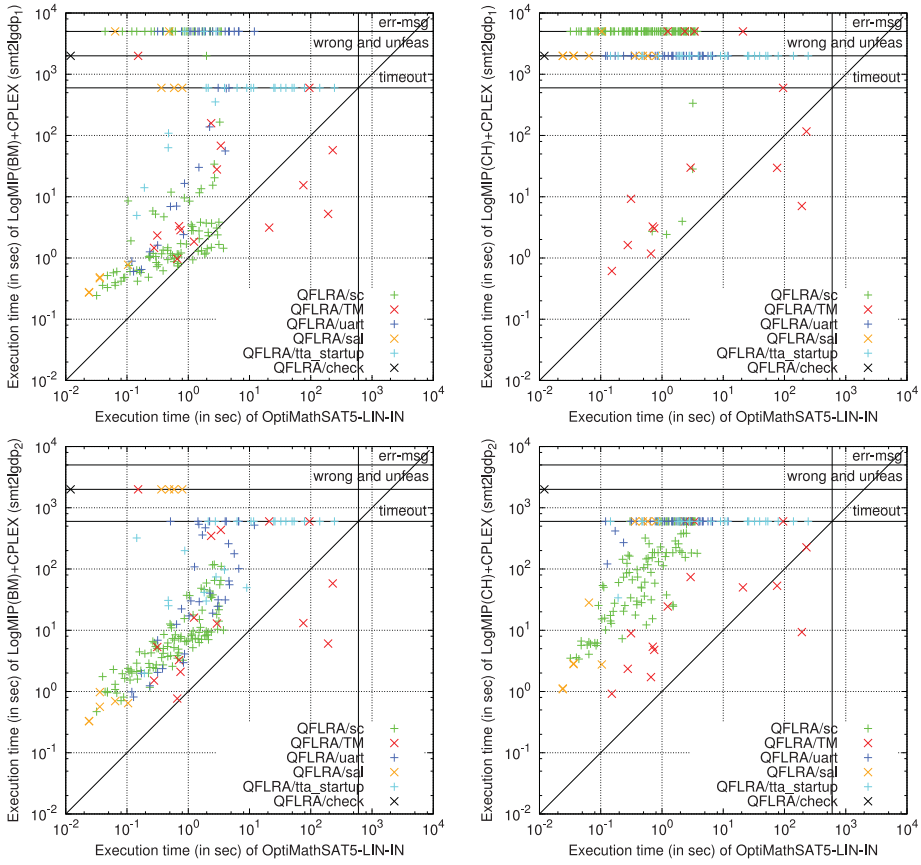

Fig. 6. Table: results for all the inline versions of OPTIMATHSAT and all the GAMS tools, on a subset of SMT-LIB $\mathcal{LA}(\mathbb{Q})$ satisfiable instances. The columns report, respectively, # of instances considered, # of instances terminating within the timeout, # of instances terminating with correct solution, # of instances terminating with error messages, # of instances terminating returning a wrong minimum, and # of instances terminating wrongly returning "unfeasible." Scatter plots: pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable instances between OPTIMATHSAT-LIN-IN and the two versions of LOGMIP+CPLEX. LGDP models are generated using SMT2LGDP$_1$ (top) and SMT2LGDP$_2$ (bottom).

encoding of different real-world problems in formal verification, planning, and optimization. They are divided into six categories, namely, `sc`, `uart`, `sal`, `TM`, `tta_startup`, and `miplib`. Notice that other SMT-LIB categories like `spider_benchmarks` and `clock_synchro` do not contain satisfiable instances and are thus not reported here.

Since we have no control on the origin of each problem and on the name and meaning of the variables, we selected iteratively one variable at random as cost variable, dropping it if the resulting minimum was $-\infty$. This forced us to eliminate a few instances; in particular, all `miplib` ones. We used both SMT2LGDP$_1$ and SMT2LGDP$_2$ to encode these problems into LGDP.

As before, to check for both correctness and effectiveness of the encodings, we also encoded the problems into LGDP by each encoding and encoded then back, to be fed to OPTIMATHSAT-LIN-IN (fourth and fifth rows). We notice that this caused a substantial difference in neither correctness nor efficiency.

We notice first that the results for GAMS tools are affected by correctness problems, with both encodings. Consider the encoding SMT2LGDP$_1$. Out of 194 samples, both GAMS tools with the CH option returned "unfeasible" (i.e., inconsistent) on 70 samples and an error message (regarding some unsatisfied disjunctions) on 108 samples. The two versions with BM returned three unfeasible solutions and 52 solutions with error messages. Only 15 samples were solved correctly by GAMS tools with the CH option and 117 (with LOGMIP) or 116 (with JAMS) samples with BM ones, while OPTIMATHSAT solved correctly all 194 samples. (We recall that all OPTIMATHSAT results were cross-checked, and that the four GAMS tools were fed with the same files.) With SMT2LGDP$_2$ encoding the number of correctly solved formulas increases, 104 with CH option and 165 (with LOGMIP) or 166 (with JAMS) with BM; there are no error messages and the number of unfeasible solutions of both GAMS tools with the BM and CH options decreases to 2 and 1 respectively, but the number of solutions with wrong minimum increases to four with the BM versions.

Importantly, with both encodings, the results for GAMS tools varied by modifying a couple of parameters from their default value, namely, "`eps`" and "`bigM Mvalue`." For example, on the previously mentioned `sal` instance with SMT2LGDP$_1$, with the default values the BM versions returned a wrong minimum value "0," the CH versions returned "unfeasible," while OPTIMATHSAT returned the correct minimum value "2"; modifying `eps` and `bigM Mvalue`, the results become unfeasible also with BM options. This highlights the fact that there are indeed some correctness and robustness problems with the GAMS tools, regardless of the encodings used.[21]

5.3.1. Discussion. We conjecture that the problems with the GAMS tools may be caused, at least in part, by the fact that GAMS tools use floating-point rather than infinite-precision arithmetic, and they introduce internally an approximated representation of strict inequalities (see Remark 5.1). Notice that, unlike with the LGDP problems in Section 5.2, SMT-LIB problems do contain occurrences of strict (nonstrict, respectively) inequalities with positive (negative, respectively) polarity.

From the perspective of the efficiency, all versions of OPTIMATHSAT solved correctly all problems within the timeout, the -BIN-IN version performing slightly better than the others; GAMS did not solve many samples (because of timeout, wrong solutions, and solutions with error messages). Looking at the scatter plots, we notice that, with the exception of a few samples, OPTIMATHSAT always outperforms the GAMS tools, often

[21]We also isolated a subproblem, small enough to be solved by hand, in which the GAMS tools returned evidently wrong results, and notified it to the GAMS support team, who reckoned the problem and promised to investigate it eventually.

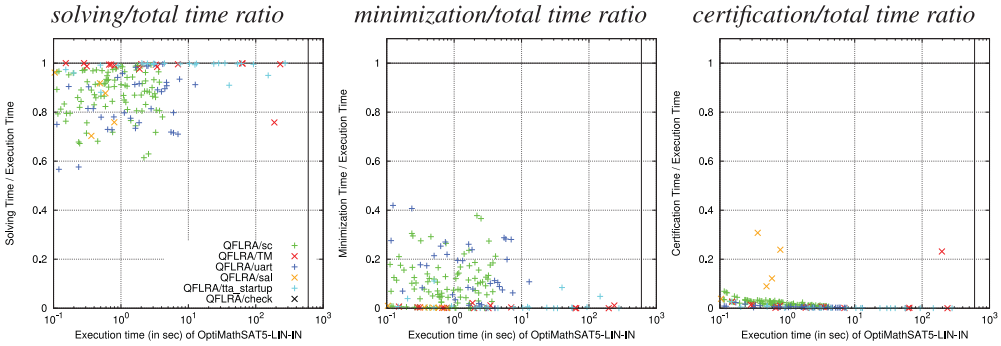

Fig. 7. Scatter plots comparing solving, minimization, and certification time (left, center, and right, respectively) with the execution time of OptiMathSAT-LIN-IN on SMT-LIB instances.

by more than one order of magnitude. We notice that on these problems SMT2LGDP$_2$ is generally more effective than SMT2LGDP$_1$ and less prone to errors.

Finally, Figure 7 reports for all SMT-LIB instances the ratios of the solution, minimization, and certification times over the total execution time for OptiMathSAT-LIN-IN. Unlike with Figure 5, we notice that here the solution time is dominating, the minimization time is significant, and the certification time is nearly negligible. This means that on these instances OptiMathSAT-LIN-IN takes on average more than half of its execution time to find the first solution, less than half to find the actual optimal solution, and very little time to prove that there is no better one. We conjecture that this is due to the fact that most satisfiable SMT-LIB instances come from the encoding of formal verification steps of bugged systems which, unlike with the LGDP problems of Section 5.2, have a limited number of solutions.

5.4. Comparison on SAL Problems

As a third comparison, in Figure 8, we compare OptiMathSAT against the GAMS tools on $\mathcal{LA}(\mathbb{Q})$ formulas obtained by using the SAL Model Checker on a set of bounded verification problems—Bounded Model Checking (BMC) of invariants [Biere et al. 1999] and K-Induction (K-IND) [Sheeran et al. 2000]—of a well-known parametric timed system, Fisher's Protocol.[22]

BMC (K-IND, respectively) takes a finite-state machine M, an invariant property Ψ, and an integer bound k, and produces a propositional formula φ, which is satisfiable (unsatisfiable, respectively) if and only if there exists a k-step execution violating Ψ (a k-step induction proof that Ψ is always verified, respectively). The approach leverages to real-time systems by producing SMT($\mathcal{LA}(\mathbb{Q})$) formulas rather than purely propositional ones (see, e.g., Audemard et al. [2002]).

Fisher's protocol ensures mutual exclusion among N processes using real-time clocks and a shared variable. The problem is parametric into two positive real values, δ_1 and δ_2, describing the delays of some actions. It is known that mutual exclusion, and other properties included in the SAL model, are verified if and only if $\delta_1 < \delta_2$.

We have produced our OMT($\mathcal{LA}(\mathbb{Q})$) problems as follows. We fixed the value of δ_2 (we chose $\delta_2 = 4$), and then we generated six groups of formulas according to the problem solved (BMC or K-IND) and the property addressed (called `mutex`, `mutual-exclusion`, `time-aux3`, and `logical-aux1`). For each group, for increasing values of $N \geq 2$ and for

[22]Problems available at http://sal.csl.sri.com/examples.shtml.

Procedure	SAL formulas						
	#inst.	#term.	#correct	#err. msg.	#wrong	#unfeas.	time
OPTIMATHSAT5-LIN-IN	392	385	**385**	0	0	0	**44129**
OPTIMATHSAT5-BIN-IN	392	382	382	0	0	0	45869
OPTIMATHSAT5-ADA-IN	392	381	381	0	0	0	44932
LGDP-Encoded Benchmarks (SMT2LGDP$_1$)							
JAMS(BM)+CPLEX	392	24	**4**	19	1	0	**1096**
JAMS(CH)+CPLEX	392	46	0	0	0	46	0
LOGMIP(BM)+CPLEX	392	24	**4**	19	1	0	**1092**
LOGMIP(CH)+CPLEX	392	46	0	0	0	46	0
LGDP-Encoded Benchmarks (SMT2LGDP$_2$)							
JAMS(BM)+CPLEX	392	28	**14**	0	14	0	**1456**
JAMS(CH)+CPLEX	392	31	2	0	0	29	122
LOGMIP(BM)+CPLEX	392	28	**14**	0	14	0	**1428**
LOGMIP(CH)+CPLEX	392	31	2	0	0	29	120

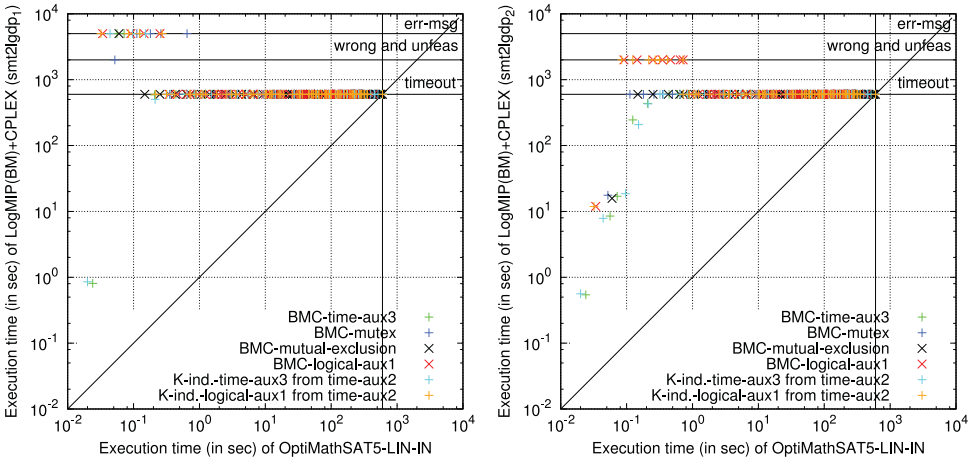

Fig. 8. Table: results for all the inline versions of OPTIMATHSAT and all the GAMS tools, on formulas generated from SAL models of Fisher's protocol. The columns report, respectively, # of instances considered, # of instances terminating within the timeout, # of instances terminating with correct solution, # of instances terminating with error messages (GAMS tools only), # of instances terminating returning a wrong minimum, and # of instances terminating wrongly returning "unfeasible". Scatter plots: comparison of the best configuration of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX on SMT2LGDP$_1$ and SMT2LGDP$_2$ encodings (left and right, respectively).

a set of sufficiently big values of $k \geq k^*$,[23] we used SAL to produce the corresponding parametric SMT($\mathcal{LA}(\mathbb{Q})$) formulas, and asked the tool under test to find the minimum value of δ_1 that made the resulting formula $\mathcal{LA}(\mathbb{Q})$-satisfiable (we knew in advance from the problem that, for k big enough, this value is $\delta_1 = \delta_2 = 4.0$). As before, we used both SMT2LGDP$_1$ and SMT2LGDP$_2$ to encode the OMT($\mathcal{LA}(\mathbb{Q})$) benchmarks into LGDP.

5.4.1. Discussion. The results are presented in Figure 8. The three versions of OPTIMATHSAT solved correctly 385, 382, and 381 out of the 392 samples, respectively, OPTIMATHSAT-LIN-IN being the best performer.

Considering the GAMS tools with the encoding SMT2LGDP$_1$, the two tools using BM solved on time and correctly only four samples over 392 and returned 19 solutions

[23]For BMC, k^* is set to the smallest value of k, which makes the formula satisfiable, imposing no upper bound on δ_1; for K-IND, k^* is set to the smallest value of k, which makes the formula encoding the inductive step unsatisfiable, imposing $\delta_2 > \delta_1$). In these experiments, k^* ranges from 5 to 10, depending on the problem; also, for each problem, k^* does not depend on N.

Procedure	MaxSMT / SMT+PB problems						
	#inst.	#term.	#correct.	#err.msg.	#wrong	#unfeas.	time
PB-MATHSAT-LIN	675	636	**636**	0	0	0	**19675**
PB-MATHSAT-BIN	675	632	632	0	0	0	13024
OMT($\mathcal{LA}(\mathbb{Q})$)-Encoded Benchmarks							
OPTIMATHSAT-LIN-IN	675	630	630	0	0	0	20744
OPTIMATHSAT-BIN-IN	675	634	634	0	0	0	16502
OPTIMATHSAT-ADA-IN	675	637	**637**	0	0	0	**18588**
LGDP-Encoded Benchmarks (SMT2LGDP$_1$)							
JAMS(BM)+CPLEX	675	509	19	423	68	8	420
JAMS(CH)+CPLEX	675	642	0	233	41	377	0
LOGMIP(BM)+CPLEX	675	510	**19**	424	68	8	**403**
LOGMIP(CH)+CPLEX	675	642	0	233	41	377	0
LGDP-Encoded Benchmarks (SMT2LGDP$_2$)							
JAMS(BM)+CPLEX	675	449	**92**	9	351	6	**1575**
JAMS(CH)+CPLEX	675	386	48	9	336	2	644
LOGMIP(BM)+CPLEX	675	449	92	9	351	6	1650
LOGMIP(CH)+CPLEX	675	383	48	9	333	2	674

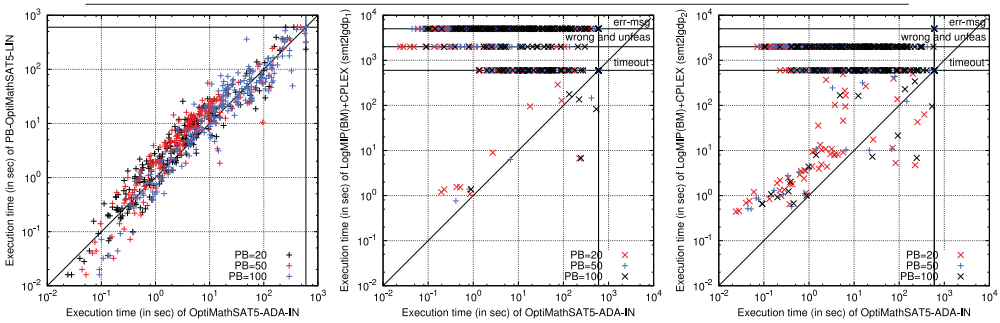

Fig. 9. Table: results for OPTIMATHSAT, PB-MATHSAT, and the GAMS tools, on the MaxSMT benchmarks from Cimatti et al. [2010]. The columns report, respectively, # of instances considered, # of instances terminating within the timeout, # of instances terminating with correct solution, # of instances terminating with error messages (GAMS tools only), # of instances terminating returning a wrong minimum, and # of instances terminating wrongly returning "unfeasible." Scatter plots: comparison of the best configuration of OPTIMATHSAT, OPTIMATHSAT-ADA-IN, against the best configuration of PB-MATHSAT, PB-MATHSAT-LIN (left) and the best configuration of GAMS tools, LOGMIP(BM)+CPLEX, on SMT2LGDP$_1$ and SMT2LGDP$_2$ encodings (center and right, respectively).

with error messages and 1 solution with wrong minimum, while the CH ones always returned "unfeasible." (We recall that all GAMS tools and options are fed the same inputs.) Considering the encoding SMT2LGDP$_2$, the GAMS tools solved more problems correctly (14 with BM tools and 2 with CH), but they returned wrong and unfeasible solutions (14 wrong solutions for BM versions and 29 unfeasible for CH ones). No solution with error messages was found.

The scatter plots compare OPTIMATHSAT-LIN-IN with the best versions of GAMS, LOGMIP(BM)+CPLEX, on both the encodings, showing that the former dramatically outperforms the latter, no matter the encoding used.

5.5. Comparison on PB SMT Problems

As a fourth comparison, in Figure 9, we evaluate OPTIMATHSAT on the problem sets used in Cimatti et al. [2010] against the usual GAMS tools and against a recent reimplementation on MATHSAT5 of the tool in Cimatti et al. [2010], namely, PB-MATHSAT,

for SMT with PB constraints (see Section 3).[24] PB-MathSAT is tested with both linear search and binary search strategies (denoted with "-LIN" and "-BIN," respectively).

As described in Cimatti et al. [2010], the problems consist of partial weighted MaxSMT problems that are generated randomly starting from satisfiable $\mathcal{LA}(\mathbb{Q})$ formulas (QF_LRA) in the SMT-LIB, then converted into SMT problems with PB constraints (see (12) in Section 3). These problems are further encoded into OMT($\mathcal{LA}(\mathbb{Q})$) problems by means of the encoding (11) in Section 3, and hence into LGDP problems by means of the usual two encodings.

5.5.1. Discussion. The results are presented in Figure 9. The three versions of OptiMathSAT solved, respectively, 630, 634, and 637 problems out of 675 problems overall, while the two versions PB-MathSAT solved, respectively, 636 and 632. Thus, despite they are both implemented on top of the same SMT solver and PB-MathSAT is specialized for PB constraints, OptiMathSAT performances are analogous to these of the more-specialized tool. The various versions of the GAMS tool perform drastically worse: with SMT2LGDP$_1$ they solve correctly only a very small number of samples (19 with BM tools and even 0 with CH), returning error messages, unfeasible results, or wrong minimum solutions on the remaining set of benchmarks; with SMT2LGDP$_2$ more samples are solved correctly and no error message is produced, but most problems produce a wrong minimum solution.

Remark 5.4. Notice that, unlike with LGDP problems (see Remark 5.3) and in part also with SMT-LIB and SAL problems, with PB problems the cost variables occur in positive unit clauses in the form $(\mathsf{cost} = \langle term \rangle)$; thus, learning $\neg(\mathsf{cost} < \mathsf{pivot})$ as a result of the binary-search steps with UNSAT results produces a constraining effect on the variables in $\langle term \rangle$, and hence a pruning effect in the search due to the early-pruning technique of the SMT solver. This might explain in part the fact that, unlike with previous problems, here binary search performs a little better than linear search.

The scatter plots in Figure 9 compare the best version of OptiMathSAT with those of PB-MathSAT and of the GAMS tools. We see that OptiMathSAT-ADA-IN performances are analogous to these of PB-MathSAT-LIN , and they are drastically superior to these of GAMS tools with both encodings.

As a side note, in Cimatti et al. [2013a], another empirical evaluation is performed on MaxSMT problems—although generated with a slightly different random method from SMT-LIB benchmarks—where OptiMathSAT performs equivalently better than PB-MathSAT and the novel specialized MaxSMT tool presented there. We refer the reader to Cimatti et al. [2013a] for details.

5.6. Comparison against GAMS with Parallel Cplex on All Problem Sets

As is common practice in the SMT literature, in this article we deal with *single-core sequential* procedures. In fact, despite a couple of attempts [Wintersteiger et al. 2009; Kalinnik et al. 2010], the parallelization of SMT-solving procedures is still an open research issue. In particular, MathSAT5, and hence OptiMathSAT, provide no support for parallelization. Thus, although the issue of efficiently parallelizing OMT is potentially a very interesting research topic, it is definitely not in the intended scope of this article.

Unlike with MathSAT5 and OptiMathSAT, however, Cplex provides full support for multiple-core parallel solving. This is an important benefit, since it allows for exploiting

[24]A comparison against the tool in Cimatti et al. [2010] would not be fair, since the latter was based on the older and slower MathSAT4. To witness this fact, a comparison of these two implementations is in Cimatti et al. [2013a].

the multiple-core CPUs of current PCs, reducing the elapsed time when searching for a solution. This gives GAMS a potential advantage with respect to OPTIMATHSAT, which the previous tests could not reveal.

In order to investigate the actual relevance of this potential advantage, we have recently enriched our empirical investigation by running all the tests in Sections 5.2–5.5 also on another GAMS tool, namely, JAMS(BM)+CPLEX-4CORES: This is the best-performing GAMS GAMS tool in the tests, JAMS(BM)+CPLEX, which uses instead the most recent version of CPLEX, v12.6, *in parallel mode on four cores* (options `opportunistic parallelmode, 4 threads`). Each of the four threads is given a timeout of 600s.[25] Therefore, JAMS(BM)+CPLEX-4CORES is given four times the CPU time resources than its competitors.

5.6.1. Discussion. The results are displayed in Figure 10. In the table, for each group of benchmarks in Sections 5.2– 5.5, we compare the performances of the best-performing OPTIMATHSAT tool, OPTIMATHSAT-LIN-IN, of the best-performing GAMS tool, JAMS(BM)+CPLEX, and of its parallel version, JAMS(BM)+CPLEX-4CORES. The results for the former two tools are taken from Figures 3–9. In the last column, we report the mean values of the speedup for JAMS(BM)+CPLEX-4CORES with respect to JAMS(BM)+CPLEX over the problems for which both tools terminated within the timeout. In the scatter plots we compare pairwise the performances of JAMS(BM)+CPLEX-4CORES and JAMS(BM)+CPLEX on the five problem sets.

From Figure 10, we notice the following facts:

—The usage of CPLEX in parallel mode on the four cores pays off in terms of elapsed time: We notice a significant average speedup from JAMS(BM)+CPLEX to JAMS(BM)+CPLEX-4CORES, ranging from 2.35 to 9.54 with the five problem sets.
—The speedup is high and reasonably regular for the two LGDP problem sets; it is lower and quite irregular for the other three problem sets.
—The speedup does not change the qualitative results of the evaluation in the previous sections: OPTIMATHSAT still performs better than all GAMS tools, including JAMS(BM)+CPLEX-4CORES, on the strip-packing, SMT-LIB, SAL, and MaxSMT/SMT+PB problem sets; it performs worse on the job-shop problem set.

Overall, we can conclude that OPTIMATHSAT is very competitive with, and often outperforms, GAMS LGDP tools on the very-extensive set of problems we have used to evaluate them, despite the possibility of GAMS to use CPLEX in parallel mode on multiple-core CPUs. This clearly demonstrates the potential of our novel OMT approach.

6. RELATED WORK

The idea of optimization in SMT was first introduced by Nieuwenhuis and Oliveras [2006], who presented a very-general 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. [2010] 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

[25]We have a technical remark: In order to use the same timeout mechanism for all tools, in all previous tests we have used the Linux command `ulimit` to handle the timeout for all OPTIMATHSAT, GAMS, and PB-MATHSAT versions. Unfortunately, `ulimit` does not seem to work properly for multithreaded processes, so that for JAMS(BM)+CPLEX-4CORES we had to use instead the GAMS/CPLEX internal timeout mechanism, which we have assumed to be reliable.

Procedure	#inst.	#term.	#correct	#err. msg.	#wrong	#unfeas.	time	average speedup
Strip-packing LGDP problems (Directly Generated Benchmarks)								
OPTIMATHSAT-LIN-IN	600	568	**568**	0	0	0	**12782**	–
JAMS(BM)+CPLEX	600	432	432	0	0	0	22318	–
JAMS(BM)+CPLEX-4CORES	600	472	472	0	0	0	17918	5.98
Job-shop LGDP problems (Directly Generated Benchmarks)								
OPTIMATHSAT-LIN-IN	600	577	577	0	0	0	43228	–
JAMS(BM)+CPLEX	600	587	587	0	0	0	34245	–
JAMS(BM)+CPLEX-4CORES	600	600	**600**	0	0	0	**10465**	9.54
SMT-LIB problems (LGDP-Encoded Benchmarks (SMT2LGDP$_2$))								
OPTIMATHSAT5-LIN-IN	194	194	**194**	0	0	0	**1604**	–
JAMS(BM)+CPLEX	194	172	166	0	4	2	6839	–
JAMS(BM)+CPLEX-4CORES	194	155	150	0	4	1	2990	2.36
SAL problems (LGDP-Encoded Benchmarks (SMT2LGDP$_2$))								
OPTIMATHSAT5-LIN-IN	392	385	**385**	0	0	0	**44129**	–
JAMS(BM)+CPLEX	392	28	14	0	14	0	1456	–
JAMS(BM)+CPLEX-4CORES	392	32	17	0	15	0	786	3.66
MaxSMT / SMT+PB problems (LGDP-Encoded Benchmarks (SMT2LGDP$_2$))								
OPTIMATHSAT-LIN-IN	675	630	**630**	0	0	0	**20744**	–
JAMS(BM)+CPLEX	675	449	92	9	351	6	1575	–
JAMS(BM)+CPLEX-4CORES	675	479	95	9	367	6	4033	2.35

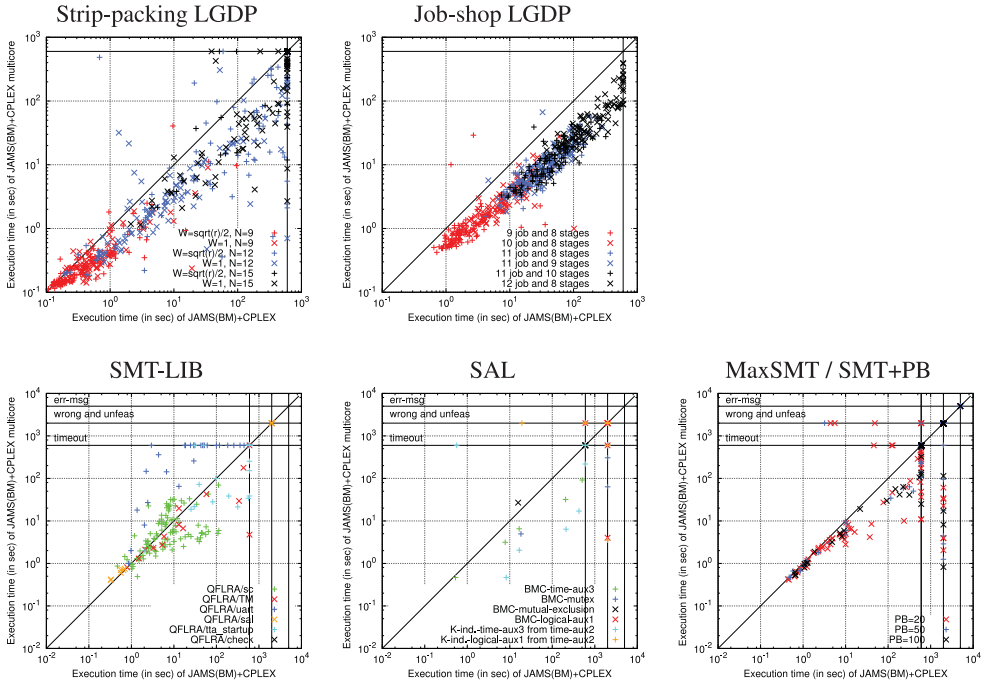

Fig. 10. Table: comparison of OPTIMATHSAT-LIN-IN, JAMS(BM)+CPLEX, and JAMS(BM)+CPLEX-4CORES on the five problem sets. Last column: average speedup for JAMS(BM)+CPLEX-4CORES with respect to JAMS(BM)+CPLEX. Scatter plots: pairwise comparison of JAMS(BM)+CPLEX-4CORES vs. JAMS(BM)+CPLEX on the five problem sets.

lazy SMT schema, and implemented a variant of the MathSAT tool able to handle SMT with PB constraints and to minimize PB cost functions.

The SMT solvers YICES [Dutertre and Moura 2006] and Z3 [de Moura and Bjørner 2008] also provide support for MaxSMT, although there is no publicly available document describing the procedures used there.

Ansótegui et al. [2011] describe the evaluation of an implementation of a MaxSMT procedure based on YICES, although this implementation is not publicly available.

Cimatti et al. [2013a] presented a "modular" approach for MaxSMT, combining a lazy SMT solver with a MaxSAT solver, which can be used as black boxes.

We recall that MaxSMT and SMT with PB functions can be encoded into each other, and that both are strictly less general than the problem addressed in this article (Section 3).

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. [2012] addressed the problem of finding *partial* models for quantified first-order formula modulo theories, which minimize the number of free variables that are assigned a value from the domain. Quoting an example from Dillig et al. [2012], given the formula $\varphi \stackrel{\text{def}}{=} (x+y+w > 0) \vee (x+y+z+w < 5)$, 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 that 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 $\mathcal{LA}(\mathbb{Z})$ and $\mathcal{EUF}$ into the MISTRAL tool.

Manolios and Papavasileiou [2013] 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-and-cut ILP method with $\mathcal{T}$-specific decision procedures, and implemented it into the INEZ tool. Notice that the approach of Manolios and Papavasileiou [2013] cannot combine ILP with $\mathcal{LA}(\mathbb{Q})$, since $\mathcal{LA}(\mathbb{Z})$ and $\mathcal{LA}(\mathbb{Q})$ are not signature disjoint (see Definition 2 in Manolios and Papavasileiou [2013]). Also, the objective function is defined on the integer domain.

We understand that neither of the previously mentioned works can handle the problem addressed in this article, and vice versa.

7. CONCLUSIONS AND FUTURE WORK

In this article, we have introduced the problem of OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), an extension of SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) with minimization of $\mathcal{LA}(\mathbb{Q})$ terms, and proposed two novel procedures addressing it. We have described, implemented, and experimentally evaluated this approach, clearly demonstrating all its potentials. We believe that OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) and its solving procedures are very-promising tools for a variety of optimization problems.

This research opens the possibility for several interesting future directions. A short-term goal, which we are currently working at, is to extend the approach to $\mathcal{LA}(\mathbb{Z})$ and to mixed $\mathcal{LA}(\mathbb{Q})/\mathcal{LA}(\mathbb{Z})$, by exploiting the solvers that are already present in MATHSAT [Griggio 2012]. As is implicitly suggested in Section 5.6, a medium-term goal is to investigate the parallelization of OMT procedures, so as to exploit the power of current multiple-core CPUs. A longer-term goal is to investigate the feasibility of extending the technique to deal with nonlinear constraints, possibly using MINLP tools as $\mathcal{T}$-Solver/Minimize.

A. APPENDIX: PROOF OF THE THEOREMS

A.1. Proof of Theorem 3.6

We first need to prove the following lemmas.

LEMMA A.1. *Let φ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula and $\mathcal{E} \stackrel{def}{=} \{\eta_1, \ldots, \eta_n\}$ be the set of all total truth assignments propositionally satisfying φ. Then* $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\eta_i \in \mathcal{E}} \mathsf{min}_{\mathsf{cost}}(\eta_i)$.

PROOF. If φ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, then $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\eta_i \in \mathcal{E}} \mathsf{min}_{\mathsf{cost}}(\eta_i) = +\infty$. Otherwise, the thesis follows straightforwardly from the fact that the set of the models of φ is the union of the sets of the models of the assignments in $\mathcal{E}$. □

LEMMA A.2. *Let φ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula and μ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable partial assignment such that $\mu \models_p \varphi$. Then there exists at least one $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable total assignment η such that $\mu \subseteq \eta$, $\eta \models_p \varphi$, and* $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\eta)$.

PROOF. Let $\mathcal{I}$ be a model for μ, and hence for φ. Then

$$\eta \stackrel{\text{def}}{=} \bigwedge_{\substack{\psi_i \in Atoms(\varphi) \\ \mathcal{I} \models \psi_i}} \psi_i \wedge \bigwedge_{\substack{\psi_i \in Atoms(\varphi) \\ \mathcal{I} \models \neg\psi_i}} \neg\psi_i. \tag{18}$$

By construction, η is a total truth assignment for φ and it is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable, $\mu \subseteq \eta$ and $\mathsf{min}_{\mathsf{cost}}(\eta) = \mathsf{min}_{\mathsf{cost}}(\mu) = \mathcal{I}(\mathsf{cost})$. Since $\mu \subseteq \eta$, then $\eta \models_p \varphi$. □

The proof of Theorem 3.6 then follows.

THEOREM 3.6. *Let φ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula and let $\mathcal{M} \stackrel{def}{=} \{\mu_1, \ldots, \mu_n\}$ be a complete collection of (possibly partial) truth assignments propositionally satisfying φ. Then* $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu)$.

PROOF. If φ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, then $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu) = +\infty$ by Definition 3.1 and Theorem 3.5. Otherwise, $\mathsf{min}_{\mathsf{cost}}(\varphi) < +\infty$. Then,

—Proof of $\mathsf{min}_{\mathsf{cost}}(\varphi) \leq min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu)$:
By reductio ad absurdum, suppose exists $\mu \in \mathcal{M}$ such that $\mathsf{min}_{\mathsf{cost}}(\mu) < \mathsf{min}_{\mathsf{cost}}(\varphi)$. By Proposition 3.3, μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ satisfiable. By Lemma A.2, there exists a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable total assignment η such that $\mu \subseteq \eta$, $\eta \models_p \varphi$, and $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\eta)$. By Lemma A.1, $\mathsf{min}_{\mathsf{cost}}(\eta) \geq \mathsf{min}_{\mathsf{cost}}(\varphi)$, and hence $\mathsf{min}_{\mathsf{cost}}(\mu) \geq \mathsf{min}_{\mathsf{cost}}(\varphi)$, contradicting the hypothesis.
—Proof of $\mathsf{min}_{\mathsf{cost}}(\varphi) \geq min_{\mu \in \mathcal{M}} \mathsf{min}_{\mathsf{cost}}(\mu)$:
From Lemma A.1, we have that $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\eta_i \in \mathcal{E}} \mathsf{min}_{\mathsf{cost}}(\eta_i)$. Let $\eta \in \mathcal{E}$ such that $\mathsf{min}_{\mathsf{cost}}(\varphi) = \mathsf{min}_{\mathsf{cost}}(\eta) < +\infty$. Hence, η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. Thus, there exists $\mu \in \mathcal{M}$ such that $\mu \subseteq \eta$. μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable since η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. From Proposition 3.3, $\mathsf{min}_{\mathsf{cost}}(\mu) \leq \mathsf{min}_{\mathsf{cost}}(\eta)$; hence, $\mathsf{min}_{\mathsf{cost}}(\mu) \leq \mathsf{min}_{\mathsf{cost}}(\varphi)$. Thus, the thesis holds. □

A.2. Proof of Theorem 3.10

THEOREM 3.10. *Let μ be as in Definition 3.9. Then,*

(a) $\mathsf{min}_{\mathsf{cost}}(\mu) = min_{\eta \in \mathcal{E}\mathcal{X}_{ed}(\mu)} \mathsf{min}_{\mathsf{cost}}(\eta)$
(b) *for all $\eta \in \mathcal{E}\mathcal{X}_{ed}(\mu)$,*

$$\mathsf{min}_{\mathsf{cost}}(\eta) = \begin{cases} +\infty & \text{if } \mu_{\mathcal{T}} \wedge \mu_{ed} \text{ is } \mathcal{T}\text{-unsatisfiable or} \\ & \text{if } \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed} \text{ is } \mathcal{LA}(\mathbb{Q})\text{-unsatisfiable} \\ \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}) & \text{otherwise.} \end{cases}$$

PROOF.

(a) Let

$$\mu' \stackrel{\text{def}}{=} \mu \wedge \bigwedge_{(x_i=x_j)\in\mathcal{IE}(\mu)} ((x_i = x_j) \vee \neg(x_i = x_j)).$$

μ and μ' are obviously $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-equivalent, so that $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\mu')$. By construction, $\mathcal{EX}_{ed}(\mu)$ is the set of all total truth assignments propositionally satisfying μ', so that $\mathsf{min}_{\mathsf{cost}}(\mu') = min_{\eta\in\mathcal{EX}_{ed}(\mu)}\mathsf{min}_{\mathsf{cost}}(\eta)$.

(b) By Theorem 3.8, η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable and $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable. Thus,

—if $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-unsatisfiable, then η is $\mathcal{LA}(\mathbb{Q})\cup\mathcal{T}$-unsatisfiable, so that $\mathsf{min}_{\mathsf{cost}}(\eta) = +\infty$.

—If $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$ is $\mathcal{LA}(\mathbb{Q})$-unsatisfiable, then η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, so that $\mathsf{min}_{\mathsf{cost}}(\eta) = \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}) = +\infty$.

—If $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed}$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable, then η is $\mathcal{LA}(\mathbb{Q})\cup\mathcal{T}$-satisfiable. We split the proof into two parts.

$\leq$ case: Let $c \in \mathbb{Q}$ be the value of $\mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})}\wedge\mu_{ed})$. Let $\mu' \stackrel{\text{def}}{=} \mu\wedge(\mathsf{cost} = c)$. Since $(\mathsf{cost} = c)$ is a $\mathcal{LA}(\mathbb{Q})$-pure atom, then $\mu' = \mu'_{\mathcal{T}} \wedge \mu'_{\mathcal{LA}(\mathbb{Q})}$ such that $\mu'_{\mathcal{T}} = \mu_{\mathcal{T}}$ and $\mu'_{\mathcal{LA}(\mathbb{Q})} = \mu_{\mathcal{LA}(\mathbb{Q})} \wedge (\mathsf{cost} = c)$, which are, respectively, $\mathcal{T}$- and $\mathcal{LA}(\mathbb{Q})$-pure and $\mathcal{T}$- and $\mathcal{LA}(\mathbb{Q})$-satisfiable by construction. Let $\eta' \stackrel{\text{def}}{=} \eta \wedge (\mathsf{cost} = c)$. Since $\mathcal{IE}(\mu) = \mathcal{IE}(\mu')$, then μ', $\mu'_{\mathcal{LA}(\mathbb{Q})}$, $\mu'_{\mathcal{T}}$, and η' match the hypothesis of Theorem 3.8, from which we have that η' is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable, so that η has a model $\mathcal{I}$ such that $\mathcal{I}(\mathsf{cost}) = c$. Thus, we have that $\mathsf{min}_{\mathsf{cost}}(\eta) \leq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed})$.

$\geq$ case: Let $c \in \mathbb{Q}$ be the value of $\mathsf{min}_{\mathsf{cost}}(\eta)$. Then, $\eta \wedge (\mathsf{cost} = c)$ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. We define μ', $\mu'_{\mathcal{LA}(\mathbb{Q})}$, $\mu'_{\mathcal{T}}$, and η' as in the "$\leq$" case. As before, they match the hypothesis of Theorem 3.8, from which we have that $\mu'_{\mathcal{LA}(\mathbb{Q})}$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable. Hence, $\mu_{\mathcal{LA}(\mathbb{Q})}$ has a model $\mathcal{I}$ such that $\mathcal{I}(\mathsf{cost}) = c$. Thus, we have that $\mathsf{min}_{\mathsf{cost}}(\eta) \geq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed})$. □

A.3. Proof of Theorem 3.12

THEOREM 3.12. *Let μ be as in Definition 3.9. Then,*

(*a*) *μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if some $\rho \in \mathcal{EX}_{edi}(\mu)$ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable.*
(*b*) $\mathsf{min}_{\mathsf{cost}}(\mu) = min_{\rho\in\mathcal{EX}_{edi}(\mu)}\mathsf{min}_{\mathsf{cost}}(\rho)$.
(*c*) *for all $\rho \in \mathcal{EX}_{edi}(\mu)$, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable.*
(*d*) *for all $\rho \in \mathcal{EX}_{edi}(\mu)$,*

$$\mathsf{min}_{\mathsf{cost}}(\rho) = \begin{cases} +\infty & \text{if } \mu_{\mathcal{T}} \wedge \mu_{ed} \text{ is } \mathcal{T}\text{-unsatisfiable or} \\ & \text{if } \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i \text{ is } \mathcal{LA}(\mathbb{Q})\text{-unsatisfiable} \\ \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i) & \text{otherwise.} \end{cases}$$

PROOF. Let

$$\mu^* \stackrel{\text{def}}{=} \mu \wedge \bigwedge_{(x_i=x_j)\in\mathcal{IE}(\mu)} \begin{pmatrix} ((x_i = x_j) \vee (x_i < x_j) \vee (x_i > x_j))\wedge \\ (\neg(x_i = x_j) \vee \neg(x_i < x_j))\wedge \\ (\neg(x_i = x_j) \vee \neg(x_i > x_j))\wedge \\ (\neg(x_i < x_j) \vee \neg(x_i > x_j)) \end{pmatrix}. \tag{19}$$

All clauses in the right conjuncts in (19) are $\mathcal{LA}(\mathbb{Q})$-valid, hence μ and μ^* are $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-equivalent, so that $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\mu^*)$. By construction, $\mathcal{EX}_{edi}(\mu)$ is the set of all total truth assignments propositionally satisfying μ^*.

(a) By Theorem 3.5, μ^* is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if some $\rho \in \mathcal{EX}_{edi}(\mu)$ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable, from which the thesis.

(b) $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\mu^*) = min_{\rho \in \mathcal{EX}_{edi}(\mu)} \mathsf{min}_{\mathsf{cost}}(\rho)$.

(c) We consider one $\rho \in \mathcal{EX}_{edi}(\mu)$. $\rho = \mu_{\mathcal{T}} \wedge \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_d \wedge \mu_i$. We notice that all literals in μ_i are $\mathcal{LA}(\mathbb{Q})$-pure, such that it is the $\mathcal{LA}(\mathbb{Q})$-pure part of ρ (namely, $\rho_{\mathcal{LA}(\mathbb{Q})}$). Thus, by Theorem 3.8, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable if and only if $\overbrace{\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i}^{\rho_{\mathcal{LA}(\mathbb{Q})}} \wedge \overbrace{\mu_e \wedge \mu_d}^{\mu_{ed}}$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable and $\mu_{\mathcal{T}} \wedge \overbrace{\mu_e \wedge \mu_d}^{\mu_{ed}}$ is $\mathcal{T}$-satisfiable. By construction, $\mu_i \models_{\mathcal{LA}(\mathbb{Q})} \mu_d$. Thus, $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i \wedge \mu_e \wedge \mu_d$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable if and only if $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i \wedge \mu_e$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable. Thus, the thesis holds.

(d) We consider one $\rho \in \mathcal{EX}_{edi}(\mu)$ and partition it as in point (c). From point (c), if $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-unsatisfiable or $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i \wedge \mu_e$ is $\mathcal{LA}(\mathbb{Q})$-unsatisfiable, then ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-unsatisfiable, so that $\mathsf{min}_{\mathsf{cost}}(\rho) = +\infty$. Otherwise, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable.

$\leq$ case: Let $c \in \mathbb{Q}$ be the value of $\mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i)$. Let $\mu' \stackrel{\text{def}}{=} \mu \wedge (\mathsf{cost} = c)$. Since $(\mathsf{cost} = c)$ is a $\mathcal{LA}(\mathbb{Q})$-pure atom, then $\mu' = \mu'_{\mathcal{T}} \wedge \mu'_{\mathcal{LA}(\mathbb{Q})}$ such that $\mu'_{\mathcal{T}} = \mu_{\mathcal{T}}$ and $\mu'_{\mathcal{LA}(\mathbb{Q})} = \mu_{\mathcal{LA}(\mathbb{Q})} \wedge (\mathsf{cost} = c)$, which are, respectively, $\mathcal{T}$- and $\mathcal{LA}(\mathbb{Q})$-pure. Also, $\mu'_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu'_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable by construction. Let $\rho' \stackrel{\text{def}}{=} \rho \wedge (\mathsf{cost} = c)$. Since $\mathcal{IE}(\mu) = \mathcal{IE}(\mu')$, then also μ', $\mu'_{\mathcal{LA}(\mathbb{Q})}$, $\mu'_{\mathcal{T}}$, and ρ' match the hypothesis of this theorem. Thus, by point (c), ρ' is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable, so that ρ has a model $\mathcal{I}$ such that $\mathcal{I}(\mathsf{cost}) = c$. Therefore, we have that $\mathsf{min}_{\mathsf{cost}}(\rho) \leq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i)$.

$\geq$ $\geq$ case: Let $c \in \mathbb{Q}$ be the value of $\mathsf{min}_{\mathsf{cost}}(\rho)$. Then, $\rho \wedge (\mathsf{cost} = c)$ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. We define μ', $\mu'_{\mathcal{LA}(\mathbb{Q})}$, $\mu'_{\mathcal{T}}$, and ρ' as in the "$\leq$" case. As before, they also match the hypothesis of this theorem, so that by point (c) $\mu'_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable. Thus, $\mu'_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ has a model $\mathcal{I}$ such that $\mathcal{I}(\mathsf{cost}) = c$. Therefore, we have that $\mathsf{min}_{\mathsf{cost}}(\rho) \geq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i)$. □

ACKNOWLEDGMENTS

We want to thank the following people for their help and contributions: Ignacio Grossmann, Nicolas Sawaya, Aldo Vecchietti, and the GAMS support team for providing help and useful suggestions via email about LGDP and the usage of the GAMS tools; our colleagues Alessandro Cimatti, Alberto Griggio, and Patrick Trentin for their feedback and suggestions; and the associate editor and the anonymous reviewers for providing many insightful comments and suggestions.

REFERENCES

Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. 2008. Constraint integer programming: A new approach to integrate CP and MIP. In *Proc. CPAIOR'08*. Lecture Notes in Computer Science, Vol. 5015. Springer, 6–20.

Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. 2011. Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem. In *SARA*.

Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani. 2005. Verifying industrial hybrid systems with MathSAT. In *Proc. BMC 2004 (ENTCS)*, Vol. 119. Elsevier.

Gilles Audemard, Alessandro Cimatti, Arthur Kornilowicz, and Roberto Sebastiani. 2002. SAT-based bounded model checking for timed systems. In *Proc. FORTE'02*. Lecture Notes in Computer Science, Vol. 2529. Springer.

Egon Balas. 1983. *Disjunctive Programming and a Hierarchy of Relaxations for Discrete Optimization Problems*. Technical Report DRC-7O-21-R3. Carnegie Mellon University. Retrieved from http://repository.cmu.edu/cgi/viewcontent.cgi?article=1949&cont ext=tepper.

Egon Balas. 1998. Disjunctive programming: Properties of the convex hull of feasible points. *Discr. Appl. Math.* 89, 1–3 (1998), 3–44.

Egon Balas and Pierre Bonami. 2007. New variants of lift-and-project cut generation from the LP tableau: Open source implementation and testing. In *IPCO*. 89–103.

Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. 2009. Satisfiabiity modulo theories. In *Handbook of Satisfiability*, Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). IOS Press, 825–885.

C. W. Barrett, D. L. Dill, and A. Stump. 2002. A generalization of Shostak's method for combining decision procedures. In *Frontiers of Combining Systems (FROCOS)*. Lecture Notes in Artificial Intelligence. Springer-Verlag. Santa Margherita Ligure, Italy.

A. Biere, A. Cimatti, E. M. Clarke, and Yunshan Zhu. 1999. Symbolic model checking without BDDs. In *Proc. TACAS'99*. 193–207.

Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability*. IOS Press. 980 pages.

Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. 2006. Efficient theory combination via boolean search. *Inf. Comput.* 204, 10 (2006), 1493–1525.

A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. 2011. *GAMS—A User's Guide*. GAMS Development Corporation, Washington, DC.

Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. 2007. Computing predicate abstractions by integrating BDDs and SMT solvers. In *FMCAD*. IEEE Comput. Soc., 69–76.

Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. 2010. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*. Lecture Notes in Computer Science, Vol. 6015. Springer, 99–113.

Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013a. A modular approach to MaxSAT modulo theories. In *International Conference on Theory and Applications of Satisfiability Testing, SAT*. Lecture Notes in Computer Science, Vol. 7962.

Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013b. The MathSAT 5 SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*. Lecture Notes in Computer Science, Vol. 7795. Springer, 95–109.

Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. 2010. Efficient generation of Craig interpolants in satisfiability modulo theories. *ACM Trans. Comput. Logics* 12, 1 (Oct. 2010).

Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. 2011. Computing small unsatisfiable cores in SAT modulo theories. *J. Artif. Intell. Res.* 40 (April 2011), 701–728.

Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *TACAS*. Lecture Notes in Computer Science, Vol. 4963. Springer, 337–340.

Isil Dillig, Thomas Dillig, Kenneth L. McMillan, and Alex Aiken. 2012. Minimum satisfying assignments for SMT. In *CAV*. 394–409.

Bruno Dutertre and Leonardo de Moura. 2006. A fast linear-arithmetic solver for DPLL(T). In *CAV*. Lecture Notes in Computer Science, Vol. 4144.

Bruno Dutertre and Leonardo De Moura. 2006. *The Yices SMT Solver*. Technical Report. SRI International. Retrieved from http://yices.csl.sri.com/tool-paper.pdf.

N. Eén and N. Sörensson. 2004. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT'03)*. Lecture Notes in Computer Science, Vol. 2919. Springer, 502–518.

Alberto Griggio. 2012. A practical approach to satisfiability modulo linear integer arithmetic. *J. Satis., Boolean Model. Comput.* 8 (2012), 1–27.

IBM. 2010. IBM ILOG CPLEX Optimizer. Retrieved from http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Natalia Kalinnik, Erika Ábrahám, Tobias Schubert, Ralf Wimmer, and Bernd Becker. 2010. Exploiting different strategies for the parallelization of an SMT solver. In *Proc. MBMV*. 97–106.

Chu Min Li and Felip Manyà. 2009. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). IOS Press, 613–631.

Andrea Lodi. 2009. Mixed integer programming computation. In *50 Years of Integer Programming 1958-2008*. Springer-Verlag, 619–645.

I. Lynce and J. P. Marques-Silva. 2004. On computing minimum unsatisfiable cores. In *Proc. 7th International Conference on Theory and Applications of Satisfiability Testing*.

Panagiotis Manolios and Vasilis Papavasileiou. 2013. ILP modulo theories. In *CAV*. 662–677.

Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. 2009. *Conflict-Driven Clause Learning SAT Solvers*, Chapter 4, 131–153. In Biere et al. [2009].

J. P. Marques-Silva and K. A. Sakallah. 1996. GRASP—A new search algorithm for satisfiability. In *Proc. ICCAD'96*. 220–227.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an efficient SAT solver. In *DAC*. 530–535.

C. G. Nelson and D. C. Oppen. 1979. Simplification by cooperating decision procedures. *TOPLAS* 1, 2 (1979), 245–257.

Robert Nieuwenhuis and Albert Oliveras. 2006. On SAT modulo theories and optimization problems. In *Proc. Theory and Applications of Satisfiability Testing (SAT'06)*. Lecture Notes in Computer Science, Vol. 4121. Springer.

R. Nieuwenhuis, A. Oliveras, and C. Tinelli. 2006. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *J. ACM* 53, 6 (Nov. 2006), 937–977.

D. A. Plaisted and S. Greenbaum. 1986. A structure-preserving clause form translation. *J. Symbol. Comput.* 2 (1986), 293–304.

R. Raman and I. E. Grossmann. 1994. Modelling and computational techniques for logic based integer programming. *Comput. Chem. Eng.* 18, 7 (1994), 563–578.

Olivier Roussel and Vaso Manquinho. 2009. *Pseudo-Boolean and Cardinality Constraints.* In *Handbook of Satisfiability*, Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). IOS Press, 695–733.

Nicolas W. Sawaya and Ignacio E. Grossmann. 2005. A cutting plane method for solving linear generalized disjunctive programming problems. *Comput. Chem. Eng.* 29, 9 (2005), 1891–1913.

Nicolas W. Sawaya and Ignacio E. Grossmann. 2012. A hierarchy of relaxations for linear generalized disjunctive programming. *Eur. J. Oper. Res.* 216, 1 (2012), 70–82.

Roberto Sebastiani. 2007. Lazy satisfiability modulo theories. *J. Sat., Boolean Model. Comput.* 3, 3–4 (2007), 141–224.

Roberto Sebastiani and Silvia Tomasi. 2012. Optimization in SMT with LA(Q) cost functions. In *IJCAR*. Lecture Notes in Artificial Intelligence, Vol. 7364. Springer, 484–498. Retrieved from http://disi.unitn.it/∼rseba/publist.html.

Meinolf Sellmann and Serdar Kadioglu. 2008. Dichotomic search protocols for constrained optimization. In *CP*. Lecture Notes in Computer Science, Vol. 5202. Springer.

Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. 2000. Checking safety properties using induction and a SAT-solver. In *FMCAD*. Lecture Notes in Computer Science. Springer, 108–125.

C. Tinelli and M. T. Harandi. 1996. A new correctness proof of the Nelson–Oppen combination procedure. In *Proc. Frontiers of Combining Systems (FroCoS'06) (Applied Logic)*. Kluwer.

A. Vecchietti. 2011. (2011). Personal communication.

A. Vecchietti and I. E. Grossmann. 2004. Computational experience with LogMIP solving linear and nonlinear disjunctive programming problems. In *Proc. of FOCAPD*. 587–590.

Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. 2009. A concurrent portfolio approach to SMT solving. In *Proc. Computer Aided Verification, CAV*. 715–720. DOI: http://dx.doi.org/10.1007/978-3-642-02658-4_60

S. Wolfman and D. Weld. 1999. The LPSAT engine & its application to resource planning. In *Proc. IJCAI*.

Received January 2014; revised September 2014; accepted December 2014