Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ Cost Functions [*]

Roberto Sebastiani and Silvia Tomasi

DISI, University of Trento, Italy

Abstract. In the contexts of automated reasoning and formal verification, 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, bit-vectors). Surprisingly, very little work has been done to extend SMT to deal with *optimization* problems; in particular, we are not aware of any work on SMT solvers able to produce solutions which minimize cost functions over *arithmetical* variables. This is unfortunate, since some problems of interest require this functionality.
In this paper we start filling this gap. We present and discuss two general procedures for leveraging SMT to handle the minimization of $\mathcal{LA}(\mathbb{Q})$ 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 AR 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 which 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.

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, e.g., those 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. (See [11] for an overview.)

Many SMT-encodable problems of interest, however, may require also the capability of finding models that are *optimal* wrt. some cost function over continuous arithmetical variables. [1] E.g., in (SMT-based) *planning with resources* [33] a plan for achieving a certain goal must be found which not only fulfills some resource constraints (e.g.

[*] R. Sebastiani is supported by Semiconductor Research Corporation under GRC Custom Research Project 2009-TJ-1880 WOLFLING and GRC Research Project 2012-TJ-2266 WOLF.

[1] Although we refer to quantifier-free formulas, as it is frequent practice in SAT and SMT, with a little abuse of terminology we often call "Boolean variables" the propositional atoms and we call "variables" the Skolem constants x_i in $\mathcal{LA}(\mathbb{Q})$-atoms like, e.g., "$3x_1 - 2x_2 + x_3 \leq 3$".

on time, gasoline consumption, ...) but that also minimizes the usage of some such resource; in SMT-based *model checking with timed or hybrid systems* (e.g. [9]) you may want to find executions which minimize some parameter (e.g. elapsed time), or which 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 which are traditionally addressed as *linear disjunctive programming (LDP)* [10] or *linear generalized disjunctive programming (LGDP)* [25, 28], or as SAT/SMT with Pseudo-Boolean (PB) constraints and Weighted Max-SAT/SMT problems [26, 19, 24, 15, 7]. Notice that the two latter problems can be easily encoded into each other.

Surprisingly, very little work has been done to extend SMT to deal with optimization problems [24, 15, 7]; in particular, to the best of our knowledge, all such works aim at minimizing cost functions over *Boolean* variables (i.e., SMT with PB cost functions or MAX-SMT), whilst we are not aware of any work on SMT solvers able to produce solutions which minimize cost functions over *arithmetical* variables. Notice that the former can be easily encoded into the latter, but not vice versa (see §2).

In this paper we start filling this gap. We present two general procedures for adding to SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) the functionality of finding models minimizing some $\mathcal{LA}(\mathbb{Q})$ cost variable —$\mathcal{T}$ being some (possibly empty) stably-infinite theory s.t. $\mathcal{T}$ 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, whilst the second, called *inline*, is more sophisticate and efficient, but it requires modifying the code of the SMT solver. (This distinction is important, since the source code of most SMT solvers is not publicly available.)

We have implemented these procedures within the MATHSAT5 SMT solver [5]. Due to the absence of competitors from AR and SMT domains, 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 which have been previously proposed as benchmarks for the latter tools. (Notice that LGDP is limited to plain $\mathcal{LA}(\mathbb{Q})$, so that, e.g., it cannot handle combination 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.

Related work. The idea of optimization in SMT was first introduced by Nieuwenhuis & Oliveras [24], 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 Max-SAT/SMT based on this framework. Cimatti et al. [15] introduced the notion of "Theory of Costs" $\mathcal{C}$ to handle PB cost functions and constraints by an ad-hoc and independent "$\mathcal{C}$-solver" in the standard lazy SMT schema, and implemented a variant of MathSAT tool able to handle SAT/SMT with PB constraints and to minimize PB cost functions. The SMT solver YICES [7] also implements Max-SAT/SMT, but we are not aware of any document describing the procedures used there.

Mixed Integer Linear Programming (MILP) is an extension of Linear Programming (LP) involving both discrete and continuous variables. 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 (see e.g. [20]). SAT techniques have also been incorporated into these procedures for MILP (see [8]).

Linear Disjunctive Programming (LDP) problems are LP problems where linear constraints are connected by conjunctions and disjunctions [10]. Closest to our domain, *Linear Generalized Disjunctive Programming (LGDP)*, is a generalization of LDP which has been proposed in [25] 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 [25, 32, 27, 28]; these reformulations focus on efficiently encoding disjunctions and logic propositions into MILP, so as to be fed to an efficient MILP solver like CPLEX.

Content. The rest of the paper is organized as follows: in §2 we define the problem addressed, and show how it generalizes many known optimization problems; in §3 we present our novel procedures; in §4 we present an experimental evaluation; in §5 we briefly conclude and highlight directions for future work.

2 Optimization in SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)

We assume the reader is familiar with the main concepts of Boolean and first-order logic. Let $\mathcal{T}$ be some stably infinite theory with equality s.t. $\mathcal{LA}(\mathbb{Q})$ and $\mathcal{T}$ are signature-disjoint, as in [23]. ($\mathcal{T}$ can be itself a combination of theories.) We call an *Optimization Modulo $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ problem, OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)*, a pair $\langle \varphi, \mathsf{cost} \rangle$ such that φ is a SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) formula and cost is a $\mathcal{LA}(\mathbb{Q})$ variable occurring in φ, representing the cost to be minimized. The problem consists in finding a model $\mathcal{M}$ for φ (if any) whose value of cost is minimum. We call an *Optimization Modulo $\mathcal{LA}(\mathbb{Q})$ problem (OMT($\mathcal{LA}(\mathbb{Q})$))* an SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) problem where $\mathcal{T}$ is empty. If φ is in the form $\varphi' \wedge (\mathsf{cost} < c)$ [resp. $\varphi' \wedge \neg(\mathsf{cost} < c)$] for some value $c \in \mathbb{Q}$, then we call c an *upper bound* [resp. *lower bound*] for cost. If ub [resp lb] is the minimum upper bound [resp. the maximum lower bound] for φ, we also call the interval $[\mathsf{lb}, \mathsf{ub}[$ the *range* of cost.

These definitions capture many interesting optimizations problems. First, it is straightforward to encode LP, LDP and LGDP into OMT($\mathcal{LA}(\mathbb{Q})$) (see [29] for details).

Pseudo-Boolean (PB) constraints (see [26]) in the form $(\sum_i \mathbf{a}_i X^i \leq b)$, s.t. X^i are Boolean atoms and $\mathbf{a}_i$ constant values in $\mathbb{Q}$, and cost functions $\mathsf{cost} = \sum_i \mathbf{a}_i X^i$, 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:

$$\bigwedge_i ((\neg X^i \vee (\mathbf{x}_i = \mathbf{a}_i)) \wedge (X^i \vee (\mathbf{x}_i = 0))). \qquad (1)$$

Moreover, since Max-SAT (see [19]) [resp. Max-SMT (see [24, 15, 7])] can be encoded into SAT [resp. SMT] with PB constraints (see e.g. [24, 15]), then optimization problems for SAT with PB constraints and Max-SAT can be encoded into OMT($\mathcal{LA}(\mathbb{Q})$), whilst those for SMT($\mathcal{T}$) with PB constraints and Max-SMT can be encoded into OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) (assuming $\mathcal{T}$ matches the definition above).

We remark 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 Max-SAT/SMT) addressed in [24, 15]. With the latter problem, the 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.

3 Procedures for OMT($\mathcal{LA}(\mathbb{Q})$) and OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$)

It may 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 former schema, 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 latter 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 §4.2.)

In this section we present more sophisticate procedures, based on the combination of SMT and minimization techniques. We first present and discuss an *offline* schema (§3.1) and an *inline* (§3.2) schema 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 (§3.2).

In what follows we assume the reader is familiar with the basics about CDCL SAT solvers and lazy SMT solvers. A detailed background section on that is available on the extended version of this paper [29]; for a much more detailed description, we refer the reader, e.g., to [22, 11] respectively.

3.1 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 optionally values for lb and ub (which are implicitly considered to be $-\infty$ and $+\infty$ if not present), and returns the model $\mathcal{M}$ of minimum cost and its cost u (the value ub if φ is $\mathcal{LA}(\mathbb{Q})$-inconsistent). We represent φ as a set of clauses, which may be pushed or popped from the input formula-stack of an incremental SMT solver.

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). [2] The solution space is then explored iteratively (rows 3-26), reducing at each loop the current range $[\mathsf{l}, \mathsf{u}[$ to explore, 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,

[2] Of course literals like $\neg(\mathsf{cost} < -\infty)$ and $(\mathsf{cost} < +\infty)$ are not added.

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: $\mathsf{l} \leftarrow \mathsf{lb}; \mathsf{u} \leftarrow \mathsf{ub}; \mathsf{PIV} \leftarrow \top; \mathcal{M} \leftarrow \emptyset$
 2: $\varphi \leftarrow \varphi \cup \{\neg(\mathsf{cost} < \mathsf{l}), (\mathsf{cost} < \mathsf{u})\}$
 3: while $(\mathsf{l} < \mathsf{u})$ do
 4:     if (BinSearchMode()) then {Binary-search Mode}
 5:         $\mathsf{pivot} \leftarrow \mathsf{ComputePivot}(\mathsf{l}, \mathsf{u})$
 6:         $\mathsf{PIV} \leftarrow (\mathsf{cost} < \mathsf{pivot})$
 7:         $\varphi \leftarrow \varphi \cup \{\mathsf{PIV}\}$
 8:         $\langle \mathsf{res}, \mu \rangle \leftarrow \mathsf{SMT.IncrementalSolve}(\varphi)$
 9:         $\eta \leftarrow \mathsf{SMT.ExtractUnsatCore}(\varphi)$
10:     else {Linear-search Mode}
11:         $\langle \mathsf{res}, \mu \rangle \leftarrow \mathsf{SMT.IncrementalSolve}(\varphi)$
12:         $\eta \leftarrow \emptyset$
13:     end if
14:     if $(\mathsf{res} = \textsc{sat})$ then
15:         $\langle \mathcal{M}, \mathsf{u} \rangle \leftarrow \mathsf{Minimize}(\mathsf{cost}, \mu)$
16:         $\varphi \leftarrow \varphi \cup \{(\mathsf{cost} < \mathsf{u})\}$
17:     else {$\mathsf{res} = \textsc{unsat}$ }
18:         if $(\mathsf{PIV} \not\in \eta)$ then
19:             $\mathsf{l} \leftarrow \mathsf{u}$
20:         else
21:             $\mathsf{l} \leftarrow \mathsf{pivot}$
22:             $\varphi \leftarrow \varphi \setminus \{\mathsf{PIV}\}$
23:             $\varphi \leftarrow \varphi \cup \{\neg\mathsf{PIV}\}$
24:         end if
25:     end if
26: end while
27: return $\langle \mathcal{M}, \mathsf{u} \rangle$
```

driven by the heuristic BinSearchMode(). Notice that if $\mathsf{u} = +\infty$ or $\mathsf{l} = -\infty$, then BinSearchMode() returns false.

In **linear-search mode**, steps 4-9 and 21-23 are not executed. First, an incremental SMT($\mathcal{LA}(\mathbb{Q})$) solver is invoked on φ (row 11). (Notice that, given the incrementality of the solver, every operation in the form "$\varphi \leftarrow \varphi \cup \{\phi_i\}$" [resp. $\varphi \leftarrow \varphi \setminus \{\phi_i\}$] is implemented as a "push" [resp. "pop"] operation on the stack representation of φ; it is also very important to recall that during the SMT call φ is updated with the clauses which are learned during the SMT search.) η is set to be empty, which forces condition 18 to hold. If φ is $\mathcal{LA}(\mathbb{Q})$-satisfiable, then it is returned res =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, returning the model $\mathcal{M}$ for μ of minimum cost u ($-\infty$ iff the problem in 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 16). 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 4-9), the value $\mathsf{pivot} \in]\mathsf{l}, \mathsf{u}[$ is computed by the heuristic function $\mathsf{ComputePivot}$ (in the simplest form, pivot is $(\mathsf{l} + \mathsf{u})/2$), the (possibly new) atom $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is pushed into the formula stack, so that to temporarily restrict the cost range to $[\mathsf{l}, \mathsf{pivot}[$; then the incremental SMT solver is invoked on φ, this time activating the feature $\mathsf{SMT.ExtractUnsatCore}$, which returns also the subset η of formulas in (the formula stack of) φ which caused the unsatisfiability of φ. This exploits techniques similar to unsat-core extraction [21]. (In practice, it suffices to say 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 $\varphi \setminus \{\mathsf{PIV}\}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent, i.e. 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 exploring 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 $\mathsf{BinSearchMode}()$ 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 $\varphi \setminus \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent. Thus, e.g., the procedure might got stuck into a 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 that $\mathsf{BinSearchMode}()$ returns false infinitely often, forcing then a "linear-search" call which finally detects the inconsistency. (In our implementation, we have empirically experienced the best performance with one linear-search loop after every binary-search one, because satisfiable calls are typically much cheaper than unsatisfiable ones.)

Under such hypothesis, it is straightforward to see the following facts: (i) Algorithm 1 terminates, in both modes, because there are only a finite number of candidate truth assignments μ to be enumerated, and steps 15-16 guarantee that the same assignment μ will never be returned twice by the SMT solver; (ii) it returns a model of minimum cost, because it explores the whole search space of candidate truth assignments, and for every suitable assignment μ $\mathsf{Minimize}$ finds the minimum-cost model for μ; (iii) it requires polynomial space, under the assumption that the underlying CDCL SAT solver adopts a polynomial-size clause discharging strategy (which is typically the case of SMT solvers, including MATHSAT).

In a nutshell, $\mathsf{Minimize}$ is a simple extension of the simplex-based $\mathcal{LA}(\mathbb{Q})$-Solver of [16] which is invoked after one solution is found, minimizing it by standard Simplex techniques. We recall that the algorithm in [16] can handle strict inequalities. Thus, if the input problem contains strict inequalities, then $\mathsf{Minimize}$ temporarily treats them as non-strict ones and finds the minimum-cost solution with standard Simplex techniques. If such minimum-cost solution $\mathbf{x}$ of cost min lays only on non-strict inequalities, then $\mathbf{x}$ is a solution; otherwise, for some $\delta > 0$ and for every cost $c \in]\mathsf{min}, \mathsf{min} + \delta]$ there exists a solution of cost c. (If needed, such solution is computed using the techniques for handling strict inequalities described in [16].) Thus the value min is returned, tagged as a non-strict minimum, so that the constraint $(\mathsf{cost} \leq \mathsf{min})$ rather than $(\mathsf{cost} < \mathsf{min})$ is added to φ.

Discussion. We remark a few facts about this procedure.

First, 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.

Second, the incrementality of the SMT solver plays an essential role here, since at every call SMT.IncrementalSolve resumes the status of the search of 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 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.

Third, we notice that in Algorithm 1 all the literals constraining the cost range (i.e., $\neg(\mathsf{cost} < \mathsf{l})$, $(\mathsf{cost} < \mathsf{u})$) are always added to φ as unit clauses; thus inside SMT.IncrementalSolve these literals are immediately unit-propagated, becoming part of each truth assignment μ from the very beginning of its construction. (We recall that the SMT solver invokes incrementally $\mathcal{LA}(\mathbb{Q})$-Solver also while building an assignment μ (*early pruning calls* [11].)) 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 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 & bound*: (i) decide a new literal l and propagate the literals which 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").

Fourth, 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 15 can be significantly smaller than pivot, so that the cost range is more than halved.

Finally, unlike with other domains (e.g., search in a sorted array) the binary-search strategy here is not "obviously faster" than the linear-search one, because the unsatisfiable calls to SMT.IncrementalSolve are typically much more expensive than the satisfiable ones, because 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 tradeoff 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 to use dynamic/adaptive strategies for ComputePivot (see [30]).

3.2 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. 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 & Pivoting. Every time the search of the CDCL SAT solver gets back to decision level 0, the range $[\mathsf{l}, \mathsf{u}[$ is updated s.t. u [resp. l] is assigned the lowest [resp.

highest] value u_i [resp. l_i] such that the atom $(\mathsf{cost} < \mathsf{u}_i)$ [resp. $\neg(\mathsf{cost} < \mathsf{u}_i)$] 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 $\mathsf{BinSearchMode}()$ is invoked: if it returns true, then $\mathsf{ComputePivot}$ computes $\mathsf{pivot} \in]\mathsf{l}, \mathsf{u}[$, and the (possibly new) atom $\mathsf{PIV} \overset{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is decided to be true (level 1) by the SAT solver. This mimics steps 4-7 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 which is found $\mathcal{LA}(\mathbb{Q})$-consistent by $\mathcal{LA}(\mathbb{Q})$-Solver, μ is also fed to $\mathsf{Minimize}$, 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, then unit-propagating $(\mathsf{cost} < \mathsf{min})$. This case mirrors steps 14-16 in Algorithm 1, permanently restricting the cost range to $[\mathsf{l}, \mathsf{min}[$. $\mathsf{Minimize}$ is embedded within $\mathcal{LA}(\mathbb{Q})$-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.
Increasing the Lower Bound. In binary-search mode, when a conflict occurs s.t. the conflict analysis of the SAT solver produces a conflict clause in the form $\neg\mathsf{PIV} \vee \neg\eta'$ s.t. all literals in η' are assigned true at level 0 (i.e., $\varphi \wedge \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent), then the SAT solver backtracks to level 0, unit-propagating $\neg\mathsf{PIV}$. This case mirrors steps 21-23 in Algorithm 1, permanently restricting the cost range to $[\mathsf{pivot}, \mathsf{u}[$.

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: e.g., 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 s.t. $\mathsf{m} \in [\mathsf{l}, \mathsf{u}[$ 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 considerations about the offline procedure in §3.1 hold for the inline version. The efficiency of the inline procedure can be further improved as follows.

First, in binary-search mode, when a truth assignment μ with a novel minimum min is found, not only $(\mathsf{cost} < \mathsf{min})$ but also $\mathsf{PIV} \overset{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ is learned as unit clause. Although redundant from the logical perspective because $\mathsf{min} < \mathsf{pivot}$, the unit clause PIV allows the SAT solver for reusing all the clauses in the form $\neg\mathsf{PIV} \vee C$ which have been learned when investigating the cost range $[\mathsf{l}, \mathsf{pivot}[$. (In Algorithm 1 this is done implicitly, since PIV is not popped from φ before step 16.) Moreover, 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 $\neg\eta \vee \neg(\mathsf{cost} < \mathsf{min})$ s.t. $\eta \subseteq \mu$, can be learned, which prevents the SAT solver from generating any assignment containing η.

Second, in binary-search mode, if the $\mathcal{LA}(\mathbb{Q})$-Solver returns a conflict set $\eta \cup \{\mathsf{PIV}\}$, then it is further asked to find the maximum value max s.t. $\eta \cup \{(\mathsf{cost} < \mathsf{max})\}$ is also $\mathcal{LA}(\mathbb{Q})$-inconsistent. (This is done with a simple modification of the algorithm in [16].) If $\mathsf{max} \geq \mathsf{u}$, then the clause $C^* \overset{\text{def}}{=} \neg\eta \vee \neg(\mathsf{cost} < \mathsf{u})$ is used do drive backjumping and learning instead of $C \overset{\text{def}}{=} \neg\eta \vee \neg\mathsf{PIV}$. Since $(\mathsf{cost} < \mathsf{u})$ is permanently assigned at level 0, the dependency of the conflict from PIV is removed. Eventually, instead of using C to drive backjumping to level 0 and propagating $\neg\mathsf{PIV}$, the SMT solver may use C^*, then forcing the procedure to stop.

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

The procedures of §3.1 and §3.2 extend to the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) case straightforwardly as follows. We assume that the underlying SMT solver handles $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$, and that φ is a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula (which for simplicity and wlog we assume to be pure [23]).

Algorithm 1 is modified as follows. First, $\mathsf{SMT.IncrementalSolve}$ in step 8 or 11 is asked to return also a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-model $\mathcal{M}$. Then $\mathsf{Minimize}$ is invoked on the pair $\langle \mathsf{cost}, \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{ei} \rangle$, s.t. $\mu_{\mathcal{LA}(\mathbb{Q})}$ is the truth assignment over the $\mathcal{LA}(\mathbb{Q})$-atoms in φ returned by the solver, and μ_{ei} is the set of equalities $(x_i = x_j)$ and strict inequalities $(x_i < x_j)$ on the shared variables x_i which are true in $\mathcal{M}$. (The equalities and strict inequalities obtained from the others by the transitivity of $=, <$ can be omitted.)

The implementation of an inline OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures comes nearly for free if the SMT solver handles $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-solving by *Delayed Theory Combination* [13], with the strategy of case-splitting automatically disequalities $\neg(x_i = x_j)$ into the two inequalities $(x_i < x_j)$ and $(x_j < x_i)$, which is implemented in MATHSAT. If so the solver enumerates truth assignments in the form $\mu' \stackrel{\text{def}}{=} \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{eid} \cup \mu_{\mathcal{T}}$, where (i) μ' propositionally satisfies φ, (ii) μ_{eid} is a set of interface equalities $(x_i = x_j)$ and disequalities $\neg(x_i = x_j)$, containing also one inequality in $\{(x_i < x_j), (x_j < x_i)\}$ for every $\neg(x_i = x_j) \in \mu_{eid}$; then $\mu'_{\mathcal{LA}(\mathbb{Q})} \stackrel{\text{def}}{=} \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{ei}$ and $\mu'_{\mathcal{T}} \stackrel{\text{def}}{=} \mu_{\mathcal{T}} \cup \mu_{ed}$ are passed to the $\mathcal{LA}(\mathbb{Q})$-Solver and $\mathcal{T}$-Solver respectively, μ_{ei} and μ_{ed} being obtained from μ_{eid} by dropping the disequalities and inequalities respectively. [3]

If this is the case, it suffices to apply $\mathsf{Minimize}$ to $\mu'_{\mathcal{LA}(\mathbb{Q})}$, then learn $(\mathsf{cost} < \mathsf{min})$ and use it for backjumping, as in §3.2.

For lack of space we omit here a detailed justification that the above procedures compute OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), which is presented in the extended paper [29]. In short, they correspond to apply the techniques of §3.1, §3.2 to look for minimum-cost $\mathcal{T}$-satisfiable and $\mathcal{LA}(\mathbb{Q})$-satisfiable truth-assignments for the $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formula $\varphi' \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{x_i, x_j \in Shared(\varphi)} ((x_i = x_j) \vee (x_i < x_j) \vee (x_j < x_i))$, which is equivalent to φ, each time passing to $\mathcal{T}$-solver, $\mathcal{LA}(\mathbb{Q})$-Solver and $\mathsf{Minimize}$ only the relevant literals.

4 Experimental Evaluation

We have implemented both the OMT($\mathcal{LA}(\mathbb{Q})$) procedures and the inline OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures of §3 on top of MATHSAT [5] (thus we refer to them as OPT-MATHSAT). We consider four different configurations of OPT-MATHSAT, depending on the approach (offline vs. inline, denoted by "-OF" and "-IN") and the search schema (linear vs. binary, denoted by "-LIN" and "-BIN"). [4]

[3] In [13] $\mu' \stackrel{\text{def}}{=} \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{ed} \cup \mu_{\mathcal{T}}$, μ_{ed} being a truth assignment over the interface equalities, and as such a set of equalities and disequalities. However, since typically a SMT($\mathcal{LA}(\mathbb{Q})$) solver handles disequalities $\neg(x_i = x_j)$ by case-splitting them into $(x_i < x_j) \vee (x_j < x_i)$, the assignment considers also one of the two strict inequalities, which is ignored by the $\mathcal{T}$-Solver and is passed to the $\mathcal{LA}(\mathbb{Q})$-Solver instead of the corresponding disequality.

[4] Here "-LIN" means that $\mathsf{BinSearchMode()}$ always returns false, whilst "-BIN" denotes the mixed linear-binary strategy described in §3.1 to ensure termination.

Due to the absence of competitors on OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$), we evaluate the performance of our four configurations of OPT-MATHSAT by comparing them against GAMS v23.7.1 [14] on OMT($\mathcal{LA}(\mathbb{Q})$) problems. GAMS provides two reformulation tools, LOGMIP v2.0 [4] and JAMS [3] (a new version of the EMP solver [2]), both of them allow to reformulate LGDP models by using either big-M (BM) or convex-hull (CH) methods [25, 28]. We use CPLEX v12.2 [18] (through an OSI/CPLEX link) to solve the reformulated MILP models. All the tools were executed using default options, as indicated to us by the authors [31].

Notice that OPT-MATHSAT uses *infinite precision arithmetic* whilst, to the best of our knowledge, the GAMS tools implement standard *floating-point arithmetic*.

All tests were executed on 2.66 GHz Xeon machines with 4GB RAM running Linux, using a timeout of 600 seconds. The correctness of the minimum costs min found by OPT-MATHSAT have been cross-checked by another SMT solver, YICES [7], by detecting 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 non-strict), 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. All tools agreed on the final results, apart from tiny rounding errors, [5] and, much more importantly, from some noteworthy exceptions on the smt-lib problems (see §4.2).

In order to make the experiments reproducible, the full-size plots, a Linux binary of OPT-MATHSAT, the problems, and the results are available at [1]. [6]

4.1 Comparison on LGDB Problems

We first performed our 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 [32, 27, 28]. We have adopted the encoding of the problems into LGDP given by the authors. [7]

The strip-packing problem. Given a set N of rectangles of different length L_j and height H_j, $j \in 1, .., N$, 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. We considered the LGDP model provided by [27] and a corresponding OMT($\mathcal{LA}(\mathbb{Q})$) encoding.

We randomly generated benchmarks 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 [27], which sorts the rectangles in non-increasing order of width and fills the strip by placing each rectangles in the bottom-left corner, and the

[5] GAMS +CPLEX often gives some errors $\leq 10^{-5}$, which we believe are due to the printing floating-point format: (e.g. "`3.091250e+00`"); notice that OPT-MATHSAT uses infinite-precision arithmetic, returning values like, e.g. "`7728125177/2500000000`".

[6] We cannot distribute the GAMS tools since they are subject to licencing restrictions. See [14].

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

Procedure	Strip-packing											
	$W = \sqrt{N}/2$						$W = 1$					
	$N = 9$		$N = 12$		$N = 15$		$N = 9$		$N = 12$		$N = 15$	
	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time	#s.	time
OPT-MATHSAT-LIN-OF	100	51	100	600	93	7862	100	588	90	4555	18	1733
OPT-MATHSAT-LIN-IN	100	32	100	449	96	8057	100	578	91	4855	22	3216
OPT-MATHSAT-BIN-OF	100	48	100	641	90	8712	100	641	88	4385	19	2251
OPT-MATHSAT-BIN-IN	100	32	100	458	96	9706	100	554	92	5892	21	3257
JAMS(BM)+CPLEX	100	381	66	8631	12	1411	50	161	88	4344	46	5978
JAMS(CH)+CPLEX	98	3414	23	1011	0	0	50	887	62	7784	14	3034
LOGMIP(BM)+CPLEX	100	239	78	10266	12	1170	100	164	91	3850	51	6619
LOGMIP(CH)+CPLEX	100	3004	27	2481	1	437	100	2032	70	7406	17	3860

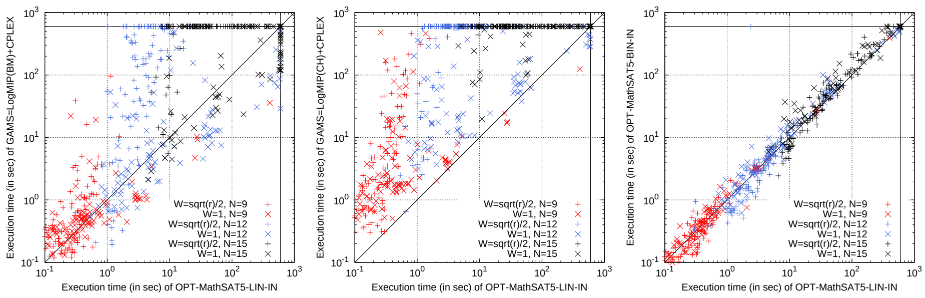

Fig. 1. Table: results (# of solved instances, cumulative time in seconds for solved instances) for OPT-MATHSAT and GAMS (using LOGMIP and JAMS) on 100 random instances each of the strip-packing problem for N rectangles, where $N = 9, 12, 15$, and width $W = \sqrt{N}/2$, 1. Scatter-plots: comparison of the best configuration of OPT-MATHSAT (OPT-MATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX (left), LOGMIP(CH)+CPLEX (center) and OPT-MATHSAT-BIN-IN (right).

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[8].

The table of Figure 1 shows the number of solved instances and their cumulative execution time for different configurations of OPT-MATHSAT and GAMS on the randomly-generated formulas. The scatter-plots of Figure 1 compare the best-performing version of OPT-MATHSAT, OPT-MATHSAT-LIN-IN, against LOGMIP with BM and CH reformulation (left and center respectively); the figure also compares the two inline versions OPT-MATHSAT-LIN-IN and OPT-MATHSAT-BIN-IN (right).

The zero-wait jobshop problem. Consider the scenario where there is a set I of jobs which 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*

[8] Notice that with $W = \sqrt{N}/2$ the filled strip looks approximatively like a square, whilst $W = 1$ is the average of two 2 rectangles.

Procedure	Job-shop					
	$I = 9, J = 8$		$I = 10, J = 8$		$I = 11, J = 8$	
	#s.	time	#s.	time	#s.	time
OPT-MATHSAT-LIN-OF	97	360	97	1749	92	9287
OPT-MATHSAT-LIN-IN	97	314	97	1436	93	7232
OPT-MATHSAT-BIN-OF	97	619	97	3337	85	13286
OPT-MATHSAT-BIN-IN	97	412	97	1984	93	9166
JAMS(BM)+CPLEX	100	263	100	1068	100	4458
JAMS(CH)+CPLEX	83	22820	6	2533	0	0
LOGMIP(BM)+CPLEX	100	259	100	1066	100	4390
LOGMIP(CH)+CPLEX	86	23663	6	2541	0	0

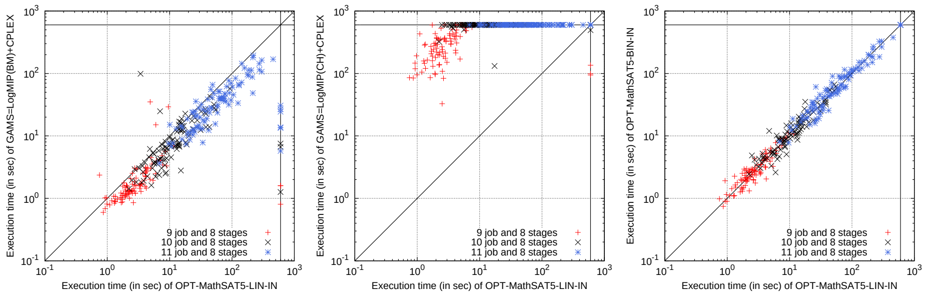

Fig. 2. Table: results (# of solved instances, cumulative time in seconds for solved instances) for OPT-MATHSAT and GAMS on 100 random samples each of the job-shop problem, for $J = 8$ stages and $I = 9, 10, 11$ jobs. Scatter-plots: comparison of the best configuration of OPT-MATHSAT (OPT-MATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX (left), LOGMIP(CH)+CPLEX (center) and OPT-MATHSAT-BIN-IN (right).

scheduling problem is to minimize the makespan, that is the total length of the schedule. In our experiments, we used the LGDP model used in [27] and a corresponding OMT($\mathcal{LA}(\mathbb{Q})$) encoding.

We randomly generated benchmarks 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 and 11 jobs and 8 stages. We set no value for ub and $\mathsf{lb} = 0$.

The table of Figure 2 shows the number of solved instances and their cumulative execution time for different configurations of OPT-MATHSAT and GAMS on the randomly-generated formulas. The scatter-plots of Figure 2 compare the best-performing version of OPT-MATHSAT, OPT-MATHSAT-LIN-IN, against LOGMIP with BM and CH reformulation (left and center respectively); the figure also compares the two inline versions OPT-MATHSAT-LIN-IN and OPT-MATHSAT-BIN-IN (right).

Discussion. The results in Figures 1 and 2 suggest some considerations.

Comparing the different version of OPT-MATHSAT, overall the -LIN options seems to perform a little better than and -BIN options (although gaps are not dramatic): in fact, OPT-MATHSAT-LIN-OF performs most often a little better than OPT-MATHSAT-

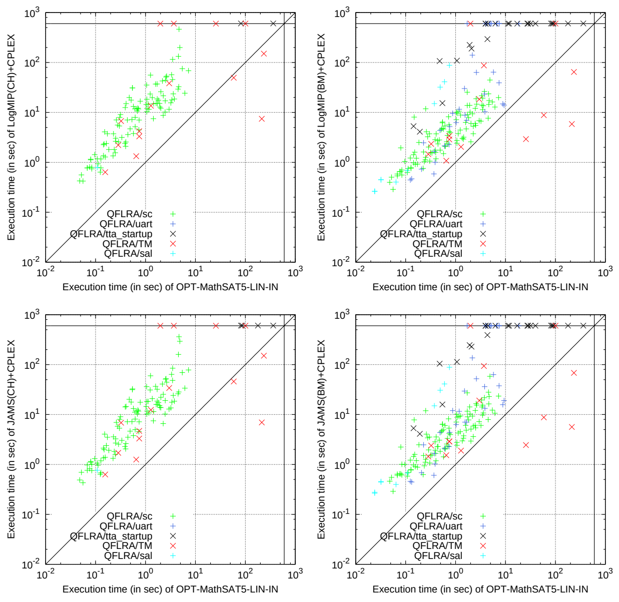

Fig. 3. Scatter-plots of the pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable instances between OPT-MATHSAT-BIN-IN and the two versions of LOGMIP (up) and JAMS. (down).

BIN-OF, OPT-MATHSAT-BIN-IN performances are slightly better than to those of OPT-MATHSAT-LIN-IN. We notice that the -IN options behave uniformly better than the -OF options.

Comparing the different versions of the GAMS tools, we see that (i) on strip-packing instances LOGMIP reformulations lead to better performance than JAMS reformulations, (ii) on job-shop instances they produce substantially identical results. For both reformulation tools, the "BM" versions uniformly outperform the "CH" ones.

Comparing the different versions of OPT-MATHSAT against all the GAMS tools, we notice that (i) on strip-packing problems all versions of OPT-MATHSAT most often outperform all GAMS tools, (ii) on job-shop problems OPT-MATHSAT outperforms the "CH" versions whilst it is beaten by the "BM" ones.

4.2 Comparison on SMT-LIB Problems

We compare OPT-MATHSAT against GAMS also on the satisfiable $\mathcal{LA}(\mathbb{Q})$-formulas (QF_LRA) in the SMT-LIB [6]. They are divided into six categories: `sc`, `uart`, `sal`, `TM`, `tta_startup`, and `miplib`. [9] Since we have no information on lower bounds on these problems, we use the linear-search version OPT-MATHSAT-`LIN-IN`. 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

[9] Notice that other SMT-LIB categories like `spider_benchmarks` and `clock_synchro` do not contain satisfiable instances and are thus not reported here.

it if the resulting minimum was $-\infty$. This forced us to eliminate a few instances, in particular all `miplib` ones.

We first noticed that some results for GAMS have some problem (see Table 1 in [29]). Using the default options, on ≈ 60 samples over 193, both GAMS tools with the CH option returned "unfeasible" (inconsistent), whilst the BM ones, when they did not timeout, returned the same minimum values as OPT-MATHSAT. (We recall that all OPT-MATHSAT results were cross-checked, and that the four GAMS tool were fed with the same files.) Moreover, on four `sal` instances the two GAMS tools with BM options returned a wrong minimum value "0", with "CH" they returned "unfeasible", whilst OPT-MATHSAT returned the minimum value "2"; by modifying a couple of parameters from their default value, namely "`eps`" and "`bigM Mvalue`", the results become unfeasible also with BM options. (We conjecture that these problems may be caused, at least in part, by the fact that GAMS tools use floating-point rather than infinite-precision arithmetic; nevertheless, this issue may deserve further investigation.)

After eliminating all flawed instances, the results appear as displayed in Figure 3. OPT-MATHSAT solved all problems within the timeout, whilst GAMS did not solve many samples. Moreover, with the exception of 3-4 samples, OPT-MATHSAT always outperforms the GAMS tool, often by more than one order magnitude.

5 Conclusions and Future Work

This research opens the possibility for several interesting future directions. A short-term goal is to improve the efficiency and applicability of OPT-MATHSAT: we plan to (i) investigate and implement novel mixed linear/binary-search strategies and heuristics (ii) extend the experimentation to novel sets of problems, possibly investigating ad-hoc customizations. A middle-term goal is to extend the approach to $\mathcal{LA}(\mathbb{Z})$ or mixed $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{LA}(\mathbb{Z})$, by exploiting the solvers which are already present in MATHSAT [17]. A much longer-term goal is to investigate the feasibility of extending the technique to deal with non-linear constraints, possibly using MINLP tools as $\mathcal{T}$-Solver/Minimize.

References

1. `http://disi.unitn.it/~rseba/ijcar12/ijcar12-tarball.tar.gz`.
2. EMP. `http://www.gams.com/dd/docs/solvers/emp.pdf`.
3. JAMS. `http://www.gams.com/`.
4. LogMIP v2.0. `http://www.logmip.ceride.gov.ar/index.html`.
5. MathSAT 5. `http://mathsat.fbk.eu/`.
6. SMT-LIB. `http://www.smtlib.org/`.
7. Yices. `http://yices.csl.sri.com/`.
8. T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: a new approach to integrate CP and MIP. In *Proc. CPAIOR'08*, LNCS, pages 6–20. Springer, 2008.
9. G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, 2002.
10. E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3 – 44, 1998.

11. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. In Biere et al. [12], February 2009.
12. A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications. IOS Press, February 2009.
13. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10), 2006.
14. A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS - A User's Guide*. GAMS Development Corporation, Washington, DC, USA, 2011.
15. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*. Springer, 2010.
16. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*, 2006.
17. A. Griggio. A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT*, 8:1–27, 2012.
18. IBM. *IBM ILOG CPLEX Optimizer*, 2010. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.
19. C. M. Li and F. Manyà. *MaxSAT, Hard and Soft Constraints*, chapter 19, pages 613–631. In Biere et al. [12], February 2009.
20. A. Lodi. Mixed Integer Programming Computation. In *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer-Verlag, 2009.
21. I. Lynce and J. Marques-Silva. On Computing Minimum Unsatisfiable Cores. In *SAT*, 2004.
22. J. P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. In Biere et al. [12], February 2009.
23. C. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *TOPLAS*, 1(2):245–257, 1979.
24. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *SAT*, volume 4121 of *LNCS*. Springer, 2006.
25. R. Raman and I. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18(7):563 – 578, 1994.
26. O. Roussel and V. Manquinho. *Pseudo-Boolean and Cardinality Constraints*, chapter 22, pages 695–733. In Biere et al. [12], February 2009.
27. N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Comput Chem Eng*, 29(9):1891–1913, 2005.
28. N. W. Sawaya and I. E. Grossmann. A hierarchy of relaxations for linear generalized disjunctive programming. *European Journal of Operational Research*, 216(1):70–82, 2012.
29. R. Sebastiani and S. Tomasi. Optimization in SMT with LA(Q) Cost Functions. Technical Report DISI-12-003, DISI, University of Trento, January 2012. `http://disi.unitn.it/~rseba/ijcar12/DISI-12-003.pdf`.
30. M. Sellmann and S. Kadioglu. Dichotomic Search Protocols for Constrained Optimization. In *CP*, volume 5202 of *LNCS*. Springer, 2008.
31. A. Vecchietti, 2011. Personal communication.
32. A. Vecchietti and I. Grossmann. Computational experience with logmip solving linear and nonlinear disjunctive programming problems. In *Proc. of FOCAPD*, pages 587–590, 2004.
33. S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.