PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

OPTIMIZATION MODULO THEORIES WITH LINEAR RATIONAL COSTS

Silvia Tomasi

Advisor:

Prof. Roberto Sebastiani

Università degli Studi di Trento

April 2014

Abstract

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, bit-vectors). Surprisingly, little work has been done to extend SMT to deal with optimization problems; in particular, concerning the development of SMT solvers able to produce solutions which minimize cost functions over arithmetical variables (we are aware of only one very-recent work [60]).

In the work described in this thesis 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 AR and FV, 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, and a very-recent SMT-based optimizer [60]. Our benchmark set consists of problems which have been previously proposed for our competitors. The results show that our tool is very competitive, and often outperforms these tools (especially LGDP ones) on their problems, clearly demonstrating the potential of the approach.*

Stochastic Local-Search (SLS) procedures are sometimes very competitive in pure SAT on both satisfiable instances and optimization problems. As a side

work, in this thesis we investigate the possibility to exploit SLS inside SMT tools which are commonly based on the lazy *approach (it combines a Conflict-Driven-Clause-Learning (CDCL) SAT solver with theory-specific decision procedures, called* $\mathcal{T}$-Solver*s). We first introduce a general procedure for integrating a SLS solver of the WalkSAT family with a* $\mathcal{T}$-Solver. *Then we present a group of techniques aimed at improving the synergy between these two components. Finally we implement all these techniques into a novel SLS-based SMT solver for the theory of linear arithmetic over the rationals, and perform an empirical evaluation on satisfiable instances. Although the results are encouraging, we concluded that the efficiency of proposed SLS-based SMT techniques is still far from being comparable to that of standard SMT solvers.*

Keywords

[Satisfiability Modulo Theory, Optimization, Linear Generalized Disjunctive Programming, Stochastic Local Search]

Contents

1 Introduction **1**

1.1 Main Contribution: Optimization Modulo Theories with Linear Rational Costs 1

1.2 A Secondary Contribution: Stochastic Local Search for SMT . . 3

1.3 Structure of the Thesis 5

1.4 Previous Publication 7

I Background and State of the Art **9**

2 Background **11**

2.1 Propositional Satisfiability 12

2.1.1 Conflict-Driven Clause Learning SAT Solvers 13

2.1.2 Sthocastic Local Search for SAT 17

2.2 Satisfiability Modulo Theories 22

2.2.1 The Satisfiability Modulo Theories Problem 22

2.2.2 Theory Solvers 26

2.2.3 Lazy SMT Solvers 29

2.2.4 Lazy SMT for Combinations of Theories 34

2.3 Linear Generalized Disjunctive Programming 38

2.3.1 Linear Programming 38

2.3.2 Mixed Integer Linear Programming 39

2.3.3 Disjunctive Programming 41
2.3.4 Linear Generalized Disjunctive Programming 42

3 State of the Art and Related Work 47
3.1 State of the Art 47
3.1.1 Optimization in SAT: MaxSAT and Pseudo-Boolean Optimization 47
3.1.2 SMT with Pseudo-Boolean Costs and MaxSMT 51
3.2 Other Forms of Optimization in SMT 54
3.3 A Very-Recent OMT($\mathcal{LA}(\mathbb{Q})$) Tool 55

II Novel Contributions 57

4 Optimization in SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) 59
4.1 Basic Definitions and Notation 59
4.2 Theoretical Results 61
4.2.1 OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) wrt. other Optimization Problems . 69

5 Procedures for OMT($\mathcal{LA}(\mathbb{Q})$) and OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) 73
5.1 An Offline Schema for OMT($\mathcal{LA}(\mathbb{Q})$) 74
5.1.1 Handling strict inequalities 77
5.1.2 Discussion. 78
5.2 An Inline Schema for OMT($\mathcal{LA}(\mathbb{Q})$) 80
5.3 Extensions to OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) 85

6 Experimental Evaluation for OMT($\mathcal{LA}(\mathbb{Q})$) 87
6.1 Encodings. 89
6.2 Comparison on LGDP Problems 91
6.2.1 The strip-packing problem. 91
6.2.2 The zero-wait jobshop problem. 93

6.2.3 Discussion 95
6.3 Comparison on SMT-LIB Problems 112
6.3.1 Discussion 114
6.4 Comparison on SAL Problems 119
6.4.1 Discussion 120
6.5 Comparison on Pseudo-Boolean SMT Problems 124
6.5.1 Discussion 124
6.6 Comparison on SYMBA Problems 129
6.6.1 Discussion 130

7 Stochastic Local Search in SMT **141**
7.1 Intuition 142
7.2 A basic WalkSMT procedure 144
7.3 Efficient $\mathcal{T}$-solvers for local search. 145
7.4 Enhancements to the basic WalkSMT procedure 146
7.4.1 Preprocessing 146
7.4.2 Single and Multiple Learning 148
7.4.3 Filterings 150

8 Experimental evaluation for WalkSMT **153**
8.1 Environment and Settings 153
8.2 WALKSMT on SMT-LIB Instances 155
8.3 WALKSMT on Random Instances 166
8.4 Discussion 169

9 Conclusions and Future Research Directions **171**

Bibliography **173**

Chapter 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 ($\mathcal{T}$-Solver$s$) for several first-order theories of practical interest like, e.g., 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. (See [79, 18] for an overview.)

1.1 Main Contribution: Optimization Modulo Theories with Linear Rational Costs

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* [94] a plan for achieving a certain goal must be found which not only ful-

[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 free constants x_i in $\mathcal{LA}(\mathbb{Q})$-atoms like, e.g., "$3x_1 - 2x_2 + x_3 \leq 3$".

fills some resource constraints (e.g. on time, gasoline consumption, ...) but that also minimizes the usage of some of such resources; in SMT-based *model checking with timed or hybrid systems* (e.g. [11, 10]) 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 *disjunctive programming (DP)* [13] or *linear generalized disjunctive programming (LGDP)* [75, 78], or as SAT/SMT with Pseudo-Boolean (PB) constraints and (weighted partial) MaxSAT/SMT problems [76, 59, 70, 31, 32]. Notice that the two latter problems can be easily encoded into each other.

Surprisingly, little work has been done so far to extend SMT to deal with *optimization* problems [70, 31, 80, 40, 32, 63, 60]. 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), whilst very little effort has been put into extending SMT solvers for producing solutions which minimize cost functions over *arithmetical* variables (we are aware of only one very-recent work [60]). Notice that the former optimization problem can be easily encoded into the latter, but not vice versa.

In this thesis we try to fill this gap producing the following contributions:

- We define the *Optimization Modulo* $\mathcal{LA}(\mathbb{Q})\cup\mathcal{T}$ *(OMT(*$\mathcal{LA}(\mathbb{Q})\cup\mathcal{T}$*))* problem which extends SMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) for finding models which minimize 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.

- We present two general procedures for solving OMT($\mathcal{LA}(\mathbb{Q})\cup\mathcal{T}$) by combining 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; 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 [33]. Due to the absence of competitors from AR and FV 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 $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$. As a last-minute addendum to this work, we also compared our implementation with the very-recent SMT-based tool presented in [60] on the authors own benchmarks problems. The results show that our tool is very competitive, and often outperforms these tools (especially LGDP ones) on their problems, clearly demonstrating the potential of the approach.

1.2 A Secondary Contribution: Stochastic Local Search for SMT

A dominant approach to SMT, called *lazy approach*, relies on the integration of a Conflict-Driven Clause-Learning (CDCL) SAT solver and of a decision procedure able to handle sets of atomic constraints in the underlying theory $\mathcal{T}$ ($\mathcal{T}$-Solver) (see, e.g., [79, 20]). In pure SAT, however, Stochastic Local-Search (SLS) procedures (see [53]) sometimes are competitive with or even outperform CDCL SAT solvers on satisfiable instances (in particular when dealing with unstructured problems) and on optimization problems where the goal is to find a solution of sufficiently high quality. Therefore, it is a natural research question

to wonder whether SLS can be exploited successfully also inside SMT tools, both solvers and optimizers. As side work, in this thesis we start investigating this issue.

Remarkably, CDCL and SLS SAT solvers are very different in the way they perform Boolean search. CDCL SAT solvers reason on *partial* truth assignments, which are updated in a stack-based manner. Moreover, they intensively use techniques like *Boolean constraint-propagation (BCP)*, *conflict-directed backtracking (backjumping)* and *learning*, which are heavily exploited in the lazy-SMT paradigm and allow for very-efficient SMT optimization techniques like *early pruning*, *theory-propagation*, *theory-driven backjumping and learning* (see [79, 20]). SLS SAT solvers, instead, reason on *total* truth assignments, which are updated by swapping the phase of single literals according to some mixed greedy/stochastic strategy. Moreover, they typically do not use BCP, backjumping and learning. Therefore, the problem of an effective integration of a $\mathcal{T}$-solver with a SLS SAT solver is not a straightforward variant of the standard integration with a CDCL solver in lazy SMT. Moreover, the standard SMT optimization techniques mentioned above cannot be applied in a straightforward way.

In order to cope with these problems, we present the following contributions:

- We present a novel and general architecture for integrating a $\mathcal{T}$-Solver with a Boolean SLS solver, which is inspired by the idea of "partially-invisible" SAT formulas.

- We analyze the differences between the interaction of a $\mathcal{T}$-solver with a CDCL-based and a SLS-based SAT solver, and we introduce and discuss a group of optimization techniques aimed at improving the synergy between an SLS solver and the $\mathcal{T}$-solver.

- We present an implementation of the proposed algorithms, called WALKSMT, which is based on the integration of the UBCSAT [89] and UBCSAT++ [23]

SLS solvers with the $\mathcal{LA}(\mathbb{Q})$-solver of MATHSAT [30].

- We provide an extensive experimental evaluation of our implementation. In particular, we evaluate the effects of the various optimization techniques, also comparing them against MATHSAT, on two groups of satisfiable instances: industrial problems coming from the SMT-LIB and randomly-generated unstructured problems. The results show that:

  1. the enhanced techniques drastically improve the performances of the basic version,
  2. the improved techniques drastically improve the performances of the basic version but WALKSMT cannot beat MATHSAT4.

Although the results are encouraging, we concluded that the efficiency of proposed SLS-based SMT techniques is still far from being comparable to that of standard SMT solvers.

1.3 Structure of the Thesis

This thesis is divided in two parts.

Part I provides the necessary background knowledge and terminology, and a survey of the literature on the topic of optimization in SAT and SMT.

Chapter 2 reviews theoretical results and algorithms at the basis of the lazy SMT approach and the LGDP paradigm. First, (§2.1) we give an overview of the state of the art in SAT solving by describing modern Conflict-Driven Clause Learning (CDCL) procedures and briefly introducing Stochastic Local Search (SLS) techniques (we focus the WalkSAT family of SLS algorithms). Then, (§2.2) we present the SMT framework and describe the algorithm for integrating a SAT solver with theory-specific decision

procedures which underlies lazy SMT solvers, its main enhanced techniques, relevant features of decision procedures for SMT, and methods for theory combination. Finally, (§2.3) we introduce LGDP and its solution approaches after recalling linear programming, mixed integer linear programming and disjunctive programming which inspired it and whose techniques are exploited by it.

Chapter 3 summarizes the state of the art and related work of optimization problems in the context of SAT and SMT. First, (§4) we describe optimization problems and procedures for solving them available in the literature of SAT (such as MaxSAT and Pseudo-Boolean Optimization) and SMT. In the context of SMT, we consider SMT with Pseudo-Boolean Costs and MaxSMT [70, 31, 6, 32]. Second, (§3.2) we report on other forms of optimization in SMT, such as the problem of finding minimum-cost assignments [40] and the "ILP Modulo Theories" framework [63]. Finally, (§3.3) we introduce a very-recent SMT-based tool for optimization, called SYMBA [60].

Part II is dedicated to the description of the novel contributions of this thesis.

Chapter 4 introduces the problem addressed in this thesis. First, (§4.1) we provide the definition of the Optimization Modulo Theory (OMT) problem and the theoretical foundations of the procedures for solving OMT, where the background theory is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$. Then, (§4.2.1) we show how the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) problem captures many interesting optimization problems described in the Part I.

Chapter 5 presents novel algorithms for solving OMT, based on the combination of SMT and minimization techniques. First, (§5.1 and §5.2) we present and discuss two procedures for OMT($\mathcal{LA}(\mathbb{Q})$), called "offline" and "inline" schema respectively. Then, (§5.3) we show how they can be extended to OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$).

Chapter 6 reports on an extensive experimental evaluation carried out for evaluating the implementation of the proposed procedures (we call it OPTIMATHSAT) against a LGDP tool, called GAMS. We consider different encodings (§6.1) and kind of benchmarks: (§6.2) LGDP problems (e.g. strip-packing and job-shop), (§6.3) SMT-LIB formulas augmented with cost functions, (§6.4) formulas obtained by using the SAL Model Checker on bounded verification problems, and (§6.5) Pseudo-Boolean SMT problems. As last-minute comparison, (§6.6) we evaluate OPTIMATHSAT against the recently-proposed tool SYMBA [60].

Chapter 7 describes a novel and general architecture for integrating a theory specific solver for $\mathcal{LA}(\mathbb{Q})$ with a SLS-based SAT solver resulting in a SLS-based SMT solver, called WALKSMT. First, we present the main intuition (§7.1), a basic architecture (§7.2). Then, (§7.3) we describe the most important features for efficient theory-specific decision procedures for SLS. Finally, (§7.4) we propose several enhanced techniques.

Chapter 8 presents the experimental evaluation conduced for evaluating the performance of WALKSMT and its enhancements. After introducing the experimental evaluation (§8.1), we report on experiments on SMT-LIB formulas and random generated problems (§8.2 and §8.3 respectively).

Chapter 9 briefly concludes the thesis and highlights directions for future work.

1.4 Previous Publication

Part of the content of this thesis contains material published in the following papers and technical reports.

Part I and Chapters 4, 5 and 7:

- Roberto Sebastiani and Silvia Tomasi. *Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ Cost Functions*. In Proceedings of IJCAR 2012, the 6th International Joint Conference on Automated Reasoning. Manchester, UK, 2012. [80].
- Roberto Sebastiani and Silvia Tomasi. *Optimization Modulo Theories with Linear Rational Costs*. Submitted to ACM Transactions on Computational Logic – TOCL [81].

Chapters 7 and 8:

- Silvia Tomasi. *Stochastic Local Search for SMT*. Technical Report. DISI-10-060, DISI, University of Trento. [87]
- Alberto Griggio, Roberto Sebastiani and Silvia Tomasi. *Stochastic Local Search for SMT: a Preliminary Report.* In Proceedings of SMT 2009, the 7th International Workshop on Satisfiability Modulo Theories. Montreal, Canada, 2009. [47]
- Alberto Griggio, Quoc Sang Phan, Roberto Sebastiani and Silvia Tomasi *Stochastic Local Search for SMT: Combining Theory Solvers with Walk-SAT*. In Proceedings of FroCoS 2011, the 8th International Symposium Frontiers of Combining Systems. Saarbrücken, Germany, 2011. [46].

Part I

Background and State of the Art

Chapter 2

Background

In this chapter we provide the background concepts and terminology of Propositional Satisfiability (SAT) (§2.1), Satisfiability Modulo Theories (SMT) (§2.2) and Linear Generalized Disjunctive Programming (LGDP) (§2.3).

Notation 2.1. We introduce a uniform notation which shall be used both in this chapter and in the rest of the thesis. We use boldface lowercase letters $\mathbf{a}, \mathbf{y}$ for arrays and boldface uppercase letters $\mathbf{A}, \mathbf{Y}$ for matrices, 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 $\mathbf{a}, ...\mathbf{e}$, ... $A, ...E$, to denote *constant* values, the last five $\mathbf{v}, ...\mathbf{z}$, ... $V, ...Z$ to denote *variables*, and the letters i, j, k, I, J, K for indexes and index sets respectively, pedices $._j$ denote the j-th element of an array or matrix, whilst apices $.^{ij}$ 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.

Disclaimer. The material presented in §2.1 is standard in SAT and it is mostly taken from [79] (and in part from [65]) for §2.1.1 and from [52, 51, 53, 88] for §2.1.2. The material presented in §2.2 is standard in SMT and it is mostly taken from [79] (other minor references are [71, 18]). The material presented in §2.2 is standard in LGDP and it is mostly taken from [61, 14, 75, 93, 77, 78].

2.1 Propositional Satisfiability

Given $\mathcal{A} = \{A_1, A_2, \ldots\}$ a non-empty set of primitive propositions, the language of propositional logic is the least set of formulas containing $\mathcal{A}$ and the primitive constants $\top$ and $\bot$ ("true" and "false" respectively) and closed under the set of standard propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. We call a *propositional atom* every primitive proposition in $\mathcal{A}$, and a *propositional literal* every propositional atom (*positive literal*) or its negation (*negative literal*). We implicitly remove double negations: e.g., if l is the negative literal $\neg A_i$, by $\neg l$ we mean A_i rather than $\neg\neg A_i$.

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.

Given a propositional formula φ, we call a *truth assignment* μ for φ a function mapping truth values $\{$**true**, **false**$\}$ to propositional atoms of φ. With a little abuse of notation, we represent μ indifferently as a *set* of literals $\{l_1, \ldots, l_n\}$, with the intended meaning that a positive [resp. negative] literal A_i means that A_i is assigned to true [resp. false], or as a *conjunction* of literals $l_1 \wedge \ldots \wedge l_n$; thus, e.g., we may say "$l_i \in \mu$" or "$\mu_1 \subseteq \mu_2$" (i.e. μ_2 *extends* μ_1 and μ_1 *subsumes* μ_2), but also "$\neg\mu$" meaning the clause "$\neg l_1 \vee \ldots \vee \neg l_n$".

A truth assignment μ *satisfies* φ, written $\mu \models \varphi$, if and only if φ is true under that truth assignment. (μ is called *model* for φ). A truth assignment is called *total* if it assigns a value to all atoms in φ, *partial* otherwise. We say that φ is *satisfiable* if and only if there exists at least one model for it. Two formulas φ_1 and φ_2 are called *equi-satisfiable* if and only if there exists μ_1 s.t. $\mu_1 \models \varphi_1$ if and only if there exists μ_2 s.t. $\mu_2 \models \varphi_2$.

The *Propositional Satisfiability (SAT)* problem is the problem of deciding the satisfiability of a propositional formula φ. We call *SAT solver* any procedure which decides whether φ is satisfiable, and returns a satisfying assignment if

this is the case. The following sections shall describe modern SAT solvers based on two different search approaches:

Systematic Search, which traverses the search space of a problem instance in a systematic manner. This guarantees that eventually a solution is found or no solution exists.

Local Search, which inspects the search space of a problem instance by iteratively moving from one search space position to a neighboring one on the basis of local knowledge only. This kind of algorithms are typically *incomplete* since there is no guarantee that eventually an existing solution is found.

2.1.1 Conflict-Driven Clause Learning SAT Solvers

Most modern SAT solvers are based on *systematic search* and are inspired by the well-known Davis-Putnam-Logemann-Loveland (DPLL) procedure [38]. These solvers take advantage of smart non-recursive implementation and very efficient data structures to handle Boolean formulas and assignments, and can be grouped into two families:

conflict-driven SAT solvers: the search process is guided by the analysis of the conflicts at every branch which fails [65];

look-ahead SAT solvers: the search process is built on top of a look-ahead procedure which calculates the reduction effect of the selection of each variable in a set [50].

In this section we focus on the former family of solvers and, in particular, on *Conflict-Driven Clause Learning* (CDCL) SAT solvers [65, 79]. A high-level schema is reported in Algorithm 1. The procedure takes as input a propositional formula φ in CNF and an initially-empty truth assignment μ which is updated

Algorithm 1 Conflict-Driven Clause Learning SAT Solver

```
Require: ⟨φ, μ⟩
 1: if preprocess(φ, μ) = CONFLICT then
 2:     return UNSAT
 3: end if
 4: loop
 5:     decide_next_branch(φ, μ)
 6:     loop
 7:         res ← boolean_constraint_propagation(φ, μ)
 8:         if res = SAT then
 9:             return SAT
10:         else if res = CONFLICT then
11:             blevel ← analyze_conflict(φ, μ)
12:             if blevel = 0 then
13:                 return UNSAT
14:             else
15:                 backtrack(blevel, φ, μ)
16:             end if
17:         else
18:             break
19:         end if
20:     end loop
21: end loop
```

in a stack-based manner. It performs a *Preprocessing* step in lines 1-3. The core part of the algorithm is given in the outer loop in lines 4-21, which alternates four main phases: *Decision*, *Boolean Constraint Propagation (BCP)*, *Conflict Analysis* and *Backjumping and Learning*.

Preprocessing preprocess(φ, μ) rewrites the input formula φ into a simpler and equi-satisfiable formula updating μ accordingly; if the resulting formula is unsatisfiable, then it returns UNSAT.

Some examples of simplification techniques are BCP, detecting and inlining Boolean equivalences among literals, performing resolutions steps to

given pairs of clauses, detecting and dropping subsumed clauses.

Decision decide_next_branch(φ, μ) selects an unassigned literal l from the pre-processed φ according to some heuristic function, and adds it to the assignment μ. The literal l is called *decision literal* end the number of decision literals in μ after this operation is called *decision level* of l.

Modern CDCL solvers compute a score at the end of each branch privileging variables that occurs in recently-learned clauses.

Boolean Constraint Propagation boolean_constraint_propagation(φ, μ) iteratively deduces literals l deriving from the current assignment, and updates φ and μ accordingly; this step is repeated until one of the following facts happens:

- μ satisfies φ and the procedure ends returning SAT;
- μ falsifies some clause ψ of φ (*conflicting clause*) and the procedures backtracks;
- no more literals can be deduced, so that the inner loop ends and a new decision is performed.

BCP is based on the iterative application of *unit propagation*: if all but one literals in a clause ψ are false, then the lonely 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. State-of-art SAT solver benefits of extremely efficient implementations of BMC (based on the *two-watched-literal scheme* [96]) and also other forms of deductions and formula simplification, e.g. on-line equivalence reasoning and variable and clause elimination.

Conflict Analysis analyze_conflict detects the subset η of μ which caused the conflict (called *conflict set*) and the decision level blevel to backtrack. Conflict analysis works as follows. Each literal is labelled with its decision

level, that is, the literal corresponding to the nth decision and the literals derived by unit-propagation after that decision are labelled with n; each non-decision literal l in μ is also labelled by a link to the clause ψ_l causing its unit-propagation (called the *antecedent clause* of l). When a clause ψ is falsified by the current assignment (in this case we say that a *conflict* occurs and ψ is the *conflicting clause*) 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 antecedent clause ψ_l of some literal l in ψ' (typically the last-assigned literal in ψ'), until some stop criterion is met. Some example follows: in the *1st-UIP Scheme* the last-assigned literal in ψ' is always picked and the process stops as soon as ψ' contains only one literal l_u assigned at the last decision level; in the *last-UIP Scheme*, l_u must be the last decision literal;

Backjumping and Learning If the computed blevel is equal to 0, then the procedure returns UNSAT since a conflict exists even without branching. Otherwise, $\mathsf{backtrack}(\mathsf{blevel}, \varphi, \mu)$ adds the *blocking clause* $\neg\eta$ to φ (*learning*) and backtracks up to blevel (*backjumping*), popping out of μ all literals whose decision level is greater than blevel, and updating φ and μ accordingly.

Other techniques often implemented in CDCL-based SAT solvers are:

search restarts causing the procedure to restart itself. Notice that previously-learned clauses are not deleted;

clause deletion policies to choose learned clauses which can be discharged "safely" when no more necessary. This guarantees the use of polynomial space without affecting the termination, correctness and completeness of the procedure.

Modern CDCL SAT solvers often provide two important features:

stack-based incremental interface, by which it is possible to push/pop (the blocks of clauses corresponding to) sub-formulas ϕ_i into a stack of formulas $\Phi \stackrel{\text{def}}{=} \{\phi_1, ..., \phi_k\}$, and check incrementally the satisfiability of $\bigwedge_{i=1}^{k} \phi_i$. The interface maintains the *status* of the search from one call to the other, also storing learned clauses. Consequently, when invoked on Φ the solver can reuse a clause C which was learned during a previous call on some Φ' if C was derived only from clauses which are still in Φ (provided C was not discharged in the meantime); in particular, if $\Phi' \subseteq \Phi$, then the solver can reuse all clauses learned while solving Φ'.

unsatisfiable-core extraction is the capability of CDCL SAT solvers, when Φ is found unsatisfiable, to return the subset of formulas in Φ which caused the unsatisfiability of Φ. Notice that such subset is not unique, and it is not necessarily minimal[1].

2.1.2 Sthocastic Local Search for SAT

Local search approach [53, 52] (see definition in §2.3) is widely used for solving hard combinatorial search problems. The idea is to examine the search space of a problem instance starting at some position and then iteratively moving from the present position to a neighboring one, where each location has a relatively small number of neighbors and moves are determined by decisions based on local knowledge only. When LS algorithms make use of randomized choices during both the initialization and the search process or of additional memory for storing historical information, they are called *Stochastic Local search (SLS)* algorithms. SLS algorithms are typically incomplete, however, in case of problems which are known to be solvable by nature or of optimization problems

[1] Φ is a minimal unsatisfiable core if $\Phi \setminus C$ is satisfiable for each clause C.

where the goal is to find a solution of sufficiently high quality, the ability to prove that no solution exists is not relevant.

SLS algorithms make use of an *evaluation function* for guiding the search towards solutions. It maps search positions onto real numbers the solutions of a given problem instance correspond to global minima of this function. However they can get stuck in local minima (i.e. positions having no improving neighbors) and plateau regions (i.e. regions not containing high-quality solutions) of the search space causing the premature stagnation of the search. There are many mechanisms for avoiding this, for example random restart which reinitializes the search if after a fixed number of steps (cutoff time) no solution has been found, or diversification steps such as random moves.

Generally SLS algorithm are often advantageous if the knowledge of the problem domain is rather limited and when relatively good solutions are required in a limited time, SLS algorithms returns the best solution found so far whereas systematic algorithms usually cannot provide approximated solutions.

SLS algorithms have been successfully applied to the solution of many NP-complete decision problems, including SAT. Typically SLS algorithms for SAT work with a CNF input formula (namely φ) and share a common high-level schema:

- initialization of the search by generating an initial truth assignment μ for φ (typically at random);
- iteratively selection of one (or more) Boolean atom A_i of φ which is then flipped within the current truth assignment μ.

The search terminates when μ satisfies the formula φ or when a given termination criterion is fulfilled. The main difference in SLS SAT algorithms is typically given by the evaluation function (e.g. the number of unsatisfied clauses of φ under μ) and the termination criterion. But almost every SLS SAT algorithm

Algorithm 2 Schema of WalkSAT Algorithms

```
Require: ⟨φ, MAX_TRIES, MAX_FLIPS⟩
 1: for i = 1 to MAX_TRIES do
 2:     μ ← initial_truth_assignment(φ)
 3:     for j = 1 to MAX_FLIPS do
 4:         if (μ ⊨ φ) then
 5:             return SAT
 6:         else
 7:             c ← choose_unsatisfied_clause(φ)
 8:             μ ← next_truth_assignment(φ, c)
 9:         end if
10:     end for
11: end for
12: return UNKNOWN
```

use random restarts.

In this section we focus on WalkSAT, a popular family of SLS-based SAT algorithms [53, 52].

WalkSAT Algorithms

The schema of WalkSAT algorithms is shown in Algorithm 2. It takes as input a CNF formula φ and two integer constants MAX_TRIES and MAX_FLIPS. Initially, initial_truth_assignment selects a complete truth assignment μ for the variables of φ (line 2) according to some heuristic criterion (e.g. uniformly at random). The search proceeds iteratively by selecting and flipping a variable in μ using a two-stage process:

1. choose_unsatisfied_clause selects a currently-unsatisfied clause $c \in \varphi$ according to some heuristic criterion (e.g. uniformly at random) (line 7).

2. next_truth_assignment chooses one of the variables occurring in the previously selected clause c according to some mixed greedy/random score function, and then flips it so that to generate another truth assignment

(line 8).

The procedure terminates when a solution is found (line 5) or after MAX_TRIES sequences of MAX_FLIPS variable flips without finding a model for φ (line 12).

Over the last ten years, several variants of the basic WalkSAT algorithm have been proposed [83, 66, 88], which differ mainly for the different heuristics used for the functions described above —in particular on the degree of greediness and randomness and in the criteria used for selecting the variable to flip in c within next_truth_assignment. One of the best performing WalkSAT-based algorithm for SAT seems to be *Adaptive Novelty*$^+$ [51, 88]. It adopts the Novelty$^+$'s variable selection heuristic, and it adjusts its degree of greediness according to the search progress. Novelty$^+$ deterministically chooses the variable to be flipped from c depending on two mechanisms:

score function which computes the difference in the total number of satisfied clauses a flip would cause,

memory which stores variable's age, i.e. the number of search steps performed since a variable was last flipped.

The variable selection heuristic works as follows:

1. if the variable with the highest score does not have minimal age among the variables in c, then it is selected;
2. otherwise, it is selected with a probability $1 - p$, where p is a parameter (called *noise setting*). While in the remaining cases p, the variable is picked uniformly at random (i.e. *random walk*).

The Adaptive Novelty$^+$'s adaptive mechanism which changes the probability of making greedy choices at runtime is based on the following idea. It starts with a completely greedy search ($p = 0$) until a solution is found or stagnation appears, in which case the noise setting p is increased and only once the stagnation situation is overcome, p is gradually decreased. (We call these two phases

diversification and *intensification* respectively). Search stagnation is detected if no progress in finding a solution has been observed over the last $\theta \times m$ search steps, where m is the number of clauses of φ and θ is an input parameter. For this reason, each time p changes, the current score function value is stored. During the search the noise setting is increased by $p = p+(1-p)\times\phi$ and decreased by $p = p - p \times \phi/2$, where ϕ is an input parameter; the asymmetry is due to the fact that detecting search stagnation is computationally more expensive than detecting search progress.

Trimming Variable Selection and Literal Commitment Strategy.

A few attempts have been made in order to enhance SLS algorithms with techniques borrowed from CDCL solvers (e.g. [23, 12]). This thesis focuses on the work of Belov and Stachniak [23], who propose two techniques that exploit the search history to improve the variable selection process of the classic SLS procedures for SAT. They modify the WalkSAT schema by adding a database (DB) that represents a set of constraints that help to guide the search process. It consists in:

- a partial truth assignment η that records assignments made by the local search heuristic (namely $\eta \subseteq \mu$);
- a set of clauses ψ obtained by storing selected unsatisfied-clauses (see line 7 of Algorithm 2).

The proposed techniques are inspired to clause learning and unit propagation, which are widely used by modern SAT solvers working with partial truth assignments (see §2.1.1):

Trimming variable selection tries to prune the search by preventing the selection of variables whose flip will cause a conflict in the database. In particular, for every variable v belonging to the selected clause c, the procedure

checks the satisfiability of $\psi \wedge \eta'$ by unit propagation, where η' is obtained from η by adding the (flipped) truth assignment of v under μ. If it is unsatisfiable, the variable v cannot be flipped. When all variables cause a conflict, the database is reset (i.e. η is set to $\emptyset$) so that any variable in c can be chosen by the local search heuristic. Notice that, once the truth value of a variable has been flipped, η is updated accordingly and the clause c is added to the database.

Literal commitment strategy aims at exploiting the power of unit propagation inside SLS procedures that naturally work with total truth assignments rather than partial ones. It iteratively deduces literals l in ψ deriving from η (i.e. $\psi \wedge \eta \models l$) and updates the current total truth assignment μ accordingly during a single search step.

2.2 Satisfiability Modulo Theories

In this section we illustrate the background on the *Satisfiability Modulo Theories* (SMT) problem. We first recall theoretical concepts and terminology in order to provide the definition of the problem; then we present the main approaches for solving it.

2.2.1 The Satisfiability Modulo Theories Problem

In what follows we recall some basic notions and terminology about first-order theories. We assume the standard syntax and semantics of first-order logic with equality as defined, e.g., in [91].

Consider a first-order signature Σ containing function and predicate symbols with their arities, and a set of variables V. A *constant* is a 0-ary function symbol c. A *Boolean atom* is a 0-ary predicate symbol A. A Σ*-term* is either a variable belonging to V or it is defined by applying function symbols in Σ to Σ-terms.

If $t_1, \ldots, t_n$ are Σ-terms and P is a predicate symbol, then $P(t_1, \ldots, t_n)$ is a Σ-*atom*. If t_1 and t_2 are two Σ-terms, then the Σ-atom $t_1 = t_2$ is called a Σ-*equality* and $\neg(t_1 = t_2)$ (also written as $l \neq r$) is called a Σ-*disequality*. A Σ-*formula* φ is built in the usual way out of the universal and existential quantifiers $\forall, \exists$, the Boolean connectives $\wedge, \neg$ and Σ-atoms. We apply the standard Boolean abbreviations, e.g. "$\phi_1 \vee \phi_2$" for "$\neg(\neg\phi_1 \wedge \neg\phi_2)$", "$\phi_1 \rightarrow \phi_2$" for "$\neg(\phi_1 \wedge \neg\phi_2)$", "$\top$" [resp. "$\bot$"] for the true [resp. false] constant. A Σ-*literal* is either a Σ-atom or its negation (i.e. a *positive literal* or a *negative literal*). By notation, we use the capital letters A_i and B_i to represent Boolean atoms, and the Greek letters α, β to represent Σ-atoms. A Σ-formula is *quantifier-free* if it does not contain quantifiers, and a *sentence* if it has no free variables. As for propositional formulas, a quantifier-free formula is in CNF if it is written as a conjunction of disjunctions of literals.

We write $\Gamma \models \phi$ to denote that the formula ϕ is a logical consequence of the (possibly infinite) set Γ of formulas. A Σ-theory is a set of first-order sentences with signature Σ. We consider theories which are first-order theories with equality. This means the equality symbol = is a predefined predicate and it is always interpreted as the identity on the underlying domain (so it will not be included in any signature Σ considered in this thesis). As a result, = is interpreted as a relation which is reflexive, symmetric, transitive, and it is also a congruence.

We call the Σ-structure $\mathcal{I}$ a *model* of a Σ-theory $\mathcal{T}$ if $\mathcal{I}$ satisfies every sentence in $\mathcal{T}$. A Σ-formula is *satisfiable* in $\mathcal{T}$ (or $\mathcal{T}$-*satisfiable*) if it is satisfiable in a model of $\mathcal{T}$. A Σ-formula is *valid* in $\mathcal{T}$ (or $\mathcal{T}$-*valid*) if it is satisfiable in all models of $\mathcal{T}$. We write $\Gamma \models_{\mathcal{T}} \phi$ to intend $\mathcal{T} \cup \Gamma \models \phi$. Two Σ-formulae ϕ and ψ are $\mathcal{T}$-*equisatisfiable* if and only if ϕ is $\mathcal{T}$-satisfiable if and only if ψ is $\mathcal{T}$-satisfiable.

Satisfiability Modulo Theory (SMT($\mathcal{T}$) for short) is the problem of deciding the $\mathcal{T}$-satisfiability of Σ-formulas, for some background theory $\mathcal{T}$[2].

[2] It is easy to see that SMT($\mathcal{T}$) subsumes SAT.

In this thesis we only consider quantifier-free Σ-formulas on some Σ-theory $\mathcal{T}$ (notice that the variables are implicitly existentially quantified). Therefore, when we refer to SMT($\mathcal{T}$) we mean the satisfiability problem in $\mathcal{T}$ of quantifier-free formulas. As it is frequent practice in SAT and SMT, with a little abuse of terminology we refer to predicates of arity zero (i.e. propositional atoms) as *Boolean variables*, uninterpreted constants as *Theory variables* (or $\mathcal{T}$*-variables*) and free constants x_i in quantifier-free linear arithmetic atoms (e.g., $(3x_1 - 2x_2 + x_3 \leq 3)$) as *variables*.

Combination of Theories

We briefly present some background on SMT($\mathcal{T}$) when $\mathcal{T}$ is a combination of theories.

We call a conjunction of $\mathcal{T}$-literals in a theory $\mathcal{T}$ *convex* if and only if for each disjunction $\bigvee_{I=1}^{n} x_i = y_i$ (where x_i, y_i are variables and $i = 1, \ldots, n$) we have that $\Gamma \models_{\mathcal{T}} \bigvee_{I=1}^{n} x_i = y_i$ if and only if $\Gamma \models_{\mathcal{T}} x_i = y_i$ for some $i \in \{1, \ldots, n\}$; A theory $\mathcal{T}$ is *convex* iff all the conjunctions of literals are convex in $\mathcal{T}$. We call a theory $\mathcal{T}$ *stably-infinite* if and only if for each $\mathcal{T}$-satisfiable formula φ, there exists a model of $\mathcal{T}$ whose domain is infinite and satisfies φ.

Consider two disjoint signatures Σ_1 and Σ_2 (i.e. $\Sigma_1 \cap \Sigma_2 = \emptyset$) and a theory $\mathcal{T}_i$ in Σ_i for $i = 1, 2$ s.t. $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$ and $\mathcal{T} \stackrel{\text{def}}{=} \mathcal{T}_1 \cup \mathcal{T}_2$. We refer to SMT($\mathcal{T}_1 \cup \mathcal{T}_2$) as the problem of discovering the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\Sigma_1 \cup \Sigma_2$-formulas.

A $\Sigma_1 \cup \Sigma_2$-term t is an *i-term* if and only if either it is a variable or it has the form $f(t_1, \ldots, t_n)$, where f is in Σ_i. (Thus a variable is both a 1-term and a 2-term.) A non-variable subterm s of an i-term t is *alien* if s is a j-term, and all superterms of s in t are i-terms, where $i, j \in \{1, 2\}$ and $i \neq j$. An i-term is *i-pure* if it does not contain alien subterms. An atom (or a literal) is *i-pure* if it contains only i-pure terms and its predicate symbol is either equality or in Σ_i. A $\Sigma_1 \cup \Sigma_2$-formula φ is called *pure* if every atom occurring in the formula is i-pure for some $i, j \in \{1, 2\}$. Notice that φ is pure if each atom belongs to only

one theory $\mathcal{T}_i$.

Consider a pure $\Sigma_1 \cup \Sigma_2$-formula φ, a variable in φ is an *interface variable* for φ if and only if it occurs in both 1-pure and 2-pure atoms of φ. An equality $(v_i = v_j)$ is an *interface equality* for φ if and only if v_i, v_j are interface variables for φ. We often refer to the interface equality $(v_i = v_j)$ as "e_{ij}".

Truth Assignments and Propositional Satisfiability in $\mathcal{T}$

We consider a generic quantifier-free decidable first-order theory $\mathcal{T}$ on a signature Σ. From now on, we will often omit the "Σ-" prefix from term, formula, theory, models, etc. We will also use the prefix "$\mathcal{T}$-" to intend "in the theory $\mathcal{T}$' (e.g. we call a "$\mathcal{T}$-formula" a formula in $\mathcal{T}$, "$\mathcal{T}$-model" a model in $\mathcal{T}$, etc.).

We call a *truth assignment* μ for a $\mathcal{T}$-formula φ a truth value assignment to the $\mathcal{T}$-atoms of φ. As for propositional satisfiability (see §2.1), a truth assignment is *total* if it assigns a value to all atoms in φ, *partial* otherwise. Notice that syntactically identical instances of the same $\mathcal{T}$-atom are always assigned identical truth values; syntactically different $\mathcal{T}$-atoms, e.g., $(t_1 \geq t_2)$ and $(t_2 \leq t_1)$, are treated differently and may thus be assigned different truth values.

We use a superscripted formula φ^p for denoting the *Boolean abstraction* of a SMT formula φ, which maps Boolean variables into themselves and theory $\mathcal{T}$-atoms into fresh Boolean atoms and distributes with sets and Boolean connectives. The formula φ is said to be the *refinement* of φ^p. 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.

We say that a total truth assignment μ *propositionally satisfies* a formula φ, written $\mu \models_p \varphi$, if $\mu^p \models \varphi^p$. We say that a partial truth assignment μ *propositionally satisfies* φ if and only if all the total truth assignments for φ which extend μ propositionally satisfy φ. (Thus, if not specified, when dealing with

propositional $\mathcal{T}$-satisfiability we do not distinguish between total and partial assignments.) If we consider a $\mathcal{T}$-formula φ as a propositional formula in its atoms, then $\models_p$ is the standard satisfiability in propositional logic (see §2.2). 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 often represent a truth assignment μ as a *conjunction* of $\mathcal{T}$-literals $l_1 \wedge \ldots \wedge l_n$ or a *set* of $\mathcal{T}$-literals $\{l_1, \ldots, l_n\}$. We adopt the same notation described in §2.2, and also use the convention s.t. if l is a negative Σ-literal $\neg\beta$, then by "$\neg l$" we conventionally mean β rather than $\neg\neg\beta$. We sometimes write a clause in the form of an implication (as in §2.1).

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 s.t. $\eta \models_p \varphi$, there exists $\mu_j \in \mathcal{M}$ s.t. $\mu_j \subseteq \eta$. Furthermore, we can see $\mathcal{M}$ as a compact representation of the whole set of total assignments propositionally satisfying φ.

Theorem 2.1 ([79])**.** *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 $\mu_j \in \mathcal{M}$.*

2.2.2 Theory Solvers

Consider some first-order theory $\mathcal{T}$. A *theory solver for* $\mathcal{T}$, $\mathcal{T}$-Solver, is any procedure able to decide the $\mathcal{T}$-satisfiability ($\mathcal{T}$-consistency) of a conjunction/set μ of $\mathcal{T}$-literals. Modern $\mathcal{T}$-Solvers support several features which are relevant to SMT($\mathcal{T}$). The most important are described in what follows.

Model Generation When a $\mathcal{T}$-Solver is invoked on a $\mathcal{T}$-satisfiable assignment μ, it may return a $\mathcal{T}$-model $\mathcal{I}$ for μ witnessing its consistency.

Conflict Set Generation When a $\mathcal{T}$-Solver is invoked on a $\mathcal{T}$-unsatisfiable assignment μ, it may return the 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*. We say that η is a *minimal theory conflict set* if all strict subsets of η are $\mathcal{T}$-consistent. A key efficiency issue for $\mathcal{T}$-Solver is the ability to produce small (possibly minimal) conflict sets.

Incrementality and Backtrackability $\mathcal{T}$-Solver$s$ are often invoked sequentially on *incremental* assignments, in a stack-based manner. For this reason, a crucial factor for efficiency of $\mathcal{T}$-Solver$s$ is that of being *incremental* and *backtrackable*.

- Incremental means that a $\mathcal{T}$-Solver remembers its computation status from one call to the other, so that, whenever it is given in input an assignment $\mu_1 \cup \mu_2$ such that μ_1 has just been proved $\mathcal{T}$-satisfiable, it avoids restarting the computation from scratch by restarting the computation from the previous status.
- Backtrackable means that it is possible to undo steps and return to a previous status on the stack in an efficient manner.

Deduction of Unassigned Literals When a $\mathcal{T}$-Solver is invoked on a $\mathcal{T}$-satisfiable assignment μ, it may return some unassigned $\mathcal{T}$-literal $l \notin \mu$ s.t. $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, where $\{l_1, ..., l_n\} \subseteq \mu$ and l is a literal on a not-yet-assigned atom in the input formula. We call $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ 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*. We say that $\mathcal{T}$-Solver is *deduction-complete* if it can perform all possible such deductions, or say that no such deduction can be performed.

Theory of Linear Arithmetic and $\mathcal{LA}$-solvers

The *Theory of Linear Arithmetic on the rationals* ($\mathcal{LA}(\mathbb{Q})$) and on the integer ($\mathcal{LA}(\mathbb{Z})$) is one of the theories of main interest in SMT. It is a first-order theory whose atoms are of the form $(a_1x_1 + \ldots + a_nx_n \diamond b)$ (i.e. $(\mathbf{ax} \diamond b)$) s.t $\diamond \in \{=, \neq, <, >, \leq, \geq,\}$. *Difference logic* on $\mathbb{Q}$ ($\mathcal{DL}(\mathbb{Q})$) is an important sub-theory of $\mathcal{LA}(\mathbb{Q})$, in which all atoms are in the form $(x_1 - x_2 \diamond b)$.

Efficient incremental and backtrackable procedures have been conceived in order to decide $\mathcal{LA}(\mathbb{Q})$ [41], $\mathcal{LA}(\mathbb{Z})$ [45] and $\mathcal{DL}$ [36]. In particular, for $\mathcal{LA}(\mathbb{Q})$ most SMT solvers implement variants of the simplex-based algorithm by Dutertre and de Moura [41] 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. This is based on the following Lemma.

Lemma 2.1 (Lemma 1 in [41]). *A set of $\mathcal{LA}(\mathbb{Q})$ atoms Γ containing strict inequalities $S = \{t_1 > 0, \ldots, t_n > 0\}$ is satisfiable iff there exists a rational number $\delta > 0$ such that $\Gamma_\delta \stackrel{def}{=} (\Gamma \cup S_\delta) \setminus S$ is satisfiable, s.t. $S_\delta \stackrel{def}{=} \{t_1 \geq \delta, \ldots, t_n \geq \delta\}$.*

The lemma states that it is possible to replace all strict inequalities by non-strict ones if a small enough δ is known. The idea of [41] is that of treating the *infinitesimal parameter* δ symbolically instead of explicitly computing its value. Strict bounds $(x < b)$ are replaced with weak ones $(x \leq b - \delta)$, and the operations on bounds are adjusted to take δ into account.

More specifically, they rewrite the original linear problem S into a new one S' where bounds and variable assignments range over pairs of rationals $\mathbb{Q}_\delta$. The intended meaning of a pair $\langle v, v_\delta \rangle \in \mathbb{Q}_\delta$ is $v + \delta v_\delta$ and the following operations

are defined in $\mathbb{Q}_\delta$:

$$\begin{aligned}\langle v, v_\delta\rangle + \langle u, u_\delta\rangle &\stackrel{\text{def}}{=} \langle v+u, v_\delta + u_\delta\rangle \\ a\langle v, v_\delta\rangle &\stackrel{\text{def}}{=} \langle av, av_\delta\rangle \\ \langle v, v_\delta\rangle \leq \langle u, u_\delta\rangle &\stackrel{\text{def}}{=} (v < u) \text{ or } ((v = u) \text{ and } (v_\delta \leq u_\delta))\end{aligned}$$

Thus, if the set of inequalities $\langle c_i, k_i\rangle \leq \langle d_i, h_i\rangle \in S'$ is satisfiable in $\mathbb{Q}_\delta$, then there is a positive rational number δ_0 s.t. the inequalities $c_i + k_i\epsilon \leq d_i + h_i\epsilon$ are satisfied for any ϵ s.t. $0 < \epsilon < \delta_0$. The solution β to the original problem S can be determined starting from the satisfying assignment β' for S' by:

1. computing the value of δ_0 as follows
$$\delta_0 = min\left\{\frac{d_i - c_i}{k_i - h_i} \mid c_i < d_i \text{ and } k_i > h_i\right\}$$
if the set on the right-hand side is nonempty or setting δ_0 to an arbitrary positive rational otherwise,

2. using δ_0 for calculating the value of every pair $\langle v, v_\delta\rangle \in \beta'$ as $v + \delta_0 v_\delta$.

2.2.3 Lazy SMT Solvers

The most popular approach to SMT($\mathcal{T}$) is called *lazy* and is based on combining a CDCL-based SAT solver (see §2.1) and one (or more) $\mathcal{T}$-Solver (s), respectively handling the Boolean and the theory-specific components of reasoning. More specifically, the SAT solver enumerates truth assignments which satisfy the Boolean abstraction of the input formula, and the $\mathcal{T}$-Solver checks the consistency in $\mathcal{T}$ of the set of literals corresponding to the assignments enumerated. We can partition lazy SMT solvers into two main categories:

***offline* solvers:** a CDCL SAT solver is used as black-box and re-invoked from scratch each time an assignment is found $\mathcal{T}$-unsatisfiable;

***inline* solvers:** the CDCL-schema of a SAT solver is modified to be used directly as an enumerator.

Algorithm 3 *Offline* Conflict-Driven Clause Learning SMT Solver

Require: φ
1: **while** CDCL-solver$(\varphi^p, \mu^p) = \text{SAT}$ **do**
2:     **if** $\mathcal{T}$-solver$(\mu) = \text{SAT}$ **then**
3:         **return** SAT
4:     **end if**
5:     $\varphi^p \leftarrow \varphi^p \wedge \neg\mu^p$
6: **end while**
7: **return** UNSAT

Offline SMT Solvers

The Algorithm 3 shows a basic schema of a typical offline SMT solver. The procedure takes as input a $\mathcal{T}$-formula φ, whose Boolean abstraction φ^p is given as input to CDCL-solver, which enumerates truth assignments μ_i^p for φ^p. If a new μ^p is found, its corresponding list of $\mathcal{T}$-literals μ is fed to $\mathcal{T}$-Solver (line 2). If μ contains also Boolean literals, then they are dropped because they do not take part in the $\mathcal{T}$-satisfiability of μ. If the Boolean refinement of μ^p is found $\mathcal{T}$-satisfiable, then φ is $\mathcal{T}$-consistent and the procedure returns SAT (line 3) possibly returning also the model $\mathcal{I}$ produced. Otherwise, $\neg\mu^p$ is added as a clause to φ^p (line 5), preventing CDCL-solver from finding the same assignment more than once, and CDCL-solver is restarted from scratch on the resulting formula. If no $\mathcal{T}$-satisfiable truth assignment is found by CDCL-solver, then the procedure returns UNSAT (line 7).

More efficiently, if the $\mathcal{T}$-Solver is able to return the conflict set η which caused the $\mathcal{T}$-inconsistency of μ, $\neg\eta^p$ is added as a clause to φ instead of $\neg\mu^p$ (typically the former set is smaller than the latter, drastically reducing the search).

Inline SMT Solvers

The basic schema of a typical inline SMT solver is presented in Algorithm 4

Algorithm 4 *Inline* Conflict-Driven Clause Learning SMT Solver

```
Require: ⟨φ, μ⟩
 1: if T-preprocess(φ, μ) = CONFLICT then
 2:     return UNSAT
 3: end if
 4: loop
 5:     T-decide_next_branch(φ^p, μ^p)
 6:     loop
 7:         res ← T-deduce(φ^p, μ^p)
 8:         if res = SAT then
 9:             return SAT
10:         else if res = CONFLICT then
11:             blevel ← T-analyze_conflict(φ^p, μ^p)
12:             if blevel = 0 then
13:                 return UNSAT
14:             else
15:                 T-backtrack(blevel, φ^p, μ^p)
16:             end if
17:         else
18:             break
19:         end if
20:     end loop
21: end loop
```

and mainly resembles the schema of a CDCL SAT Solver shown in Algorithm 1. The procedure takes as input a $\mathcal{T}$-formula φ (in CNF) and an (initially empty) set of $\mathcal{T}$-literals μ. It behaves as follows.

Preprocessing $\mathcal{T}$-preprocess rewrite the input formula φ into a simpler one and updates μ accordingly preserving the $\mathcal{T}$-satisfiability of $\varphi \wedge \mu$. If it finds some conflict, then the procedure returns UNSAT. $\mathcal{T}$-preprocess combines Boolean preprocessing (see §2.1.1) with theory-specific rewriting steps on the $\mathcal{T}$-literals of φ, e.g. normalizing $\mathcal{T}$-atoms and static learning (see [79] for details).

Decision $\mathcal{T}$-decide_next_branch resembles decide_next_branch in the CDCL schema (see §2.1.1), but it may also consider the semantics in $\mathcal{T}$ of the literals to select.

Deduction $\mathcal{T}$-deduce performs similarly to boolean_constraint_propagation in the CDCL schema: it iteratively deduces Boolean literals l^p which derive propositionally from the current assignment (i.e., s.t. $\varphi^p \wedge \mu^p \models_p l^p$) and updates φ^p and μ^p accordingly, until one of the following facts happens:

1. μ^p propositionally violates φ^p (i.e. $\mu^p \wedge \varphi^p \models \bot$): $\mathcal{T}$-deduce behaves like boolean_constraint_propagation in CDCL schema, returning CONFLICT.
2. μ^p propositionally satisfies φ^p (i.e. $\mu^p \models_p \varphi^p$): $\mathcal{T}$-deduce invokes $\mathcal{T}$-Solver on μ. If the latter returns SAT, then $\mathcal{T}$-deduce returns SAT; otherwise, $\mathcal{T}$-deduce returns CONFLICT.
3. no more literals can be deduced: $\mathcal{T}$-deduce returns UNKNOWN.

Important enhancements in $\mathcal{T}$-deduce can be implemented invoking the $\mathcal{T}$-Solver when an assignment μ is still under construction:

early pruning if μ is $\mathcal{T}$-unsatisfiable (i.e. $\mathcal{T}$-Solver returns UNSAT) then $\mathcal{T}$-deduce returns CONFLICT and the procedure backtracks, without exploring the (possibly many) extensions of μ;

$\mathcal{T}$-propagation if μ is $\mathcal{T}$-satisfiable (i.e. $\mathcal{T}$-Solver returns SAT) and the $\mathcal{T}$-Solver is able to perform deductions of unassigned literals $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, then $\mathcal{T}$-deduce can iteratively deduce literals l which 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.

Conflict Analysis $\mathcal{T}$-analyze_conflict extends analyze_conflict in CDCL schema:

- if the conflict produced by deduce is caused by a Boolean failure (case (1) above), then $\mathcal{T}$-analyze_conflict conflict produces a Boolean conflict set η^p and the corresponding value of blevel, as described in §2.1.1;
- if the conflict is caused by a $\mathcal{T}$-inconsistency revealed by $\mathcal{T}$-Solver (case (2) or (3) above), then $\mathcal{T}$-analyze_conflict conflict produces as a conflict set the Boolean abstraction η^p of the theory conflict set η produced by $\mathcal{T}$-Solver, or computes a mixed Boolean+theory conflict set by a backward-traversal of the implication graph starting from the conflicting clause $\neg\eta^p$ (see §2.1.1). If $\mathcal{T}$-Solver is not able to return a theory conflict set, the whole assignment μ may be used, after removing all Boolean literals from μ.

$\mathcal{T}$-Backjumping and $\mathcal{T}$-Learning Once the conflict set η^p and blevel have been computed, $\mathcal{T}$-backtrack behaves analogously to backtrack in CDCL schema: it adds the clause $\neg\eta^p$ to φ^p and backtracks up to blevel.

Other relevant enhancements for CDCL SMT Solvers are:

Pure-literal filtering if some $\mathcal{LA}(\mathbb{Q})$-atoms occur only positively [resp. negatively] in the original formula (learned clauses are ignored), then we can safely drop every negative [resp. positive] occurrence of them from the assignment μ to be fed into $\mathcal{T}$-Solver [79]. The benefits of this action are:

1. reduction of the workload for the $\mathcal{T}$-Solver which receives smaller sets of $\mathcal{T}$-literals;
2. increase in the probability of finding a $\mathcal{T}$-consistent satisfying assignment by removing "useless" $\mathcal{T}$-literals which may cause the $\mathcal{T}$-inconsistency of μ.

Ghost-literal filtering $\mathcal{T}$-literals occurring only in clauses which have already been satisfied (we call them *ghost $\mathcal{T}$-literals*) in μ, increase the work of

the $\mathcal{T}$-Solver and may affect the $\mathcal{T}$-satisfiability of μ forcing unnecessary backtracks and causing unnecessary Boolean search and hence useless calls to the $\mathcal{T}$-Solver. Thus, every occurrence of ghost $\mathcal{T}$-literals may be safely removed from μ, e.g. by monitoring the satisfaction of the (original) clauses in φ in which the selected literal occurs.

Static learning it detects a priori short and "obviously $\mathcal{T}$-inconsistent" assignments to $\mathcal{T}$-atoms in φ (typically pairs or triplets), e.g. incompatible value assignments ($\{x = 0, x = 1\}$), transitivity constraints ($\{(x - y \leq 2), (y - z \leq 4), \neg(x - z \leq 7)\}$), equivalence constraints ($\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$).

2.2.4 Lazy SMT for Combinations of Theories

Many practical applications of SMT require a combination of two or more theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ rather than just one. Two main approaches to the development of lazy SMT($\mathcal{T}$) solvers for combination of theories[3] have been proposed:

Nelson-Oppen (N.O.) Combination: Nelson and Oppen [69, 72] were the pioneers in this field (together with Shostak [85]) and established the theoretical foundations onto which most modern combined procedures are based on. They also proposed a general-purpose procedure for integrating $\mathcal{T}_i$-solvers into one combined $\mathcal{T}$-solver which is integrated into a SMT solver according to the CDCL schema.

Delayed Theory Combination (DTC): it is a combination procedure (proposed by Bozzano et al. [27]) which builds a combined SMT solver directly by exploiting the CDCL schema also for theory combination.

Modern SMT($\mathcal{T}$) solver are built on top of variants or evolutions of these two approaches [17, 39, 55, 27].

[3]For simplicity we often refer to combinations of two theories $\mathcal{T}_1 \cup \mathcal{T}_2$ only; however, all the discourse can be easily generalized to combination of many theories.

Nelson-Oppen Theory Combination

Consider two decidable stably-infinite theories with equality $\mathcal{T}_1$ and $\mathcal{T}_2$ and disjoint signatures Σ_1 and Σ_2 (often called Nelson-Oppen theories) and consider a pure conjunction of $\mathcal{T}_1 \cup \mathcal{T}_2$-literals $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ s.t. $\mu_{\mathcal{T}_i}$ is i-pure for each i. Nelson and Oppen's key observation is that μ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable if and only if it is possible to find two satisfying interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ s.t. $\mathcal{I}_1 \models_{\mathcal{T}_1} \mu_{\mathcal{T}_1}$ and $\mathcal{I}_2 \models_{\mathcal{T}_2} \mu_{\mathcal{T}_2}$ which agree on all equalities on the shared variables.

Overall, the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability problem of a set of pure literals μ is reduced to the problem of finding an equivalence relation on the shared variables which is consistent with both pure parts of μ. The condition of having only pure conjunctions as input allows to partition the problem into two independent $\mathcal{T}_i$-satisfiability problems $\mu_{\mathcal{T}_i} \wedge \mu_e$. This condition is easy to address, because every non-pure $\mathcal{T}_1 \cup \mathcal{T}_2$-formula φ can be converted into a $\mathcal{T}_1 \cup \mathcal{T}_2$-equisatisfiable and pure one by recursively replacing each alien subterm t by a new variable v_t and conjoining the equality $v_t = t$ with φ (this process is called *purification* [68]). The condition of having stably-infinite theories is sufficient to guarantee enough values in the domain to allow the satisfiability of every possible set of disequalities one may encounter.

The combined decision procedure $\mathcal{T}_1 \cup \mathcal{T}_2$-solver works by performing a structured interchange of interface equalities (disjunctions of interface equalities if $\mathcal{T}_i$ is non-convex) which are inferred by either $\mathcal{T}_i$-solver and then propagated to the other, until convergence is reached. Each $\mathcal{T}_i$-solver must be e_{ij}-deduction complete, i.e. it must be able to derive the (disjunctions of) interface equalities e_{ij} which are entailed by its current facts φ.

If the theories are convex, then the $\mathcal{T}_1 \cup \mathcal{T}_2$-solver receives from the CDCL SAT solver a pure set of literals μ, and partitions it into $\mu_{\mathcal{T}_1} \cup \mu_{\mathcal{T}_2}$, s.t. $\mu_{\mathcal{T}_i}$ is i-pure, and feeds each $\mu_{\mathcal{T}_i}$ to the respective $\mathcal{T}_i$-solver. Each $\mathcal{T}_i$-solver, in turn:

1. checks the $\mathcal{T}_i$-satisfiability of $\mu_{\mathcal{T}_i}$,

2. deduces all the interface equalities deriving from $\mu_{\mathcal{T}_i}$,

3. passes them to the other $\mathcal{T}$-solver, which adds it to his own set of literals.

This process is repeated until either one $\mathcal{T}_i$-solver detects inconsistency ($\mu_{\mathcal{T}_1} \cup \mu_{\mathcal{T}_2}$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsatisfiable), or no more e_{ij}-deduction is possible ($\mu_{\mathcal{T}_1} \cup \mu_{\mathcal{T}_2}$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiable).

If the theories are non-convex, then the two solvers need to exchange arbitrary disjunctions of interface equalities. As each $\mathcal{T}_i$-solver can handle only conjunctions of literals, the disjunctions must be managed by means of case splitting and of backtrack search. Thus, the N.O. procedure must explore a number of branches to check the consistency of a set of literals which depends on how many disjunctions of equalities are exchanged at each step: if the current set of literals is μ, and one of the $\mathcal{T}_i$-solver sends the disjunction $\bigvee_{k=1}^{n} = (e_{ij})_k$ to the other, the latter must further investigate up to n branches to check the consistency of each of the $\mu \cup (e_{ij})_k$ sets separately.

Delayed Theory Combination

Delayed Theory Combination (DTC) is based on the Nelson and Oppen's formal framework and thus considers signature-disjoint stably-infinite theories with their respective $\mathcal{T}_i$-solvers, and pure input formulas. No assumption is made about the deduction capabilities of the $\mathcal{T}_i$-solvers.

Each of the two $\mathcal{T}_i$-solvers interacts directly and only with the CDCL SAT solver, so that there is no direct exchange of information between the $\mathcal{T}_i$-solvers. The CDCL SAT solver is instructed to assign truth values not only to the atoms of φ, but also to the interface equalities e_{ij}'s. Consequently, each assignment enumerated by the SAT solver μ^p is partitioned into three components $\mu^p_{\mathcal{T}_1}$, $\mu^p_{\mathcal{T}_2}$ and μ^p_e, s.t. each $\mu_{\mathcal{T}_i}$ is the set of i-pure literals and μ_e is the set of interface (dis)equalities in μ, so that each $\mu_{\mathcal{T}_i} \cup \mu_e$ is passed to the respective $\mathcal{T}_i$-solver.

An implementation of DTC is based on modern CDCL SMT solvers whose

schema is shown in Algorithm 4 in §2.2.3. Each of the two $\mathcal{T}_i$-solvers interacts with the CDCL engine by exchanging literals via the assignment μ in a stack-based manner. The Algorithm 4 in §2.2.3 is modified to the following extents:

1. The CDCL solver must assigns truth values not only to the atoms in φ, but also to the interface equalities not occurring in φ (the Boolean abstraction and the Boolean refinement are modified accordingly).

2. $\mathcal{T}$-decide_next_branch is modified to select also interface equalities e_{ij}'s not occurring in the formula yet, after the current assignment propositionally satisfies φ.

3. $\mathcal{T}$-deduce is modified to work as follows: for each $\mathcal{T}_i$, $\mu_{\mathcal{T}_i} \cup \mu_e$, is fed to the respective $\mathcal{T}_i$-solver. If both return SAT, then $\mathcal{T}$-deduce returns SAT, otherwise it returns CONFLICT.

4. $\mathcal{T}$-analyze_conflict and $\mathcal{T}$-backtrack are modified so that to use the conflict set returned by one $\mathcal{T}_i$-solver for $\mathcal{T}$-backjumping and $\mathcal{T}$-learning; such conflict sets may contain interface (dis)equalities.

5. Early-pruning and $\mathcal{T}$-propagation are performed: if one $\mathcal{T}_i$-solver performs the e_{ij}-deduction $\mu^* \models_{\mathcal{T}_i} \bigvee_{j=1}^{k} e_j$ s.t. $\mu^* \subseteq \mu_{\mathcal{T}_i} \cup \mu_e$, each e_j being an interface equality, then the Boolean abstraction of the deduction clause $\mu^* \rightarrow \bigvee_{j=1}^{k} e_j$ is learned.

6. If both $\mathcal{T}_i$-solvers are e_{ij}-deduction complete, then an assignment μ which propositionally satisfies φ is found $\mathcal{T}_i$-satisfiable for both $\mathcal{T}_i$'s, and neither $\mathcal{T}_i$-solver performs any e_{ij}-deduction from μ, then the procedure stops returning SAT.

In short, in DTC the embedded CDCL engine not only enumerates truth assignments for the atoms of the input formula, but it also assigns truth values for the interface equalities that the $\mathcal{T}$-solver's are not capable of inferring, and

handles the case-split induced by the entailment of disjunctions of interface equalities in non-convex theories. The rationale is to exploit the full power of a modern CDCL engine by delegating to it part of the heavy reasoning effort previously due to the $\mathcal{T}_i$-solvers.

2.3 Linear Generalized Disjunctive Programming

In this section we provide the necessary background on *Linear Generalized Disjunctive Programming*. We start from recalling the classic *Linear Programming*, which is effectively applied for addressing more complex problems; we proceed by describing *Mixed Integer Linear Programming*, and *Disjunctive Programming* which inspired it and provide approaches for solving it.

2.3.1 Linear Programming

Linear Programming (LP) is the problem of optimizing a linear function over a system of inequalities, formally written as:

$$\min\{\mathbf{cx} \mid \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0\} \tag{2.1}$$

where $\mathbf{A}$ is a matrix, $\mathbf{c}$ and $\mathbf{b}$ are constant vectors and $\mathbf{x}$ a vector of rational variables. The LP problem is solved by a variety of methods, such as the well-known simplex method developed by Dantzig [37] and the more recent interior-point methods; whereas the former searches for solutions on the boundary of the constraints set trying to improve the value of the objective function until the optimal solution is found, the latter searches for solutions in the interior of the constraints set, and only at the end of the search they jump to its boundary. We refer the reader to [26] for details interior-point methods. Although limited to linear inequalities with continuous variables and convex[4] constraints sets, LP is practically used as relaxation method [61].

[4] A shape or set is *convex if for any two points that are part of the shape, the whole connecting line segment is also part of the shape.*

LP can be seen as a special case of *convex optimization* [26], a class of optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, n \end{array} \tag{2.2}$$

where the functions $f_i : \mathbb{Q}^n \leftarrow \mathbb{Q}$ are convex[5]. Convex optimization problems are effectively solved by interior-point methods and can be applied to non-convex problems. For instance, they can be used for computing upper and lower bounds on the optimal solution quality, and as relaxation methods by replacing non-convex constraints with looser, but convex, constraints.

2.3.2 Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) is an extension of Linear Programming (LP) involving both discrete and continuous variables [61]. MILP problems have the following form:

$$\min\{\mathbf{cx} : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x}_j \in \mathbb{Z}\, \forall j \in I\} \tag{2.3}$$

that is the form of a LP problem augmented with an integrality requirement on the $\mathbf{x}$ variables in the set I. We call a *LP relaxation* a MILP problem where the integrality constraint on the variables $\mathbf{x}_j$, for all $j \in I$, is dropped.

Special cases of MILP are:

- *Integer Linear Programming* (ILP), which refers to a MILP problem in which all variables are constrained to be integers;
- *0-1 Mixed Linear Programming*, which is a special case of MILP where non-rational variable are restricted to be binary.

[5] A function f is convex if it satisfies $f(\alpha x + \beta y) \leq \alpha f(x) + \beta f(y)$ for all $x, y \in \mathbb{Q}^n$ and all $\alpha, \beta \in \mathbb{Q}$ with $\alpha + \beta = 1$ and $\alpha \geq 0, \beta \geq 0$.

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 proposed by Padberg and Rinaldi [73].

Branch-and-bound search In its basic version of Land and Doig [57], it iteratively partitions the solution space of the original MILP problem into subproblems and solves their LP relaxation until all variables are integral in the LP relaxation. The solutions that are infeasible in the original problem guide the search in the following way:

- if the optimal solution of a LP relaxation is greater than or equal to the optimal solution found so far, the search backtracks, since there cannot exist a better solution;
- if a variable $\mathbf{x}_j$ is required to be integral in the original problem, the rounding of its value a in the LP relaxation suggests how to branch by requiring $\mathbf{x}_j \leq \lfloor a \rfloor$ in one branch and $\mathbf{x}_j \geq \lfloor a \rfloor + 1$ in the other.

Cutting planes MILP problems can be solved by simply finding the *convex hull*[6] of its (mixed-)integer solutions. The cutting plane algorithm was proposed by Gomory [44] for solving IP and requires to interactively solve the *separation problem*: given a MILP problem and a solution $\mathbf{x}^*$ of the LP relaxation which is not feasible for it, its goal is to find a linear inequality $\mathbf{ax} \geq \mathbf{b}$ which is satisfied by all feasible solutions of the MILP problem, while it is violated by $\mathbf{x}^*$, i.e. $\mathbf{ax}^* < \mathbf{b}$. Any inequality solving the separation problem is called a *cutting plane* (or *cut* for short).

Cutting planes (e.g. Gomory mixed-integer and lift-and-project cuts, see [61]) can be inferred and added to the original MILP problem and its subproblems in order to cut away non-integer solutions of the LP relaxation

[6]For any subset C of the plane (e.g. set of points, rectangle, simple polygon), its *convex hull is the smallest convex set that contains* C.

and obtain a tighter relaxation (which better approximates the convex hull).

Modern MILP exploits effective evolutions of cutting planes, branching heuristics and preprocessing (see e.g. [61] for details on characteristics of current MILP solvers).

Notice that SAT techniques have also been incorporated into these procedures for MILP (see [4]).

2.3.3 Disjunctive Programming

Disjunctive Programming (DP) problems are LP problems where linear constraints are connected by the logical operations of conjunction and disjunction (paradigm proposed by Balas [13]). Typically, the constraint set is expressed by a disjunction of linear systems:

$$\bigvee_{i \in I} (\mathbf{A}^i \mathbf{x} \geq \mathbf{b}^i) \tag{2.4}$$

or, alternatively, as:

$$(\mathbf{A}\mathbf{x} \geq \mathbf{b}) \wedge \bigwedge_{j=1}^{t} \bigvee_{k \in I_j} (\mathbf{c}^k \mathbf{x} \geq d^k) \tag{2.5}$$

where $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ consists of the inequalities common to $\mathbf{A}^i \mathbf{x} \geq \mathbf{b}^i$ for $i \in I$, I_j for $j = 1, \ldots, t$ contains one inequality of each system $\mathbf{A}^i \mathbf{x} \geq \mathbf{b}^i$ and t is the number of sets I_j having this property. DP problems are effectively solved by the lift-and-project approach, which combines a family of cutting planes, called lift-and-project cuts, and the branch-and-bound schema (see, e.g., [15]).

It is easy to see that MILP is a specialization of DP. Given a MILP problem (as in Equation 2.3) having the condition that variable $x_j \in I$ is constrained to be integer in the range $[0, n]$, we can re-write the integrality condition of the MILP problem as a disjunction of constraints $\bigvee_{j=0}^{n} (x_i = j)$.

Disjunctive Programming has given very important contributions for Integer Linear Programming and 0-1 Mixed Linear Programming, that have been so far its main application (see, e.g., [14, 15, 16]):

2.3.4 Linear Generalized Disjunctive Programming

Closest to our domain is *Linear Generalized Disjunctive Programming* (LGDP), a generalization of DP which has been proposed by Raman and Grossmann in [75] 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 [75, 93, 77, 78]; 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 [75]:

$$
\begin{array}{lll}
\min & \sum_{\forall k \in K} \mathbf{z}_k + \mathbf{dx} & \\
\text{s.t.} & \mathbf{Bx} \leq \mathbf{b} & \\
& \bigvee_{j \in J_k} \begin{bmatrix} Y^{jk} \\ \mathbf{A}^{jk}\mathbf{x} \geq \mathbf{a}^{jk} \\ \mathbf{z}_k = c^{jk} \end{bmatrix} & \forall k \in K \\
& \phi & \\
& 0 \leq \mathbf{x} \leq \mathbf{e} & \\
& \mathbf{z}_k \in \mathbb{R}^1_+, Y^{jk} \in \{True, False\} & \forall j \in J_k, \forall k \in K
\end{array}
\tag{2.6}
$$

where $\mathbf{x}$ is a vector of rational variables, $\mathbf{z}$ is a vector representing the cost assigned to each disjunction and c^{jk} are fixed charges, $\mathbf{e}$ is a vector of upper bounds for $\mathbf{x}$ and Y^{jk} are Boolean variables. Each disjunction $k \in K$ is composed by two or more disjuncts $j \in J_k$ that contain a set of linear constraints

$\mathbf{A}^{jk}\mathbf{x} \geq \mathbf{a}^{jk}$, where $(\mathbf{A}^{jk}, \mathbf{a}^{jk})$ is a $m_{jk} \times (n+1)$ matrix, for all $j \in J_k$ and $k \in K$, that are connected by the logical OR operator. Boolean variables Y^{jk} and logic propositions ϕ in terms of Y^{jk} (expressed in Conjunctive Normal Form) represents discrete decisions. Only the constraints inside the disjuncts $j \in J_k$, where Y^{jk} is true, are enforced. $\mathbf{Bx} \leq \mathbf{b}$, where $(\mathbf{B}, \mathbf{b})$ is a $m \times (n+1)$ matrix, are constraints that must hold regardless of disjuncts.

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.

big-M reformulation Boolean variables Y^{jk} and logic constraints ϕ are respectively replaced by binary variables $\mathbf{Y}_{jk}$ and linear inequalities as follows [75]:

$$\begin{array}{lll} \min & \sum_{\forall k \in K} \sum_{\forall j \in J_k} c^{jk}\mathbf{Y}_{jk} + \mathbf{dx} & \\ \text{s.t.} & \mathbf{Bx} \leq \mathbf{b} & \\ & \mathbf{A}^{jk}\mathbf{x} - \mathbf{a}^{jk} \leq \mathbf{M}^{jk}(1 - \mathbf{Y}_{jk}) & \forall j \in J_k, \forall k \in K \\ & \sum_{\forall j \in J_k} \mathbf{Y}_{jk} = 1 & \forall k \in K \\ & \mathbf{DY} \leq \mathbf{D}' & \\ & \mathbf{x} \in \mathbb{R}^n_+, \mathbf{Y}_{jk} \in \{0,1\} & \forall j \in J_k, \forall k \in K \end{array} \tag{2.7}$$

where $\mathbf{M}^{jk}$ are the "big-M" parameters that makes 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.

convex hull reformulation Boolean variables Y^{jk} are replaced by binary variables $\mathbf{Y}_{jk}$ and variables $\mathbf{x} \in \mathbb{R}^n$ are disaggregated into new variables $\mathbf{v} \in \mathbb{R}^n$ (using convex hull constraints for each disjunction [13, 75]) in

the following way:

$$
\begin{array}{llr}
\min & \sum_{\forall k \in K} \sum_{\forall j \in J_k} c^{jk} \mathbf{Y}_{jk} + \mathbf{dx} & \\
\text{s.t.} & \mathbf{Bx} \leq \mathbf{b} & \\
& \mathbf{A}^{kj} \mathbf{v}^{jk} \leq \mathbf{a}^{jk} \mathbf{Y}_{jk} \qquad \forall j \in J_k, \forall k \in K & \\
& \mathbf{x} = \sum_{\forall j \in J_k} \mathbf{v}_{jk} \qquad \forall k \in K & (2.8) \\
& \mathbf{v}_{jk} \leq \mathbf{Y}_{jk} \mathbf{e}^{jk} \qquad \forall j \in J_k, \forall k \in K & \\
& \sum_{\forall j \in J_k} \mathbf{Y}_{jk} = 1 \qquad \forall k \in K & \\
& \mathbf{DY} \leq \mathbf{D}' & \\
& \mathbf{x}, \mathbf{v} \in \mathbb{R}^n_+, \mathbf{Y}_{jk} \in \{0, 1\} \qquad \forall j \in J_k, \forall k \in K &
\end{array}
$$

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

In comparing BM and CH reformulations, two facts can be observed:

1. the relaxation of BM is often weak causing a higher number of nodes examined in the branch-and-bound search;

2. the disaggregated variables and new constraints increase the size of the reformulation leading to a high computational effort.

In order to overcome these issues, Sawaya and Grossman [77] proposed a cutting plane method (based on the lift-and-project cutting planes developed by [16]) that consists in computing a sequence of BM relaxations with cutting planes that are obtained by solving a LP separation problem and finding the most violated constraint of the convex hull that is projected onto the space of the original variables of the BM relaxation, until the optimal solution for the original MILP is found or until there is no improvement within a specified tolerance ϵ; in this case it switches to the branch-and-bound method for solving the resulting BM relaxation with all the cutting planes that have been generated. We refer the reader to [77] for a more detailed explanation.

Sawaya and Grossman provided an evaluation of the presented algorithm on three different problems: strip-packing, retrofit planning and zero-wait job-shop scheduling problems.

Chapter 3

State of the Art and Related Work

This chapter aims at presenting a brief survey of the literature on the topic of optimization in SAT and SMT.

Disclaimer. The material presented in §3.1.1 is standard in SAT and it is mostly taken from [59, 76].

3.1 State of the Art

This section describes optimization problems and procedures for solving them available in the literature of SAT and SMT.

3.1.1 Optimization in SAT: MaxSAT and Pseudo-Boolean Optimization

Two optimization problems can be seen as generalization of SAT: *Maximum Satisfiability* [59] and *Pseudo-Boolean Optimization* [76].

Maximum Satisfiability (MaxSAT) is the problem of finding a truth assignment μ that maximizes the number of satisfied clauses in a given (propositional) CNF formula φ. Notice that MaxSAT can be also seen as the problem of finding the minimum number of unsatisfied clauses in φ.

MaxSAT has three important extensions:

- *Partial MaxSAT*: given a CNF formula where some clauses are declared to be "soft" and the rest are declared to be "hard", it is the problem of finding an assignment that satisfies all the hard clauses and minimize the number of unsatisfied soft clauses.
- *Weighted MaxSAT*: given a weighted CNF formula, i.e. a CNF formula where one weight (i.e. a positive number) w_i is assigned to each clause C_i, it is the problem of minimizing the sum of weights of unsatisfied clauses.
- *Weighted Partial MaxSAT*: combines the first two problems whose goal is to minimize the sum of weights of unsatisfied soft clauses.

Notice that MaxSAT can be seen as Weighted MaxSAT where clauses have weight 1, and as Partial MaxSAT where all the clauses are declared to be soft.

Pseudo-Boolean Optimization (PBO) is the optimization version of Pseudo-Boolean Solving (PBS), a generalization of SAT whose goal is to decide the satisfiability of conjunctions of pseudo-Boolean constraints, i.e. constraints of the form:

$$\sum_{j=1}^{n} a_j l_j \geq b \tag{3.1}$$

where a_j, b are positive integer constants and l_j are Boolean literals. Notice that pseudo-Boolean constraints are generalization of Boolean clauses, in fact the pseudo-Boolean constraint $\sum_i x_i \geq 1$ can be equivalently written as the clause $\bigvee_i x_i$.

PBO aims at finding a valuation of variables such that all constraints are satisfied and the value of a given cost function is minimized:

$$\begin{array}{ll} \text{minimize} & \sum_j c_j x_j \\ \text{subject to} & \bigwedge_i \sum_j a_{i,j} x_j \geq b_i \end{array} \tag{3.2}$$

where $c_j, a_{i,j}, b_i \in \mathbb{Z}$ and x_j are pseudo-Boolean variables. PBO problems can consider many cost functions that define a ranking of solutions based on some criteria.

Exact algorithms for solving SAT optimization problems benefits from many effective SAT solving techniques (e.g. the CDCL schema described in §2.1.1) and are based the following approaches:

Linear search which explores the space of all possible solutions (i.e. truth assignments) of the input formula φ and, instead of stopping when a solution is found, it adds a new constraint to φ and restarts the search. Since the current solution (namely the number of unsatisfied clauses of φ under μ) becomes an upper bound ub on the optimal solution quality, the next computed solution (if any) will have a higher quality than the previous one. If the resulting formula is unsatisfiable, the optimal solution is given by the last stored ub.

This schema was successfully implemented for PBO by integrating the search for optimum inside a classic CDCL pseudo-Boolean solver (see [21]).

Branch and bound which extends the linear-search schema to avoid redundant search. It explores the solutions space of φ pruning the search at every partial solution η of φ whose (estimation of) lower bound lb exceeds the current upper bound ub on the optimal solution quality. Otherwise, it goes on with the search to find a better solution μ by extending the current partial one (i.e. $\mu \supseteq \eta$).

Modern and competitive MaxSAT solvers integrates the CDCL schema with branch and bound enhanced by powerful inference techniques, effective lower bound computation methods, clever variable selection heuristics and efficient data structures (e.g. [5, 95, 48]). Also state-of-the-art PBO

solvers are based on the branch-and-bound schema [58] and take advantage of the experience in Integer Linear Programming in order to estimate lower bounds lb and prune the search [76].

Binary search which is based on the idea of restricting the search within the interval $[\mathsf{lb}, \mathsf{ub}]$, where ub is an upper bound and lb a lower bound on the solution quality of φ, and bisect it (e.g. $M = (\mathsf{lb}{+}\mathsf{ub})/2$). At each step, the optimal solution is searched in the interval $[\mathsf{lb}, M]$. If a solution is found, the search goes on in the interval $[\mathsf{lb}, C-1]$, where C is the cost of the solution found. Otherwise, the search proceeds in the interval $[M+1, \mathsf{ub}]$. Clearly, the branch-and-bound schema can be defined as the binary-search schema where $M = \mathsf{ub}$. Even if binary search usually spends much more time than the two previous approaches to find the first solution, it converges faster toward the optimal solution [76]. Nevertheless, this approach may be difficult to implement since every learned constraint that was inferred from the search bounds must be forgotten (or better, reused) from one step to another.

A common approach to PBO is to translate pseudo-Boolean constraints into propositional clauses and then apply either a linear-search schema or a binary-search schema [76].

Core-guided which are based on unsatisfiable core extraction (e.g. [49, 67] for MaxSAT). The basic idea is that an unsatisfiable formula φ will contain one or more unsatisfiable core but no satisfiable subset of φ can contain any complete cores; thus, any solution must leave unsatisfied at least one clause from every core. This approach reduces the search space by identifying unsatisfiable cores of φ and only considering clauses within those cores as potential "removals".

Inference which applies ad-hoc inference rules while performing a standard branch-and-bound search (e.g. [7] for MaxSAT).

When it is sufficient to find near-optimal solutions rather then optimal ones, *local search* is a fast and quite effective approach [53]. Most of exact algorithms for MaxSAT exploit them in the branch-and-bound schema in order to compute the initial upper bound [48].

Approximations algorithms are good alternatives to local search algorithms. Though slower, they can give approximate solutions in polynomial time and guarantee the quality of the solutions found. Over the last years, there have been proposed several approximation algorithms for MaxSAT. At the beginning, the greedy approach was proposed [56, 42] but recently the most promising approach is based on semidefinite programming [43]. Greedy approximations algorithms can also be used in PBO solvers for computing lower bounds on the optimal solution quality [22].

Even thought there exist efficient approaches for solving the optimization versions of SAT and PBS, MaxSAT and PBO only involve cost functions that are expressed in terms of Boolean conditions over Boolean atoms.

3.1.2 SMT with Pseudo-Boolean Costs and MaxSMT

The idea of optimization in SMT was first introduced by Nieuwenhuis and Oliveras [70], who presented a very-general logical framework of "SMT with progressively stronger theories" where, during the search process, the initial background first-order theory $\mathcal{T}_0$ is progressively strengthened, raising the bar every time a new optimal solution is found. The theory $\mathcal{T}_0$ is extended with other first-order theories $\mathcal{T}_i$, for $i = 1, \ldots, n$, resulting in the theory $\mathcal{T}' = \bigcup_{i=0}^{n} \mathcal{T}_i$, each $\mathcal{T}_i$ forcing the search for a solution which improves the current optimum.

They use the proposed SMT procedure for solving two optimization problems, Weighted MaxSAT and Weighted MaxSMT (the SMT version of the former problem), by implementing a branch-and-bound approach within the CDCL schema (described in §2.2.3). The language of the extended theory $\mathcal{T}_i$ allows for expressing cost functions and bounds on the cost function value. In par-

ticular, a cost function is modelled using atoms of the form $(k_0 + \ldots + k_m \leq c)$, $(p_i \rightarrow (k_i = w_i))$ and $(\neg p_i \rightarrow (k_i = 0))$. Since the values of the variables k_i depend on the truth values assigned to the Boolean atoms p_i, the cost function can be seen as a set of logical conditions determining a given cost. Whenever a local optimal solution u_i is found, the theory $\mathcal{T}'$ is strengthened with a relation $c < u_i$ causing a conflict in the current status. Consequently, thanks to the CDCL approach, the search is pruned and the search restarts.

Cimatti et al. [31] 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 SMT with PB constraints and to minimize PB cost functions. More specifically, the language of the theory $\mathcal{C}$ allows for expressing (multiple) cost functions that have the following form:

$$cost^i = \sum_{j=0}^{N_i} ite(A^i_j, d^i_j, 0) \tag{3.3}$$

where ite is a function that returns the integer constant d^i_j if the Boolean atoms A^i_j is assigned to true, 0 otherwise. The language consists in fresh variables c^i that represent the values of the cost functions $cost^i$ and two predicates: the binary predicate $BC(c^i, d)$ states that c^i must be less than or equal to the integer constant d; the ternary predicate $IC(c^i, j, d^i_j)$ states that the solution quality c^i is increased by the integer constant d^i_j (where j is the index in the sum (3.3)). BC, IC and Boolean predicates are combined together in order to define Boolean conditions that determine different cost increases and express (upper and lower) bounds on the solution quality. Notice that the proposed decision procedure for $\mathcal{C}$ can be integrated into every lazy SMT solver, resulting in a solver for $\mathcal{C} \cup \mathcal{T}$. The cost decision problem addressed by Cimatti et al. is defined by a SMT formula φ, a set of cost function of the form (3.3), and a upper bound and a lower bound, $\mathsf{ub}^i, \mathsf{lb}^i$ respectively, on the value of the ith cost function, for

every i. They express this problem as an SMT($\mathcal{T} \cup \mathcal{C}$) problem by means of the formula

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

and then solve the problem by checking the satisfiability of $\varphi_{\mathcal{C}}$. The authors also deal with the problem of finding a satisfiable assignment for $\varphi_{\mathcal{C}}$ that minimizes a given cost function by seeing it as a finite sequence of decision problems.

Two procedures are described in the paper. Both of them exploit the SMT solver for $\mathcal{C} \cup \mathcal{T}$ in order to implement the branch-and-bound schema and the binary-search schema. Third, the paper show how to encode the Pseudo-Boolean Optimization problem and Weighted Partial MaxSMT (i.e. the SMT version of Weighted Partial MaxSAT) into SMT with Boolean cost functions and an provide experimental evaluation on them.

The SMT solvers YICES [2] and Z3 [3] also provide support for MaxSMT, although there is no publicly-available document describing the procedures used there.

Ansótegui et al. [6] described the evaluation of an implementation of a MaxSMT procedure based on YICES, although this implementation is not publicly available. The proposed evaluation considers several SMT($\mathcal{LA}(\mathbb{Q})$) encodings of the Resource-Constrained Project Scheduling Problem (RCPSP) and applies three different solving approaches. The first approach calls the SMT solver iteratively as a black-box performing a binary search strategy; the second one uses the SMT solver directly for solving a Weighted MaxSMT representation of RCPSP; the last one exploits YICES's API for implementing an algorithm based on the detection of unsatisfiable cores.

Cimatti et al. [32] presented a "modular" approach for (weighted partial) MaxSMT, combining a lazy SMT solver with a purely-propositional MaxSAT solver, which can be used as black-boxes. The author's idea is to make the solvers exchange information iteratively: the SMT solver produces an increas-

ing set of theory lemmas which are fed to the MaxSAT solver, who progressively refines an approximation of the final subset of the (soft) clauses, which is eventually returned as output. The proposed approach is implemented on top of the MathSAT5 SMT solver and of a selection of external MaxSAT solvers.

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 thesis (see §4).

3.2 Other Forms of Optimization in SMT

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. [40] addressed the problem of finding *minimum-cost assignments*[1], i.e. *partial* models for quantified first-order formulas modulo theories, which minimize the number of free variables which are assigned a value from the domain. Quoting an example from [40], 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 (partial) assignment satisfying φ which assigns less then one variable. The authors proposed a general branch-and-bound procedure addressing the problem for every decidable theory $\mathcal{T}$ admitting quantifier elimination, and some enhancements for computing good variable orders and initial cost bounds, and pruning the search space. They also presented an experimental evaluation of an implementation for $\mathcal{LA}(\mathbb{Z})$ and $\mathcal{EUF}$ into the MISTRAL tool.

Manolios and Papavasileiou [63] 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

[1] In propositional logic, the *minimum-cost assignments* problem is commonly known as the *minimum prime implicants* problem [64].

stably-infinite theories $\mathcal{T}$. Notice that the approach of [63] 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 [63].) Also, the objective function is defined on the Integer domain. they presented a general algorithm by integrating the Branch&Cut ILP method with $\mathcal{T}$-specific decision procedures, and implemented it into the INEZ tool.

We understand that neither of the above-mentioned works can handle the problem addressed in this thesis, and vice versa.

3.3 A Very-Recent OMT($\mathcal{LA}(\mathbb{Q})$) Tool

Closest in spirit to our work is a very-recent paper from Li et al. [60], presented last January at POPL 2014 conference. It extends the OMT($\mathcal{LA}(\mathbb{Q})$) problem we introduced in [80] (see §4) to "multiple-objectives", by considering contemporarily a set of *independent* cost variables for the input formula φ, namely $\{\mathsf{cost}_1, ..., \mathsf{cost}_k\}$, so that the problem consists in enumerating k independent models for φ, each minimizing one specific cost_i. [2] (Intuitively, enumerating such models is in general more efficient than solving one optimization problem at the time, because it allows for sharing the SMT search steps among different cost objectives.) Then [60] proposes a multiple-objective generalization of our linear-search algorithm of [80], and presents an implementation called SYMBA on top of the Z3 SMT solver [3]. Similarly to our work of [80], the multiple-objective minimization is performed in two alternative ways: an "offline" version, in which a sequence of black-box calls to the SMT solver allows for finding progressively-better solutions along one objective direction, and a more efficient "inline" version, in which the simplex algorithm inside the $\mathcal{LA}(\mathbb{Q})$-solver of Z3 is modified to find the optimum, as in our inline version described in [80] and in §5.2. SYMBA also makes use of an effective ad-hoc heuristic for checking infinite costs (whilst our tool, OPTIMATHSAT, applies

[2]More precisely, in [60] the objectives are *maximized*, but the problem is dual.

the technique used in the standard simplex algorithm). In §6.6 we empirically compare their approach with ours. Notice that, if SYMBA is restricted to work on single objectives, we see no substantial algorithmic difference between the two "inline" procedures, apart from the previously mentioned heuristic and the fact that they are built on top of Z3 and MATHSAT respectively.

Part II

Novel Contributions

Chapter 4

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

We extend the standard SMT framework for solving optimization problems, resulting in a *Optimization Modulo Theory* (*OMT*) framework. The chapter is organized as follows: in §4.1 we formally define the OMT problem; in §4.2 we introduce the necessary formal foundations for solving OMT; in §4.2.1 we show how it generalizes many known optimization problems from the literature.

Disclaimer. The work presented in this chapter (and also in §5 and §6) was done in collaboration with Roberto Sebastiani and was presented in [80, 81].

4.1 Basic Definitions and Notation

In very-general terms, we define *Optimization Modulo Theory* (*OMT*) as follows.

Definition 4.1 (OMT($\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$))**.** Let φ be a ground formula in some background theory $\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$, where $\mathcal{T}_{\preceq}$ has some *total order* $\preceq$ over its domain values, and let cost be a $\mathcal{T}_{\preceq}$-variable occurring in φ. (The other theories $\mathcal{T}_i$'s may possibly be empty.) We call *Optimization Modulo* $\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$, written *OMT*($\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$), the problem of finding a model $\mathcal{I}$ for φ whose value of cost is minimum according to $\preceq$.

In this thesis we consider signature-disjoint stably-infinite theories with equality ("Nelson-Oppen theories" introduced in §2.2.4) and we restrict our interest to $\mathcal{LA}(\mathbb{Q})$ as $\mathcal{T}_{\preceq}$. Thus we assume $\mathcal{T}$ to be some stably-infinite theory with equality, s.t. $\mathcal{LA}(\mathbb{Q})$ and $\mathcal{T}$ are signature-disjoint. ($\mathcal{T}$ can be itself a combination of theories.)

Definition 4.2 (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 $\mathcal{I}$ 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 w.l.o.g. that all $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ formulas are *pure*. We recall from §2.2.1 that an atom in a ground $\mathcal{T}_1 \cup \mathcal{T}_2$ formula is said to be $\mathcal{T}_i$-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 iff all its atoms are either $\mathcal{T}_1$-pure or $\mathcal{T}_2$-pure. Although the purity assumption is not necessary (see [19]), it much simplifies the explanation, since it allows us for speaking of "$\mathcal{LA}(\mathbb{Q})$-atoms" or "$\mathcal{T}$-atoms" without further specifying. Moreover, every non-pure formula can be easily purified [68]. We also assume w.l.o.g. that all $\mathcal{LA}(\mathbb{Q})$-atoms containing the variable cost are in the form $(t \bowtie \mathsf{cost})$, s.t. $\bowtie \in \{=, \leq, \geq, <, >\}$ and cost does not occur in t.

Definition 4.3 (Bounds and range for cost)**.** 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.

Notice that we adopt the convention of defining upper bounds to be strict and lower bounds to be non-strict 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, whilst a lower bound $\neg(\mathsf{cost} < c)$ derives either from the user's knowledge (e.g. "the cost cannot be lower than zero") of from the fact that the formula $\varphi \wedge (\mathsf{cost} < c)$ has been previously found $\mathcal{T}$-unsatisfiable whilst φ has not.

4.2 Theoretical Results

We present here the theoretical foundations of the procedures for solving the OMT problem we shall propose in §5.

The following facts follow straightforwardly from Definition 4.2.

Proposition 4.1. *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 $\mathsf{min}_{\mathsf{cost}}(\varphi) < +\infty$.

Theorem 2.1 in §2.2 is the theoretical foundation of the lazy SMT approach (see §2.2.3), where a CDCL SAT solver enumerates a complete collection $\mathcal{M}$ of truth assignments as above, whose $\mathcal{T}$-satisfiability is checked by a $\mathcal{T}$-Solver. Notice that in Theorem 2.1 the theory $\mathcal{T}$ can be any combination of theories $\mathcal{T}_i$, including $\mathcal{LA}(\mathbb{Q})$. Here we extend Theorem 2.1 to OMT$(\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T})$ as follows.

Lemma 4.1. *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 4.2. *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 s.t. $\mu \models_p \varphi$. Then there exists at least one $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable* **total** *assignment η s.t. $\mu \subseteq \eta$, $\eta \models_p \varphi$, 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

$$\eta \stackrel{\text{def}}{=} \bigwedge_{\substack{\psi_i\in Atoms(\varphi)\\ \mathcal{I}\models\psi_i}} \psi_i \wedge \bigwedge_{\substack{\psi_i\in Atoms(\varphi)\\ \mathcal{I}\models\neg\psi_i}} \neg\psi_i \tag{4.1}$$

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 $\eta \models_p \varphi$. □

Theorem 4.1. *Let φ be a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-formula and let $\mathcal{M} \stackrel{\text{def}}{=} \{\mu_1, \ldots, \mu_n\}$ 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 4.2 and Theorem 2.1. 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 absurd, suppose exists $\mu \in \mathcal{M}$ s.t. $\mathsf{min}_{\mathsf{cost}}(\mu) < \mathsf{min}_{\mathsf{cost}}(\varphi)$. By Proposition 4.1, μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$ satisfiable. By Lemma 4.2, there exists a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable total assignment η s.t. $\mu \subseteq \eta$, $\eta \models_p \varphi$, and $\mathsf{min}_{\mathsf{cost}}(\mu) = \mathsf{min}_{\mathsf{cost}}(\eta)$. By lemma 4.1, $\mathsf{min}_{\mathsf{cost}}(\eta) \geq \mathsf{min}_{\mathsf{cost}}(\varphi)$, 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 4.1 we have that $\mathsf{min}_{\mathsf{cost}}(\varphi) = min_{\eta_i\in\mathcal{E}}\mathsf{min}_{\mathsf{cost}}(\eta_i)$. Let $\eta \in \mathcal{E}$ s.t. $\mathsf{min}_{\mathsf{cost}}(\varphi) = \mathsf{min}_{\mathsf{cost}}(\eta) < +\infty$. Hence η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. Thus, there exists $\mu \in \mathcal{M}$ s.t. $\mu \subseteq \eta$. μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable since η is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable. From Proposition 4.1, $\mathsf{min}_{\mathsf{cost}}(\mu) \leq \mathsf{min}_{\mathsf{cost}}(\eta)$, hence $\mathsf{min}_{\mathsf{cost}}(\mu) \leq \mathsf{min}_{\mathsf{cost}}(\varphi)$. Thus the thesis holds.

□

(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 4.1 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.

Consider the definitions of *interface variables* and *interface equalities* introduced in §2.2.1. (We recall that an *interface variable* is a variable occurring in both $\mu_{\mathcal{T}_1}$ and $\mu_{\mathcal{T}_2}$, and an *interface equality* is an equality $(x_i = x_j)$ on interface variables.) As common practice (see e.g. [86]) 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 restrict w.l.o.g. 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 $\{(x_i = x_j), (x_j = x_i)\}$.

Notation-wise, in what follows we use the pedexes e, d, i in "$\mu_{...}$", like in "μ_{ed}", to denote conjunctions of equalities, disequalities and inequalities be-

tween interface variables respectively.

Theorem 4.2 ([86]). *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably-infinite theories with equality and disjoint signatures; let $\mu \stackrel{\text{def}}{=} \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ be a conjunction of $\mathcal{T}_1 \cup \mathcal{T}_2$-literals s.t. 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 $e() \subseteq \mathcal{IE}(\mu)$ over the interface variables of μ and the corresponding total truth assignment μ_{ed} to the interface equalities over μ:* [1]

$$\mu_{ed} \stackrel{\text{def}}{=} \mu_e \wedge \mu_d, \;\; s.t.\; \mu_e \stackrel{\text{def}}{=} \bigwedge_{(x_i,x_j)\,\in\, e()} (x_i = x_j), \;\; \mu_d \stackrel{\text{def}}{=} \bigwedge_{(x_i,x_j)\,\notin\, e()} \neg(x_i = x_j) \tag{4.2}$$

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

Theorem 4.2 is the theoretical foundation of, among others, the *Delayed Theory Combination* SMT technique for combined theories (see 2.2.4), 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 which propositionally satisfies it. μ can be written as $\mu \stackrel{\text{def}}{=} \mu_{\mathbb{B}} \wedge \mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{\mathcal{T}}$, s.t. $\mu_{\mathbb{B}}$ is a consistent conjunction of Boolean literals, $\mu_{\mathcal{LA}(\mathbb{Q})}$ and $\mu_{\mathcal{T}}$ are $\mathcal{LA}(\mathbb{Q})$-pure and $\mathcal{T}$-pure conjunctions of literals respectively. (Notice that the $\mu_{\mathcal{B}}$ 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 4.4. 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, s.t. $\mu_{\mathbb{B}}$ is a consistent conjunction of Boolean literals, $\mu_{\mathcal{LA}(\mathbb{Q})}$ 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 $\mathcal{EX}_{ed}(\mu) \stackrel{\text{def}}{=}$

[1] μ_{ed} is called an *arrangement* in [86].

$\{\eta_1, ..., \eta_n\}$ of all possible assignments in the form $\mu \wedge \mu_{ed}$, where μ_{ed} is in the form (4.2), for every equivalence class $e()$ in $\mathcal{IE}(\mu)$.

Theorem 4.3. *Let μ be as in Definition 4.4. Then*

(a) $\mathsf{min}_{\mathsf{cost}}(\mu) = min_{\eta \in \mathcal{EX}_{ed}(\mu)} \mathsf{min}_{\mathsf{cost}}(\eta)$

(b) forall $\eta \in \mathcal{EX}_{ed}(\mu)$,

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

Proof.

(a) Let

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

where 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 4.2, 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 $\mu' = \mu'_{\mathcal{T}} \wedge \mu'_{\mathcal{LA}(\mathbb{Q})}$ s.t. $\mu'_{\mathcal{T}} = \mu_{\mathcal{T}}$ 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 4.2, from which we have that η' is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable, so that η has a model $\mathcal{I}$ s.t. $\mathcal{I}(\mathsf{cost}) = c$. 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 4.2, 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 $\mathcal{I}$ s.t. $\mathcal{I}(\mathsf{cost}) = c$. Thus, we have that $\mathsf{min}_{\mathsf{cost}}(\eta) \geq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed})$.

$\square$

We notice that, at least in principle, computing $\mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ed})$ is an operation which can be performed by standard linear-programming techniques (see §5). Thus, by combining Theorems 4.1 and 4.3 we have a general method for computing $\mathsf{min}_{\mathsf{cost}}(\varphi)$ also in the general case of non-empty theory $\mathcal{T}$.

In practice, however, it is often the case that $\mathcal{LA}(\mathbb{Q})$-solvers/optimizers cannot handle efficiently negated equalities like, e.g., $\neg(x_i = x_j)$ (see [41]). Thus, a technique which is adopted by most SMT solver 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 4.5. Let μ be as in Definition 4.4. We call the *complete set of edi-extensions of* μ the set $\mathcal{EX}_{edi}(\mu) \stackrel{\text{def}}{=} \{\rho_1, ..., \rho_n\}$ of all possible truth assignments in the form $\mu \wedge \mu_{ed} \wedge \mu_i$, where μ_{ed} is as in Definition 4.4 and μ_i is a total truth assignment to the atoms $(x_i < x_j)$, $(x_i > x_j)$ s.t. $(x_i = x_j) \in \mathcal{IE}(\mu)$ and $\mu_{ed} \wedge \mu_i$ are $\mathcal{LA}(\mathbb{Q})$-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 4.4. *Let μ be as in Definition 4.4. Then*

(a) μ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable iff 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) forall $\rho \in \mathcal{EX}_{edi}(\mu)$, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable iff $\mu_{\mathcal{T}} \wedge \mu_{ed}$ is $\mathcal{T}$-satisfiable and $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i$ is $\mathcal{LA}(\mathbb{Q})$-satisfiable.

(d) forall $\rho \in \mathcal{EX}_{edi}(\mu)$,

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

Proof. Let

$$\mu^* \stackrel{\text{def}}{=} \mu \wedge \bigwedge_{(x_i = x_j) \in \mathcal{IE}(\mu)} \begin{pmatrix} ((x_i = x_j) \vee (x_i < x_j) \vee (x_i > x_j)) \wedge \\ (\neg(x_i = x_j) \vee \neg(x_i < x_j)) \wedge \\ (\neg(x_i = x_j) \vee \neg(x_i > x_j)) \wedge \\ (\neg(x_i < x_j) \vee \neg(x_i > x_j)) \end{pmatrix} \quad (4.3)$$

All clauses in the right conjuncts in (4.3) 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 2.1, μ^* is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable iff 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, s.t. it is the $\mathcal{LA}(\mathbb{Q})$-pure part of ρ (namely, $\rho_{\mathcal{LA}(\mathbb{Q})}$). Thus, by Theorem 4.2, ρ is $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-satisfiable iff $\overbrace{\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i}^{\rho_{\mathcal{LA}(\mathbb{Q})}} \wedge \overbrace{\mu_e \wedge \mu_d}^{\mu_{ed}}$ 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 iff $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_i \wedge \mu_e$ 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 $\mu' = \mu'_{\mathcal{T}} \wedge \mu'_{\mathcal{LA}(\mathbb{Q})}$ s.t. $\mu'_{\mathcal{T}} = \mu_{\mathcal{T}}$ 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 $\mathcal{I}$ s.t. $\mathcal{I}(\mathsf{cost}) = c$. Therefore we have that $\mathsf{min}_{\mathsf{cost}}(\rho) \leq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i)$.

$\geq$ **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 $\mathcal{I}$ s.t. $\mathcal{I}(\mathsf{cost}) = c$. Therefore we have that $\mathsf{min}_{\mathsf{cost}}(\rho) \geq \mathsf{min}_{\mathsf{cost}}(\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_e \wedge \mu_i)$.

□

Thus, by combining Theorems 4.1 and 4.4 we have a general method for computing $\mathsf{min}_{\mathsf{cost}}(\varphi)$ in the case of non-empty theory $\mathcal{T}$, which is compliant with an efficient usage of standard $\mathcal{LA}(\mathbb{Q})$-solvers/optimizers.

4.2.1 OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) wrt. other Optimization Problems

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

Linear Programming (LP) is a particular subcase of OMT($\mathcal{LA}(\mathbb{Q})$) with no Boolean component, since we can see (2.1) as $\varphi \stackrel{\text{def}}{=} \varphi' \wedge (\mathsf{cost} = \sum_i \mathbf{a}_i \mathbf{x}_i)$ where $\varphi' = \bigwedge_j (\sum_i \mathbf{A}_{ij} \mathbf{x}_i \leq \mathbf{b}_j)$.

Disjunctive programming (DP) is also easily encoded into OMT($\mathcal{LA}(\mathbb{Q})$), since (2.4) can be written as

$$\bigvee_i \bigwedge_j (\mathbf{A}^i_j \mathbf{x} \geq \mathbf{b}^i_j) \text{ or } \bigwedge_j (\mathbf{A}_j \mathbf{x} \geq \mathbf{b}_j) \wedge \bigwedge_{j=1}^t \bigvee_{k \in I_j} (\mathbf{c}^k \mathbf{x} \geq d^k) \quad (4.4)$$

respectively, where $\mathbf{A}^i_j$ and $\mathbf{A}_j$ are respectively the jth row of the matrices $\mathbf{A}^i$ and $\mathbf{A}$, $\mathbf{b}^i_j$ and $\mathbf{b}_j$ are respectively the jth row of the vectors $\mathbf{b}^i$ and $\mathbf{b}$. Since the left equation (4.4) is not in CNF, the CNF-ization process of [74] is then applied.

Linear Generalized Disjunctive programming (LGDP) (2.6) is straightforwardly encoded into a OMT($\mathcal{LA}(\mathbb{Q})$) problem $\langle \varphi, \mathsf{cost} \rangle$:

$$\begin{aligned} \varphi \stackrel{\text{def}}{=} & (\mathsf{cost} = \textstyle\sum_{\forall k \in K} \mathbf{z}_k + \mathbf{d}\mathbf{x}) \wedge [[\mathbf{B}\mathbf{x} \leq \mathbf{b}]] \wedge \phi \wedge [[\mathbf{0} \leq \mathbf{x}]] \wedge [[\mathbf{x} \leq \mathbf{e}]] \\ \wedge & \textstyle\bigwedge_{k \in K} \bigvee_{j \in J_k} (Y^{jk} \wedge [[\mathbf{A}^{jk} \mathbf{x} \geq \mathbf{a}^{jk}]] \wedge (\mathbf{z}_k = c^{jk})) \end{aligned} \quad (4.5)$$

s.t. $[[\mathbf{x} \bowtie \mathbf{a}]]$ and $[[\mathbf{A}\mathbf{x} \bowtie \mathbf{a}]]$ are abbreviations respectively for $\bigwedge_i(\mathbf{x}_i \bowtie \mathbf{a}_i)$ and $\bigwedge_i(\mathbf{A}_{i.}\mathbf{x} \bowtie \mathbf{a}_i)$, $\bowtie \in \{=, \neq \leq, \geq, <, >\}$. Since (4.5) is not in CNF, the CNF-ization process of [74] is then applied.

Pseudo-Boolean (PB) constraints (see [76]) in the form $(\sum_i \mathbf{a}_i X^i \leq b)$, s.t. X^i are Boolean atoms and $\mathbf{a}_i$ constant values in $\mathbb{Q}$, and cost functions cost $= \sum_i \mathbf{a}_i X^i$, are encoded into OMT($\mathcal{LA}(\mathbb{Q})$) 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: [2]

$$\bigwedge_i((\neg X^i \vee (\mathbf{x}_i = \mathbf{a}_i)) \wedge (X^i \vee (\mathbf{x}_i = 0)) \wedge (\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)\ ). \tag{4.6}$$

A **partial weighted MaxSMT** problem (see [70, 31, 32]) is a pair $\langle \varphi_h, \varphi_s \rangle$ where φ_h is a set of "hard" $\mathcal{T}$-clauses and φ_s is a set of weighted "soft" $\mathcal{T}$-clauses, s.t. a positive weight $\mathbf{a}_i$ is associated to each soft $\mathcal{T}$-clause $C_i \in \varphi_s$; the problem consists into finding a maximum-weight set of soft $\mathcal{T}$-clauses ψ_s s.t. $\psi_s \subseteq \varphi_s$ and $\varphi_h \cup \psi_s$ is $\mathcal{T}$-satisfiable. (One can see $\mathbf{a}_i$ as a penalty to pay if 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 variables X^j as follows:

$$\varphi' \stackrel{\text{def}}{=} \varphi_h \cup \bigcup_{\mathcal{C}_j \in \varphi_s} \{(X^j \vee \mathcal{C}_j)\}; \;\; \mathsf{cost} \stackrel{\text{def}}{=} \sum_{\mathcal{C}_j \in \varphi_s} \mathbf{a}_j X^j. \tag{4.7}$$

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

$$\varphi_h \stackrel{\text{def}}{=} \varphi'; \;\; \varphi_s \stackrel{\text{def}}{=} \bigcup_j \{\underbrace{(\neg X^j)}_{\mathbf{a}_j}\}. \tag{4.8}$$

Thus, combining (4.6) and (4.7), optimization problems for SAT with PB constraints and MaxSAT can be encoded into OMT($\mathcal{LA}(\mathbb{Q})$), whilst those for SMT($\mathcal{T}$) with PB constraints and MaxSMT can be encoded into OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) (assuming $\mathcal{T}$ matches the definition above).

[2]The term "$(\mathbf{x}_i \geq 0) \wedge (\mathbf{x}_i \leq \mathbf{a}_i)$" in (4.6) 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.

Remark 4.1. We notice the deep difference between OMT($\mathcal{LA}(\mathbb{Q})$) (or its extension to combination of theories 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 [70, 31]. With the latter problems, 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.

Chapter 5

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 §6.3.)

In this chapter we present more sophisticate procedures, based on the combination of SMT and minimization techniques. We first present and discuss an *offline* schema (§5.1) and an *inline* (§5.2) schema for an OMT($\mathcal{LA}(\mathbb{Q})$) procedure; the former is much simpler since it uses an incremental SMT solver as a black-box, whilst the latter 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.) Then we show how

to extend them to the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) case (§5.3).

5.1 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 5. 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). Notice that by providing a lower bound lb [resp. an upper bound ub] the user implicitly assumes the responsibility of asserting there is no model whose cost is lower than lb [there is a model whose cost is ub]. 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). [1] 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, driven by the heuristic $\mathsf{BinSearchMode}()$. Notice that if $\mathsf{u} = +\infty$ or $\mathsf{l} = -\infty$, then $\mathsf{BinSearchMode}()$ 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, see §2.1; it is also very important to recall that during the SMT call φ is updated with the clauses which are learned during the

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

Algorithm 5 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} = $ 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** {res = 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** $\langle \mathcal{M}, \mathsf{u} \rangle$

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, [2] returning the model $\mathcal{M}$ for μ of minimum cost u ($-\infty$ iff the problem

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

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 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 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 SMT.ExtractUnsatCore, which returns also the subset η of formulas in (the formula stack of) φ which caused the unsatisfiability of φ (see §2.1). This exploits techniques similar to unsat-core extraction [62]. (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 BinSearchMode() always returned true, then Algorithm 5 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. [3]

[3] 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 which occur as leaves of such proof (see e.g. [35] for details). If the SMT solver is invoked on φ even $\varphi \setminus \mathsf{PIV}$ is $\mathcal{LA}(\mathbb{Q})$-inconsistent, then it can "use" PIV and return a proof involving it even though another PIV-less proof exists.

Thus, e.g., the procedure might got stuck into a "Zeno" [4] 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.)

Under such hypothesis, as a consequence of Theorem 4.1 of §4.2, we have that:

(i) Algorithm 5 terminates. In linear-search mode it terminates 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. In mixed linear/binary-search mode, as above, it terminates since there can be at most finitely-many binary-search loops between two consequent linear-search loops;

(ii) Algorithm 5 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 5 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).

5.1.1 Handling strict inequalities

Minimize is a simple extension of the simplex-based $\mathcal{LA}(\mathbb{Q})$-Solver of [41], which is invoked after one solution is found, minimizing it by standard simplex techniques. We recall that the algorithm in [41] can handle strict inequalities.

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

Thus, if μ contains strict inequalities, then Minimize temporarily relaxes them into non-strict ones and finds the minimum-cost solution of the relaxed problem. Then:

1. if such minimum-cost solution $\mathbf{x}$ of cost min lays only on non-strict inequalities, then $\mathbf{x}$ is a solution of the original problem, hence min can be returned;

2. otherwise, we may have two alternative subcases:

   (i) there is some other solution of cost min. [5] If so, the value min can be returned;

   (ii) there is no solution of cost min. If so, then for some $\delta > 0$ and for every cost $c \in]\mathsf{min}, \mathsf{min} + \delta]$ there exists a solution of cost c. (If needed explicitly, such solution can be computed using the techniques for handling strict inequalities briefly described in §2.2.2.) Thus the value min can be tagged as a non-strict minimum and returned, so that the constraint $(\mathsf{cost} \leq \mathsf{min})$, rather than $(\mathsf{cost} < \mathsf{min})$, is added to φ.

Condition 2.(i) can be checked easily, e.g., by temporarily adding the constraint $(\mathsf{cost} \leq \mathsf{min})$ to μ and then by invoking again the $\mathcal{LA}(\mathbb{Q})$-solving procedure of [41] on μ (without minimization), since such algorithm can handle strict inequalities. (In our implementation, we have directly modified the algorithm of [41] so that to perform this check internally.)

5.1.2 Discussion.

We remark a few facts about this procedure.

[5] This subcase is rare in practice but it is possible in principle. For instance, suppose we have that $\mu = \{(\mathsf{cost} \geq 1), (\mathsf{cost} > y), (\mathsf{cost} > -y)\}$. If we temporarily relax strict inequalities into non-strict ones, then $\{\mathsf{cost} = 1, y = 1\}$ is a minimum-cost solution which lays on the strict inequality $(\mathsf{cost} > y)$. Nevertheless, there is a solution of cost 1, e.g., $\{\mathsf{cost} = 1, y = 0.9999\}$.

1. If Algorithm 5 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 §2.1 and §2.2) plays an essential role here, since at every call $\mathsf{SMT.IncrementalSolve}$ 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 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 5 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 $\mathsf{SMT.IncrementalSolve}$ 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 §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, $\mathsf{SMT.IncrementalSolve}$ implicitly plays a form of *branch & bound*: (i) decide a new literal l and unit- or theory-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").

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

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 15 can be smaller than pivot, so that the cost range is more

than halved.

6. Unlike with other domains (e.g., 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 to use dynamic/adaptive strategies for ComputePivot (see [82]).

5.2 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 §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 5, 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 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 4-7 in Algorithm 5, temporarily restricting the cost range to $[\mathsf{l}, \mathsf{pivot}[$.

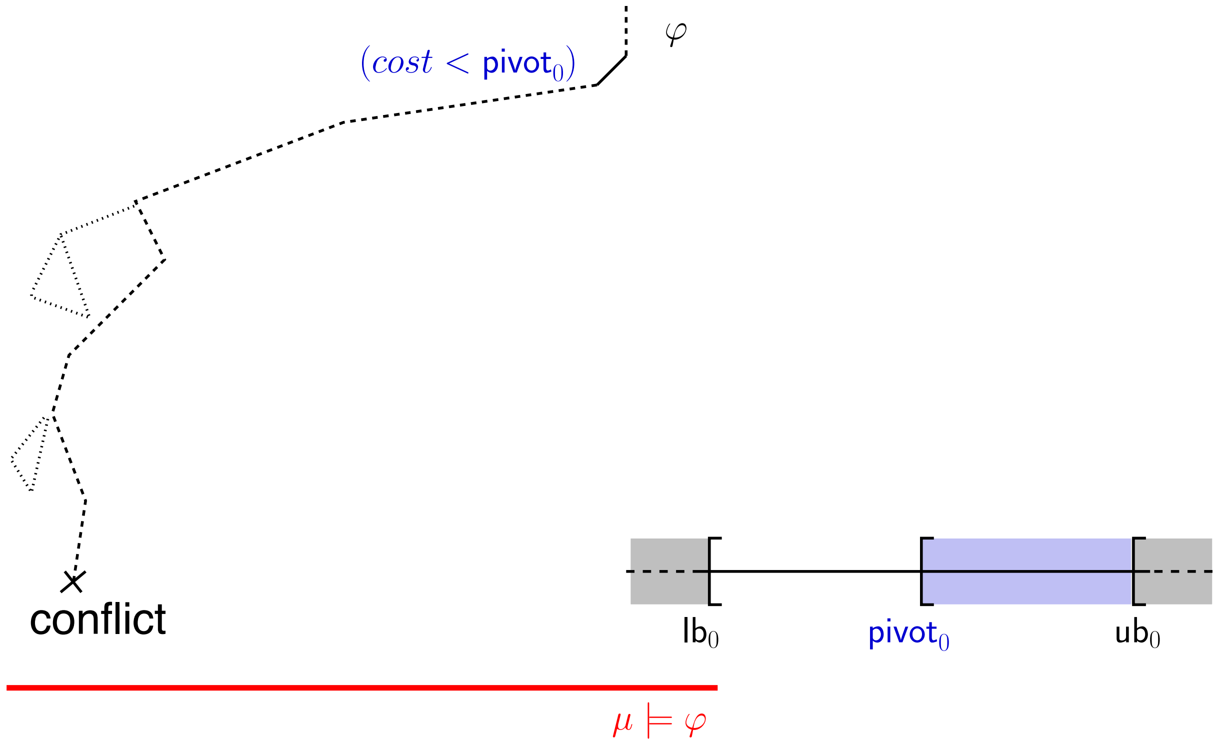

Figure 5.1: One piece of possible execution of an inline procedure: pivoting on $(\mathsf{cost} < \mathsf{pivot}_0)$.

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 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 5, permanently restricting the cost range to $[\mathsf{l}, \mathsf{min}[$. 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 (see Figures 5.1-5.3).

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., $\varphi \wedge$

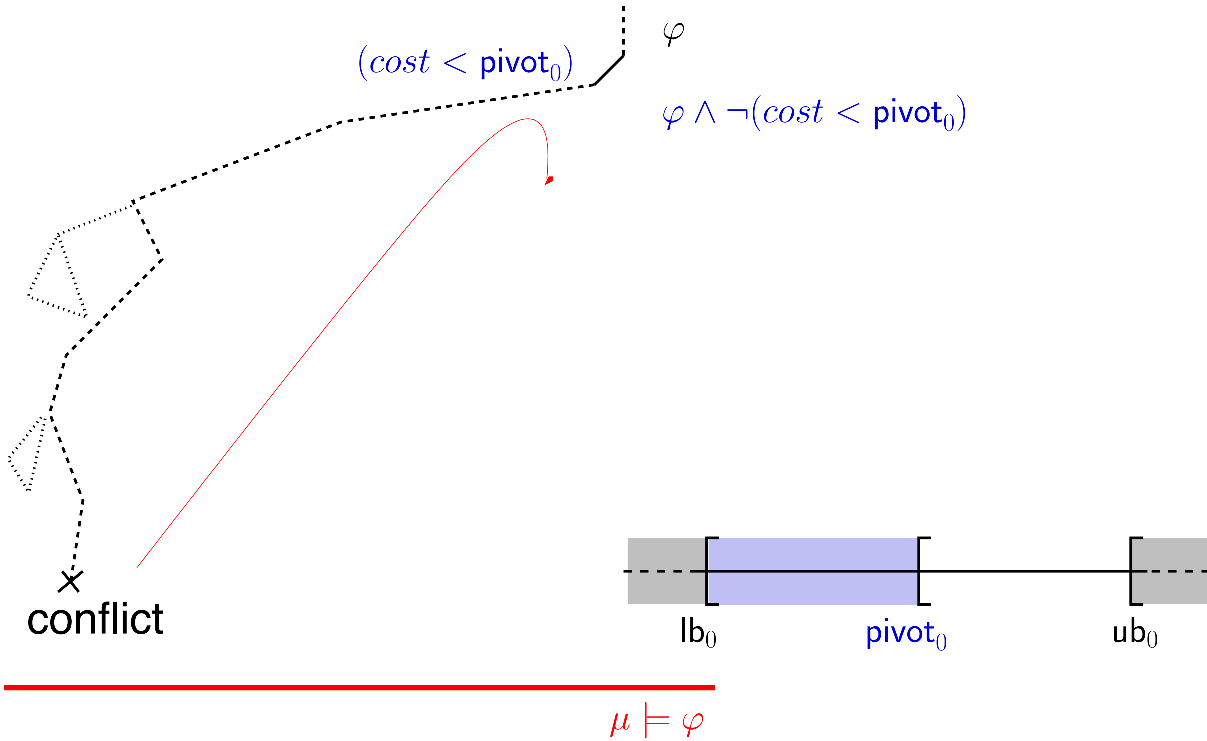

Figure 5.2: One piece of possible execution of an inline procedure: increasing the lower bound to pivot_0.

PIV is $\mathcal{LA}(\mathbb{Q})$-inconsistent), then the SAT solver backtracks to level 0, unit-propagating $\neg$PIV. This case mirrors steps 21-23 in Algorithm 5, permanently restricting the cost range to $[\mathsf{pivot}, \mathsf{u}[$.

Although the modified SMT solver mimics to some extent the behaviour of Algorithm 5, 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 facts 1.-6. about the offline procedure in §5.1 hold for the inline version. The efficiency of the inline procedure can be further improved as fol-

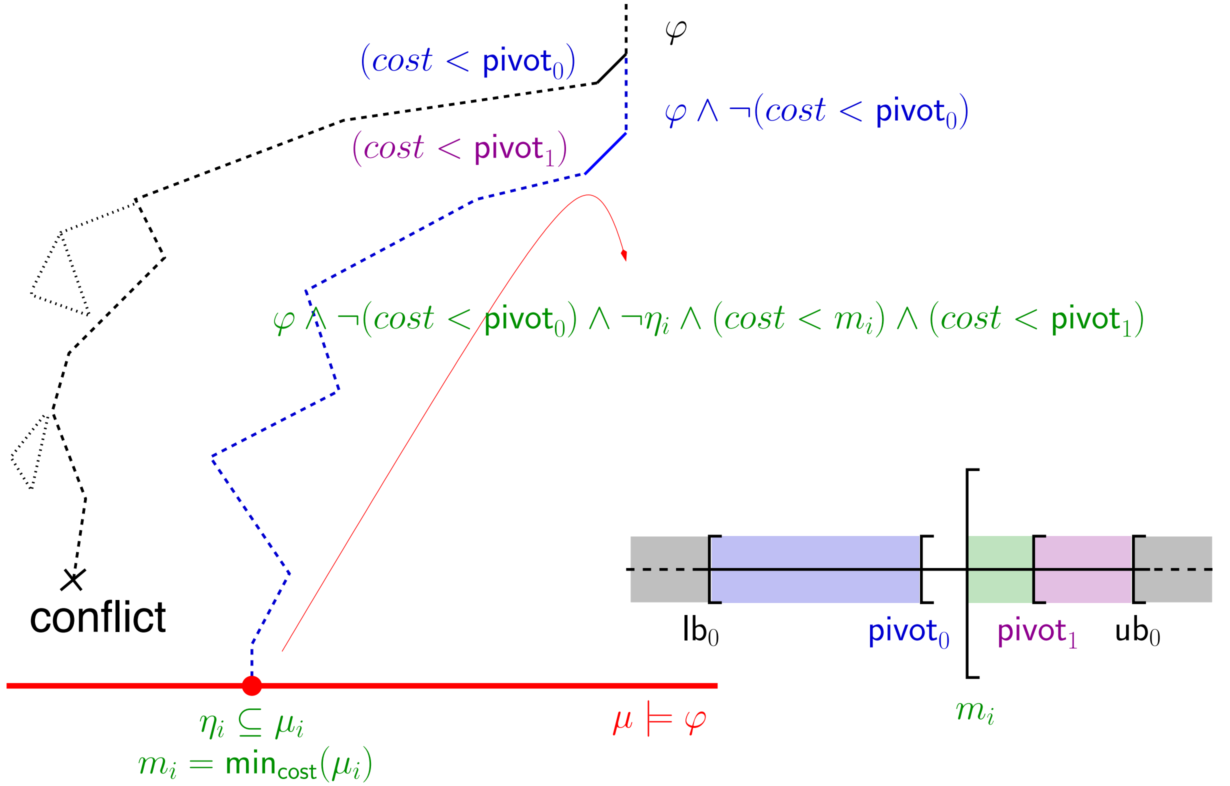

Figure 5.3: One piece of possible execution of an inline procedure: decreasing the upper bound to $\mathsf{min}_{\mathsf{cost}}(\mu_i)$.

lows.

Activating previously-learned clauses. In binary-search mode, when a truth assignment μ with a novel minimum min is found, not only $(\mathsf{cost} < \mathsf{min})$ but also $\mathsf{PIV} \stackrel{\text{def}}{=} (\mathsf{cost} < \mathsf{pivot})$ 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 5 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.

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 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 [41].) 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 $(\mathsf{cost} < \mathsf{u})$ 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. If $\mathsf{u} > \mathsf{max} > \mathsf{pivot}$, then the two 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 $C \stackrel{\text{def}}{=} \neg\eta \vee \neg\mathsf{PIV}$. In particular, C_2 forces backjumping to level 1 ad propagating the (possibly fresh) atom $(\mathsf{cost} < \mathsf{max})$; eventually, instead of using C do drive backjumping to level 0 and propagating $\neg\mathsf{PIV}$, the SMT solver may use C_1 for backjumping to level 0 and propagating $\neg(\mathsf{cost} < \mathsf{max})$, restricting the range to $[\mathsf{max}, \mathsf{u}[$ rather than to $[\mathsf{pivot}, \mathsf{u}[$.

Example 5.1. 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 $\mathsf{BinSearchMode}()$ 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, then the heuristic selects

linear search mode. Otherwise, it selects binary-search mode if and only if

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

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 binary-search steps performed, estimating the progress achieved by such steps, whilst $\Delta\#\mathsf{conf}_{lin}$ and $\Delta\#\mathsf{conf}_{bin}$ are respectively the number of conflicts produced in such steps, estimating their expense. (Notice that if a binary-search step returns UNSAT, then $\Delta\mathsf{ub}_{bin} = 0$ and linear-search is chosen, in compliance with the strategy to avoid infinite "Zeno" sequences described in §5.1.)

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

We recall the terminology, assumptions, definitions and results of §4.2. Theorems 4.1, 4.3 and 4.4 allow for extending to the OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) case the procedures of §5.1 and §5.2 as follows.

As suggested by Theorem 4.3, straightforward OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) extensions of the procedures for OMT($\mathcal{LA}(\mathbb{Q})$) of §5.1, §5.2 would be such that the SMT solver enumerates ed-extended satisfying truth assignments $\eta \stackrel{\text{def}}{=} \mu \wedge \mu_{ed}$ as in Definition 4.4, 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, whilst correctness and completeness is guaranteed by Theorems 4.1 and 4.3.)

Nevertheless, as suggested in §4.2, 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 $\rho \stackrel{\text{def}}{=} \mu \wedge \mu_{eid}$ as in Definition 4.5, 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. (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 4.1 and 4.4.) This prevents from passing negated interface equalities to Minimize.

This motivates and explains the following OMT($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) variants of the offline and inline procedures of §5.1 and §5.2 respectively.

Algorithm 5 is modified as follows. First, SMT.IncrementalSolve in steps 8 and 11 is asked to return also a $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-model $\mathcal{I}$. Then in step 15 Minimize is invoked instead on $\langle \mathsf{cost}, \mu_{\mathcal{LA}(\mathbb{Q})} \cup \mu_{ei} \rangle$, s.t.

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

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($\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$) procedures comes nearly for free once the SMT solver handles $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{T}$-solving by *Delayed Theory Combination* [28], with the strategy of case-splitting automatically 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 4.5, 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 [28], implicitly implements points (a) and (c) of Theorem 4.4.) If so, then, in accordance with points (b) and (d) of Theorem 4.4, it suffices to apply Minimize to $\mu_{\mathcal{LA}(\mathbb{Q})} \wedge \mu_{ei}$, then learn $(\mathsf{cost} < \mathsf{min})$ and use it for backjumping, as in §5.2. As with the inline version, in practice the negated strict inequalities are omitted from μ_{ei}, because they are entailed by the corresponding non-negated equalities/inequalities.

Chapter 6

Experimental Evaluation for OMT($\mathcal{LA}(\mathbb{Q})$)

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 §5 on top of MATHSAT5 [1] [33] (we refer to them as OPTIMATHSAT). 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. binary vs. adaptive, denoted respectively by "-LIN", "-BIN" and "-ADA"). [2] 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 [3] v23.7.1 [29] 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 [54]. GAMS

[1] `http://mathsat.fbk.eu/`.

[2] Here "-LIN" means that BinSearchMode() always returns false, "-BIN" denotes the mixed linear-binary strategy described in §5.1 to ensure termination, whilst "-ADA" refers to the adaptive strategy illustrated in §5.2.

[3] `http://www.gams.com`.

provides two reformulation tools, LOGMIP [4] v2.0 and JAMS [5] (a new version of the EMP [6] solver), s.t. both of them allow for reformulating LGDP models by using either big-M (BM) or convex-hull (CH) methods [75, 78]. We use CPLEX v12.2 [54] (through an OSI/CPLEX link) to solve the reformulated MILP models. All the tools were executed using default options, as suggested by the authors [92]. We also compared OPTIMATHSAT against MATHSAT augmented by Pseudo-Boolean (PB) optimization [31] (we call it PB-MATHSAT) on MaxSMT problems.

Remark 6.1. Importantly, MATHSAT and OPTIMATHSAT use *infinite-precision arithmetic* whilst the GAMS tools and CPLEX implement standard *floating-point arithmetic*. Moreover the former handle strict inequalities natively (see §2.2), whilst the GAMS tools use an approximation with a very-small constant value "eps" ϵ (default $\epsilon \stackrel{\text{def}}{=} 10^{-6}$), so that, e.g., "$(x > 0)$ *is internally rewritten into* $(x \geq 10^{-6})$" [7].

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

- (§6.2) LGDP problems, proposed by LOGMIP and JAMS authors [93, 77, 78];
- (§6.3) OMT($\mathcal{LA}(\mathbb{Q})$) problems from SMT-LIB [8];
- (§6.4) OMT($\mathcal{LA}(\mathbb{Q})$) problems, coming from encoding parametric verification problems from the SAL [9] model checker;
- (§6.5) the MaxSMT problems from [31].

The encodings from LGDP to OMT($\mathcal{LA}(\mathbb{Q})$) and back are described in §6.1.

[4] http://www.logmip.ceride.gov.ar/index.html.
[5] http://www.gams.com/.
[6] http://www.gams.com/dd/docs/solvers/emp.pdf.
[7] GAMS support team, email personal communication, 2012.
[8] http://www.smtlib.org/.
[9] http://sal.csl.sri.com.

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 OPTIMATHSAT have been cross-checked by another SMT solver, YICES [10] 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 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 versions of OPTIMATHSAT passed the above checks. On the LGDP problems (§6.2) all tools agreed on the final results, apart from tiny rounding errors by GAMS tools; [11] on all the other problem collections (§6.3, §6.4, §6.5) 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, a Linux binary of OPTIMATHSAT, the problems, and the results are made available. [12] (We cannot distribute the GAMS tools since they are subject to licencing restrictions, see [29]; however, they can be obtained at GAMS url.)

6.1 Encodings.

In order to translate LGDP models into OMT($\mathcal{LA}(\mathbb{Q})$) problems we use the encoding in (4.5) of §4.2.1, namely LGDP2SMT. [13]

In order to translate OMT($\mathcal{LA}(\mathbb{Q})$) problems into LGDP models we consider

[10] `http://yices.csl.sri.com/`.

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

[12] `http://disi.unitn.it/~rseba/optimathsat2014.tgz`

[13] 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 Converted tool for converting complex GAMS specifications (e.g. containing sets and indexed equations) into simpler specifications.

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 non-strict inequalities $\neg(t_1 \geq t_2)$. [14] 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 [74], we compute the Boolean abstraction φ'^p of φ' (which plays the role of formula ϕ in (2.6)) and then, for each $\mathcal{LA}$-atom ψ_i occurring positively [resp. negatively] in φ', we add the disjunction $\neg A_i \vee \psi_i$ [resp. $A_i \vee \neg\psi_i$], 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 non-unit clause $(l_{i1} \vee \ldots \vee l_{in}) \in \varphi'$ as a LGDP disjunction $[Y_i^1 \wedge l_{i1}] \vee \ldots \vee [Y_i^n \wedge l_{in}]$, where $Y_i^1, \ldots, Y_i^n$ are fresh Boolean variables.

Remark 6.2. We decided to provide two different encodings for a bunch of reasons. SMT2LGDP$_1$ is a straightforward and very-natural encoding. However, we have verified empirically, and some discussion with GAMS support team confirmed it [15], that some GAMS tools/options have often problems in handling efficiently and even correctly the Boolean structure of the formulas ϕ in (2.6) (see e.g. the number of problems terminated with error messages in §6.3-§6.5). Thus, following also the suggestions of the GAMS support team, we have introduced SMT2LGDP$_2$, which eliminates any Boolean structure, reducing the encoding substantially to a set of LGDP disjunctions. Notice, however, that SMT2LGDP$_2$ benefits from the CNF encoder of MATHSAT5.

[14]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.

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

6.2 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 [93, 77, 78]. We adopted the encoding of the problems into LGDP given by the authors [16] 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 §6.1, and to check the robustness of the tools wrt. 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.

6.2.1 The strip-packing problem.

Given a set of N rectangles of different length L_i and height H_i, $i \in 1, .., N$, 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. (See Figure 6.1.)

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

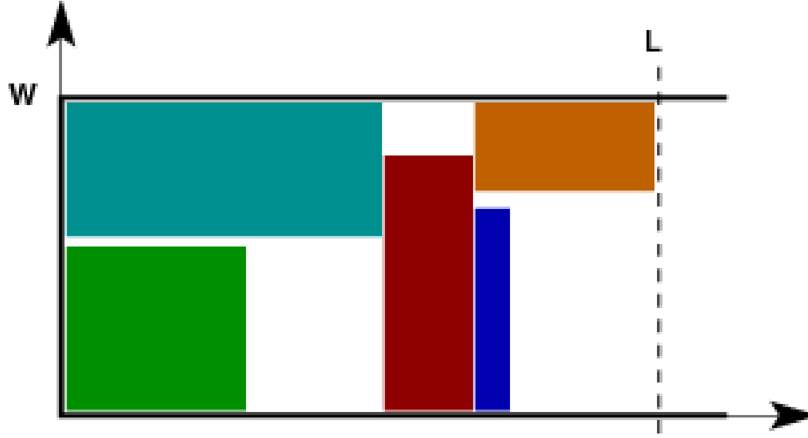

Figure 6.1: Graphical representation of a strip-packing problem.

The LGDP model provided by [77] is the following:

$$
\begin{aligned}
\min \quad & L & \\
\text{s.t.} \quad & L \geq x_i + L_i & \forall i \in N \\
& \begin{bmatrix} Y_{ij}^1 \\ x_i + L_i \leq x_j \end{bmatrix} \vee \begin{bmatrix} Y_{ij}^2 \\ x_j + L_j \leq x_i \end{bmatrix} & \\
& \vee \begin{bmatrix} Y_{ij}^3 \\ y_i - H_i \geq y_j \end{bmatrix} \vee \begin{bmatrix} Y_{ij}^4 \\ y_j - H_j \geq y_i \end{bmatrix} & \forall i, j \in N, i < j \\
& x_i \leq \mathsf{ub} - L_i & \forall i \in N \\
& H_i \leq y_i \leq W & \forall i \in N \\
& L, x_i, y_i \in \mathbb{R}_+^1, Y_{ij}^1, Y_{ij}^2, Y_{ij}^3, Y_{ij}^4 \in \{True, False\} &
\end{aligned}
\tag{6.1}
$$

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 2-dimensional space). 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 (6.1) into OMT($\mathcal{LA}(\mathbb{Q})$) as follows:

$$\begin{aligned}\varphi &\stackrel{\text{def}}{=} (\mathsf{cost} = L) \wedge \textstyle\bigwedge_{\forall i \in N}(L \geq x_i + L_i) \\ &\wedge \textstyle\bigwedge_{\forall i,j \in N, i<j}\Big((x_i + L_i \leq x_j) \vee (x_j + L_j \leq x_i) \\ &\quad \vee (y_i - H_i \geq y_j) \vee (y_j - H_j \geq y_i)\Big) \\ &\wedge \textstyle\bigwedge_{\forall i \in N}(x_i \leq \mathsf{ub} - L_i) \wedge \bigwedge_{\forall i \in N}(x_i \geq 0) \\ &\wedge \textstyle\bigwedge_{\forall i \in N}(H_i \leq y_i) \wedge \bigwedge_{\forall i \in N}(W \geq y_i) \wedge \bigwedge_{\forall i \in N}(y_i \geq 0)\end{aligned} \tag{6.2}$$

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 [77], 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 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, whilst $W = 1$ is half the average size of one rectangle.)

6.2.2 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 scheduling* problem is to minimize the makespan, that is the total length of the schedule. (See Figure 6.2.)

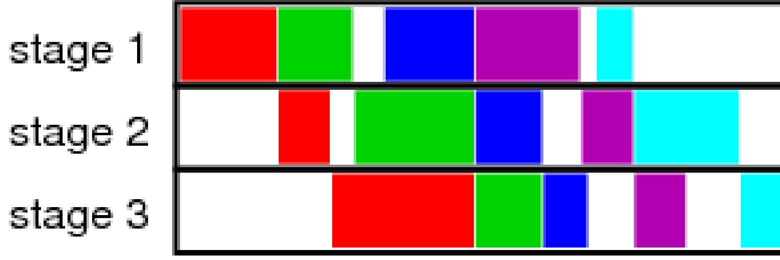

Figure 6.2: Graphical representation of a zero-wait jobshop problem.

The LGDP model provided by [77] is:

$$
\begin{array}{lcr}
\min & M & \\
\text{s.t.} & M \geq \sum_{\forall j \in J_i} t_{ij} & \forall i \in I \\
& \left[\begin{array}{c} Y^1_{ik} \\ t_i + \sum_{\forall m \in J_i, m \leq j} t_{im} \leq t_k + \sum_{\forall m \in J_k, m < j} t_{km} \end{array} \right] \vee & \\
& \left[\begin{array}{c} Y^2_{ik} \\ t_k + \sum_{\forall m \in J_k, m \leq j} t_{km} \leq t_i + \sum_{\forall m \in J_i, m < j} t_{im} \end{array} \right] & \begin{array}{c} \forall j \in C_{ik}, \\ \forall i, k \in I, i < k \end{array} \\
& M, t_i \in \mathbb{R}^1_+, Y^1_{ik}, Y^2_{ik} \in \{True, False\} & \forall i, k \in I, i < k
\end{array}
\tag{6.3}
$$

where M corresponds to the objective function to minimize and every jobs $i \in I$ 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 occur at any stage at the same time. We encoded the corresponding LGDP model (6.3) into OMT($\mathcal{LA}(\mathbb{Q})$) as follows:

$$
\begin{array}{rcl}
\varphi & \stackrel{\text{def}}{=} & (\mathsf{cost} = M) \\
& \wedge & \bigwedge_{i \in I} (M \geq s_i + \sum_{\forall j \in J_i} t_{ij}) \wedge \bigwedge_{i \in I} (s_i \geq 0) \\
& \wedge & \bigwedge_{\forall j \in C_{ik}, \forall i, k \in I, i < k} \Big((s_i + \sum_{\forall m \in J_i, m \leq j} t_{im} \leq s_k + \sum_{\forall m \in J_k, m < j} t_{km}) \\
& & \qquad \vee \, (s_k + \sum_{\forall m \in J_k, m \leq j} t_{km} \leq s_i + \sum_{\forall m \in J_i, m < j} t_{im}) \Big)
\end{array}
\tag{6.4}
$$

We generated randomly instances of the zero-wait jobshop 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 $\mathsf{lb} = 0$.

6.2.3 Discussion

The table of Figure 6.3 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 Figures 6.4-6.6 compare the best-performing version of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against LOGMIP+CPLEX with BM and CH reformulation (left and center respectively) and the two inline versions OPTIMATHSAT-LIN-IN and OPTIMATHSAT-BIN-IN on "directly generated" benchmarks.

The table of Figure 6.7 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 Figures 6.8-6.10 compare the best-performing version of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against LOGMIP with BM and CH reformulation (left and center respectively) on "directly encoded" benchmarks; the figure also compares the two inline versions OPTIMATHSAT-LIN-IN and OPTIMATHSAT-BIN-IN.

The results on the LGDP problems 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 seems to perform a little better than the corresponding -BIN and -ADA options (although gaps are not dramatic).

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

Figure 6.3: 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) each of the strip-packing problem for N rectangles, where $N = 9, 12, 15$, and width $W = \sqrt{N}/2$, 1. Values highlighted in **bold** represent best performances.

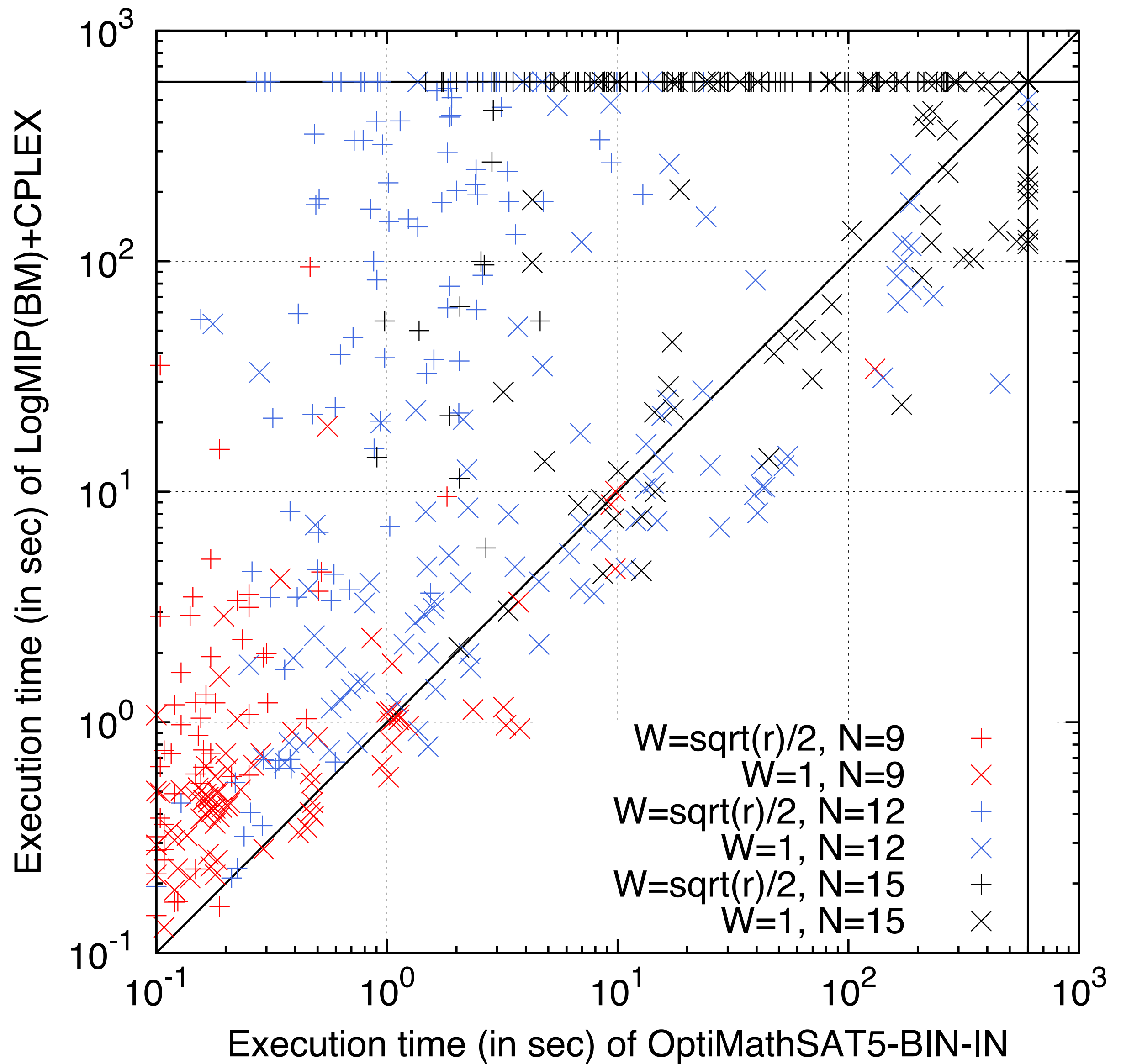

Figure 6.4: Comparison of the binary-search configuration of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX on "directly generated" instances of the strip-packing problem.

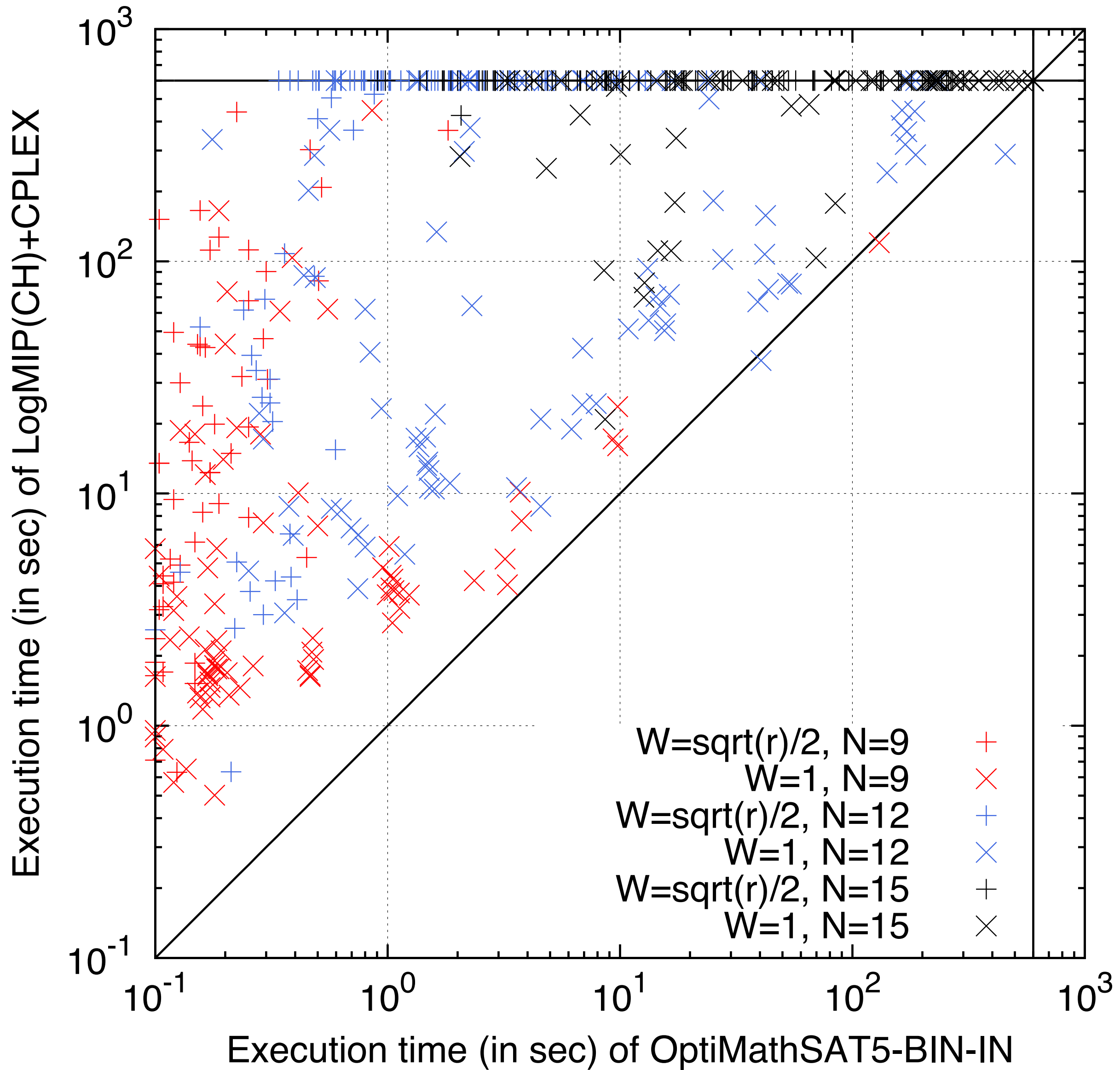

Figure 6.5: Comparison of the binary-search configuration of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) and LOGMIP(CH)+CPLEX on "directly generated" instances of the strip-packing problem.

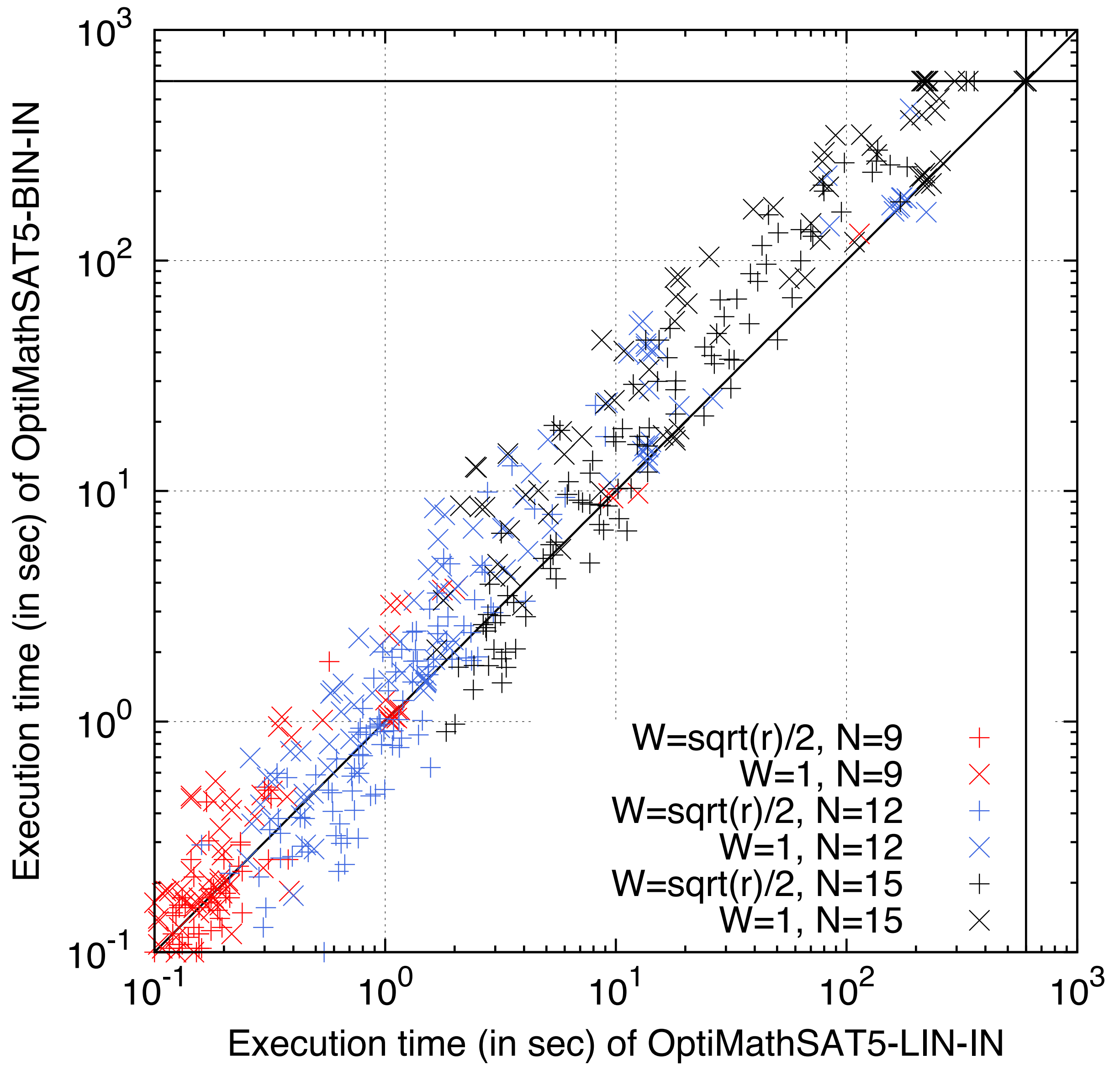

Figure 6.6: Comparison of the inline versions of OPTIMATHSAT on "directly generated" benchmarks of the strip-packing problem: OPTIMATHSAT-LIN-IN against OPTIMATHSAT-BIN-IN.

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														
OPTIMATHSAT5-LIN-OF	100	386	100	1854	97	9396	57	14051	100	9637	99	10670	553	45995
OPTIMATHSAT5-LIN-IN	100	317	100	1584	100	8100	77	18046	100	7738	100	7433	**577**	**43228**
OPTIMATHSAT5-BIN-OF	100	726	100	3817	88	13222	38	12529	92	14183	90	13287	508	57764
OPTIMATHSAT5-BIN-IN	100	602	100	3270	97	12878	54	16234	96	13159	96	12350	543	58493
OPTIMATHSAT5-ADA-IN	100	596	100	3230	97	12262	53	14810	96	12805	96	12125	542	55828
JAMS(BM)+CPLEX	100	268	100	1113	100	4734	87	17067	100	4941	100	6122	**587**	**34245**
JAMS(CH)+CPLEX	84	23830	4	1596	0	0	0	0	0	0	1	363	89	25789
LOGMIP(BM)+CPLEX	100	267	100	1114	100	4718	87	17108	100	4962	100	6174	**587**	**34343**
LOGMIP(CH)+CPLEX	84	23871	4	1622	0	0	0	0	0	0	1	338	89	25831
LGDP2SMT Encoded Benchmarks														
OPTIMATHSAT5-LIN-IN	100	324	100	1571	100	7739	74	16494	100	7175	100	7504	574	40807
SMT2LGDP$_1$-LGDP2SMT Encoded Benchmarks														
OPTIMATHSAT5-LIN-IN	100	336	100	1578	100	7762	71	16589	100	7726	100	7706	571	41697
SMT2LGDP$_2$-LGDP2SMT Encoded Benchmarks														
OPTIMATHSAT5-LIN-IN	100	320	100	1533	100	7623	68	15120	100	7216	100	7598	568	39410
SMT2LGDP$_1$ Encoded Benchmarks														
JAMS(BM)+CPLEX	100	239	100	1128	100	5516	84	19949	100	6667	100	4176	**584**	**37675**
JAMS(CH)+CPLEX	100	14527	46	17887	0	0	0	0	1	497	0	0	147	32911
LOGMIP(BM)+CPLEX	100	240	100	1122	100	5510	83	19489	100	6684	100	4180	583	37225
LOGMIP(CH)+CPLEX	100	14465	47	18206	0	0	0	0	1	495	0	0	148	33166
SMT2LGDP$_2$ Encoded Benchmarks														
JAMS(BM)+CPLEX	100	319	100	1865	100	12470	45	15704	97	13189	96	15773	**538**	**59320**
JAMS(CH)+CPLEX	95	22435	18	8030	2	671	0	0	1	526	3	1043	119	32723
LOGMIP(BM)+CPLEX	100	319	100	1871	100	12440	45	15747	98	13661	95	15102	**538**	**59140**
LOGMIP(CH)+CPLEX	95	22401	18	7991	1	163	0	0	1	437	3	1020	118	32012

Figure 6.7: 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) each of the job-shop problem for $I = 9, 10, 11, 12$ jobs and $J = 8$ stages and for $I = 11$ jobs and $J = 9, 10$ stage.

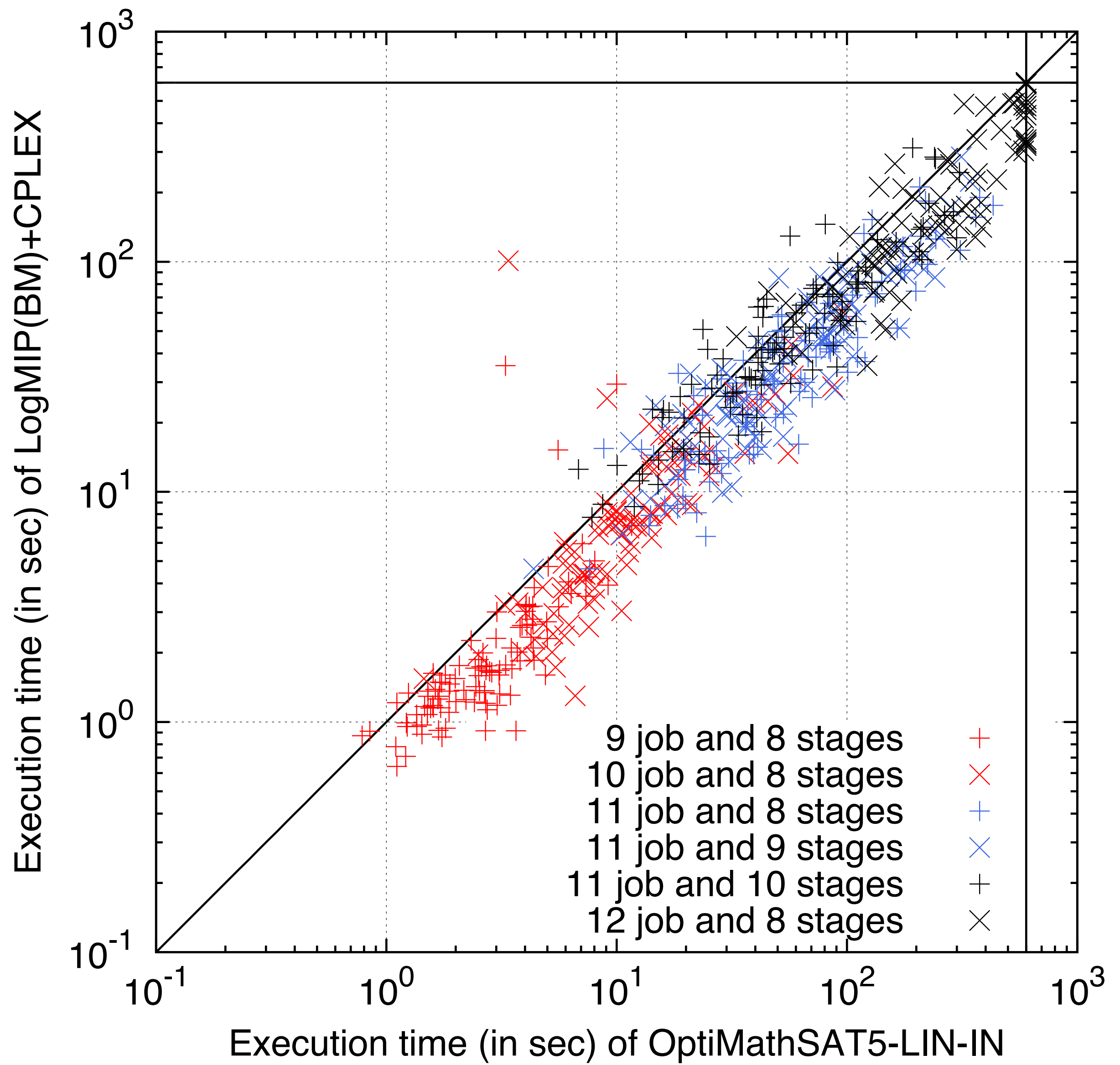

Figure 6.8: Comparison of the best version of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX on "directly generated" benchmarks of job shop problem.

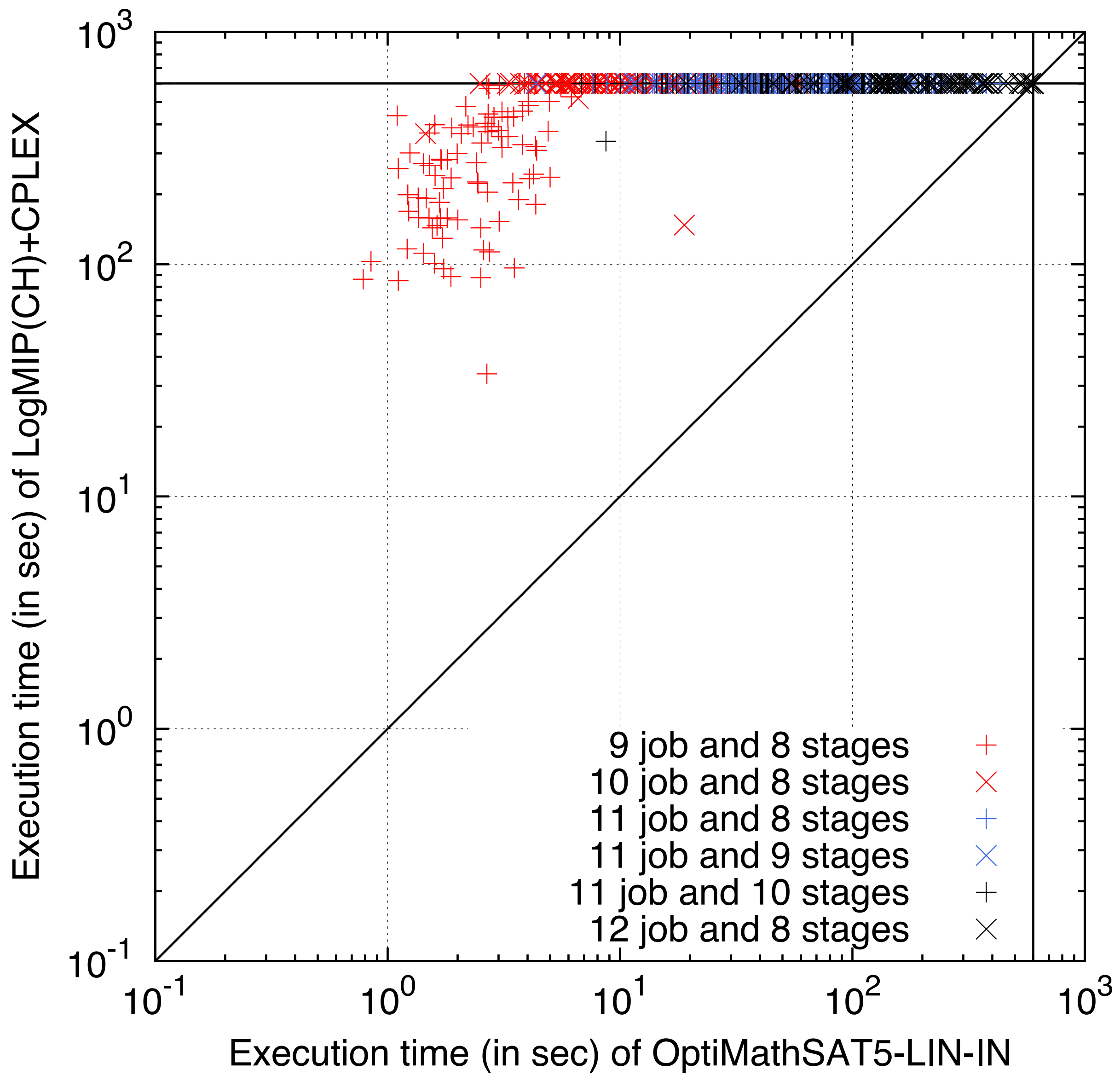

Figure 6.9: Comparison of the best version of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(CH)+CPLEX on "directly generated" benchmarks of job shop problem.

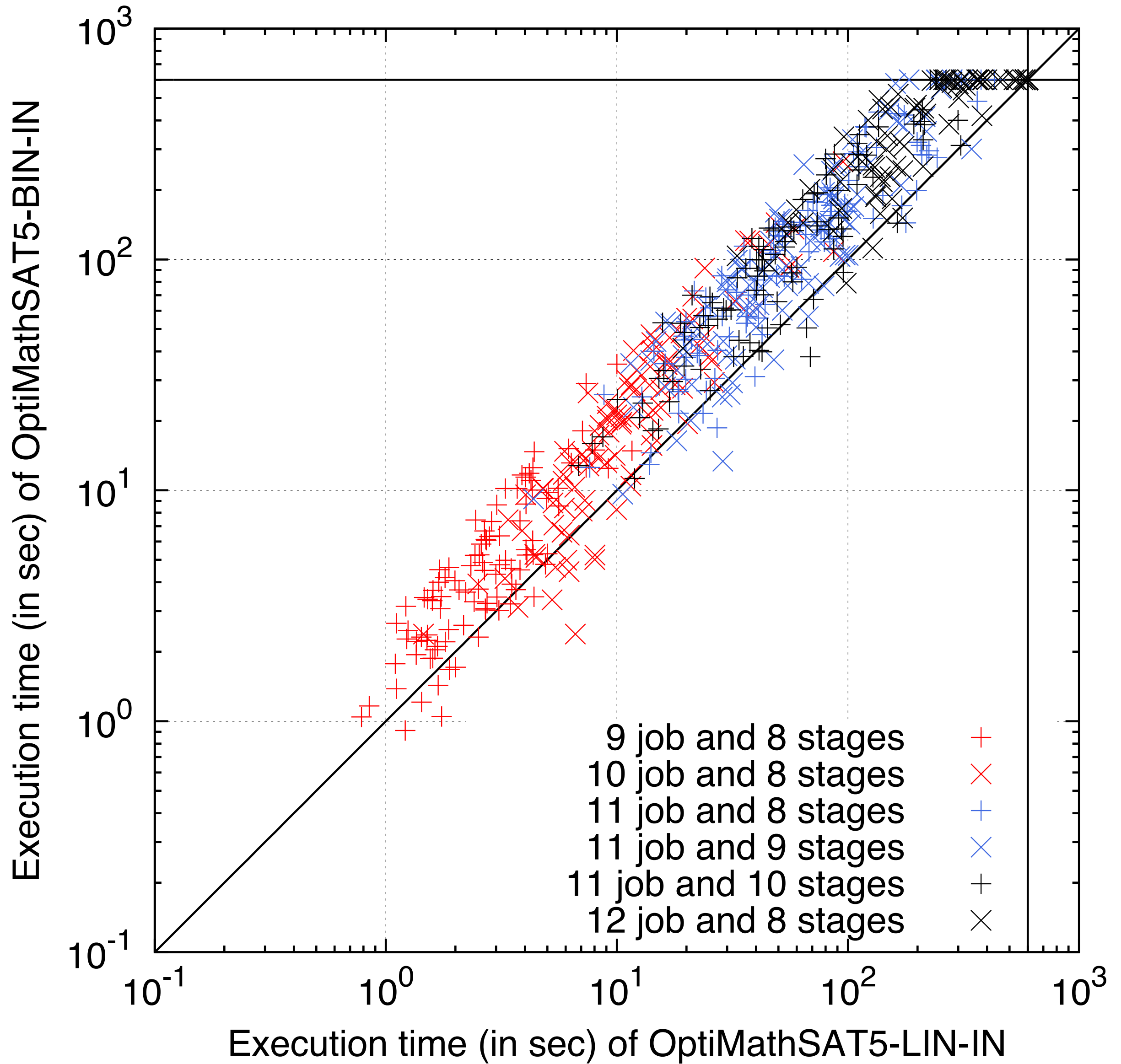

Figure 6.10: Comparison of the two inline versions of OPTIMATHSAT on "directly generated" benchmarks of job shop problem: OPTIMATHSAT-LIN-IN against OPTIMATHSAT-BIN-IN.

Remark 6.3. We notice that with LGDP problems binary search is not "obviously faster" than linear search, in compliance with what stated in point 6. in §5.1. This is further enforced by the fact that in strip-packing (6.2) [resp. job-shop (6.4)] encodings, the cost variables cost $\stackrel{\text{def}}{=} L$ [resp. cost $\stackrel{\text{def}}{=} M$] occurs only in positive unit clauses in the form $(L \geq \langle term \rangle)$ [resp. $(M \geq \langle term \rangle)$];

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. This latter fact is relevant, since the authors of [93, 77, 78] overall do not champion one method over the other.

Comparing the performances of the versions of OPTIMATHSAT against these of the GAMS tools, we notice that

- on *strip-packing problems* all versions of OPTIMATHSAT outperform all GAMS versions, regardless the encoding used. E.g., 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, whilst it is slightly beaten by the BM versions on "directly generated" and "SMT2LGDP$_1$ encoded" benchmarks. E.g., 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 §6 that all solvers agreed on the results, regardless 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 is 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 wrt. the encodings of these problems.
- On GAMS tools the effects of the different encodings are more relevant, although very heterogeneous: e.g., wrt. 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, slightly less on strip-packing and a much more on job-shop with CH options. For this reason, in next sections we always report the results with both encodings.

Analysis of OPTIMATHSAT performances.

We want to perform a more fine-grained analysis of the performances of the best version of OPTIMATHSAT, OPTIMATHSAT-LIN-IN. To this extent, we partition the total execution time taken on each problem into three consecutive components:

- *solving time*, i.e. the time spent on finding the first sub-optimal solution,
- *minimization time*, i.e. the time required to search for the optimal solution,
- and *certification time*, i.e. the time needed for checking there is no better solution.

Figures 6.11-6.16 reports, for all strip-packing and job-shop instances, the ratios of the three components above 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:

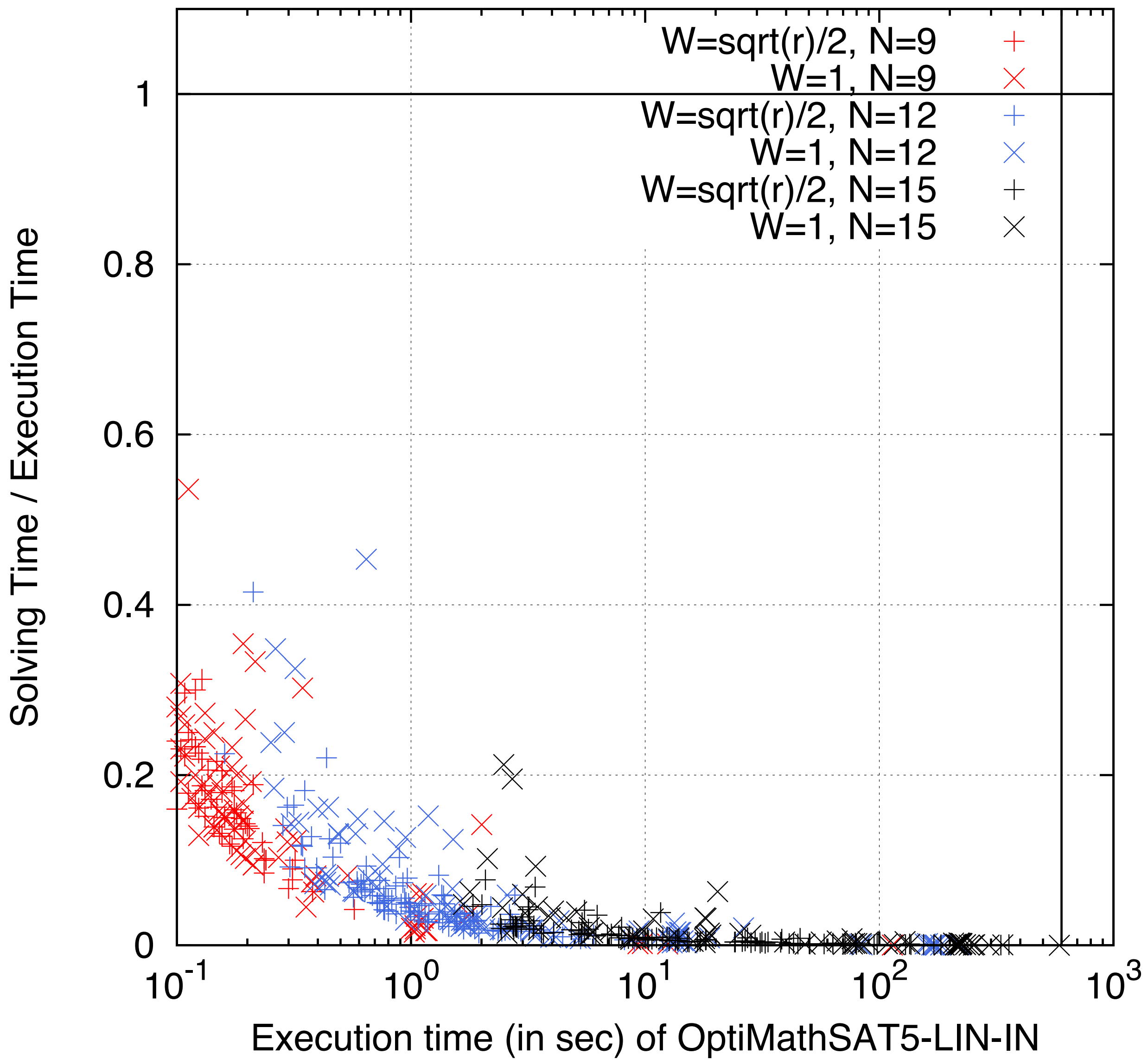

Figure 6.11: Comparing solving time with the execution time of OPTIMATHSAT-LIN-IN on "directly generated" instances of strip-packing.

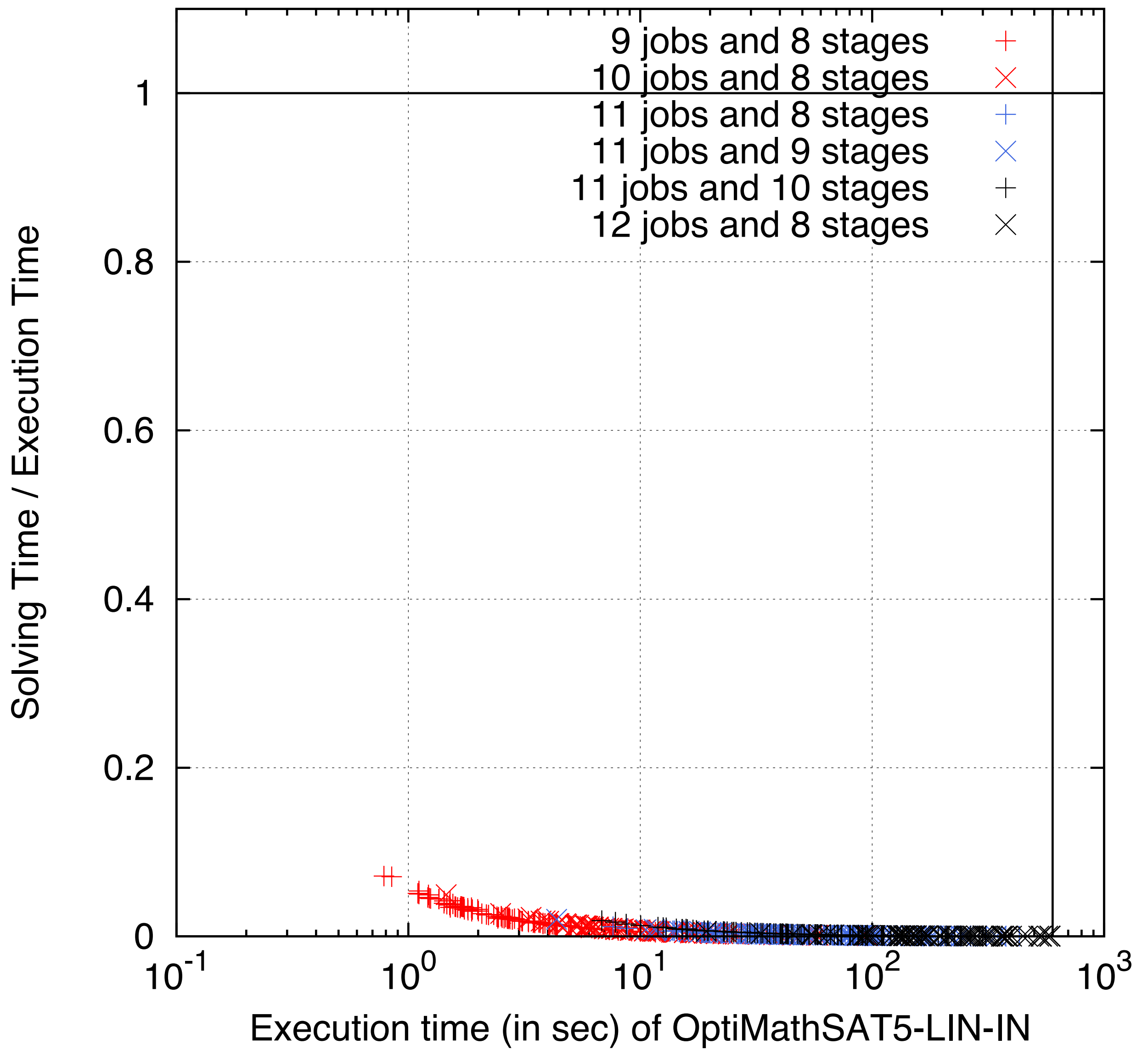

Figure 6.12: Comparing solving time with the execution time of OPTIMATHSAT-LIN-IN on "directly generated" instances of job-shop.

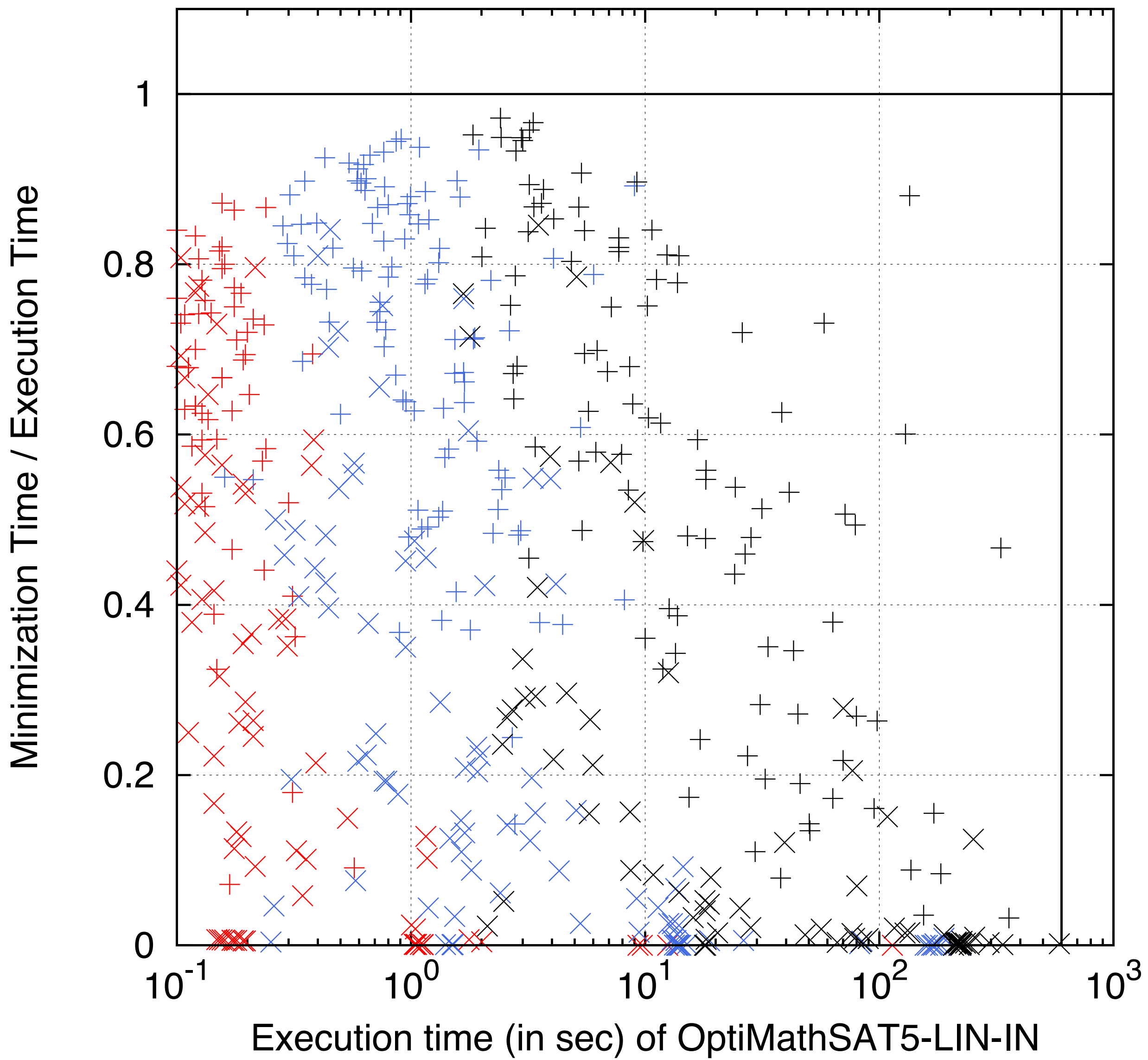

Figure 6.13: Comparing minimization time with the execution time of OPTIMATHSAT-LIN-IN on "directly generated" instances of strip-packing..

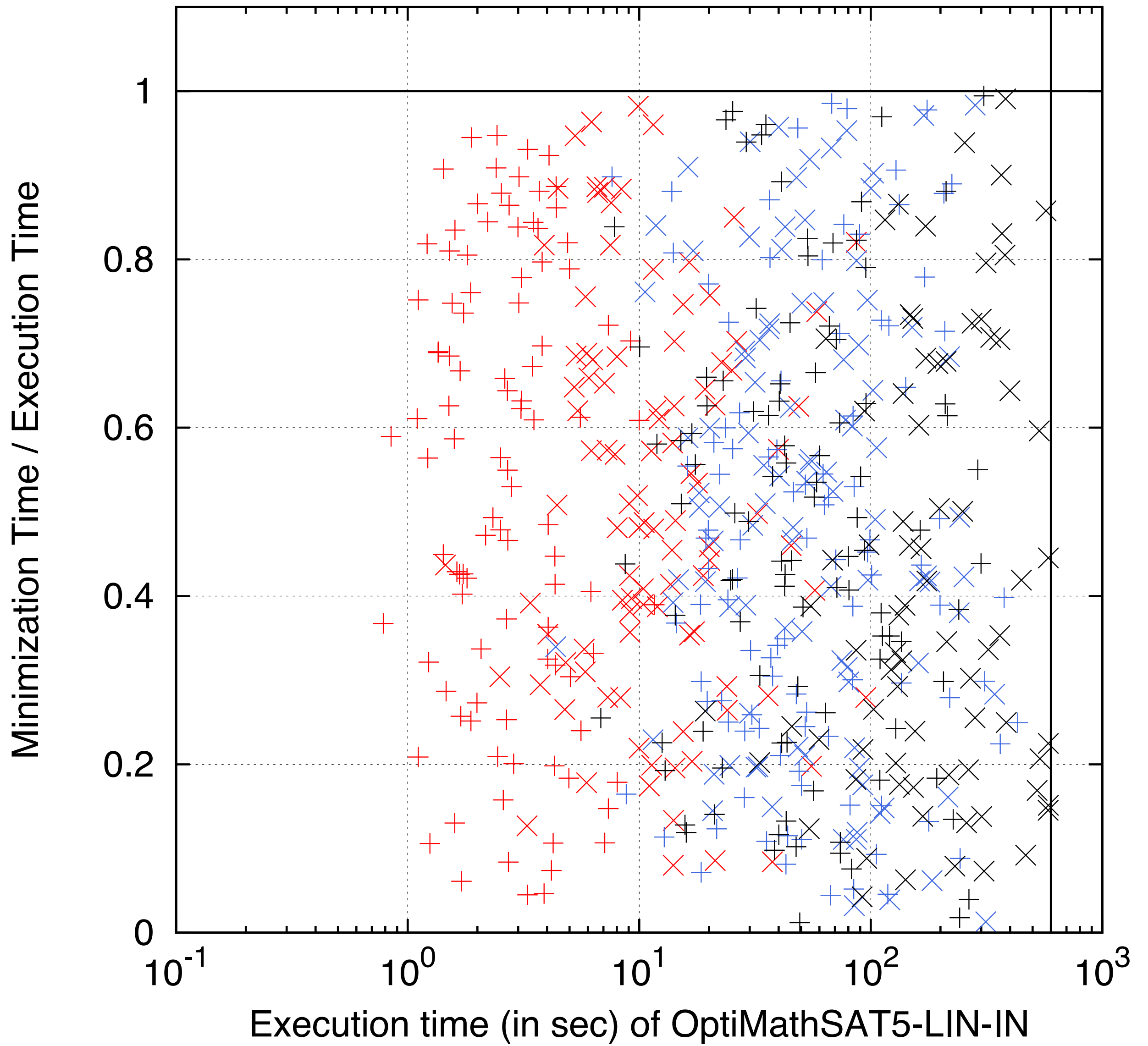

Figure 6.14: Comparing minimization time with the execution time of OPTIMATHSAT-LIN-IN on “directly generated” instances of job-shop.

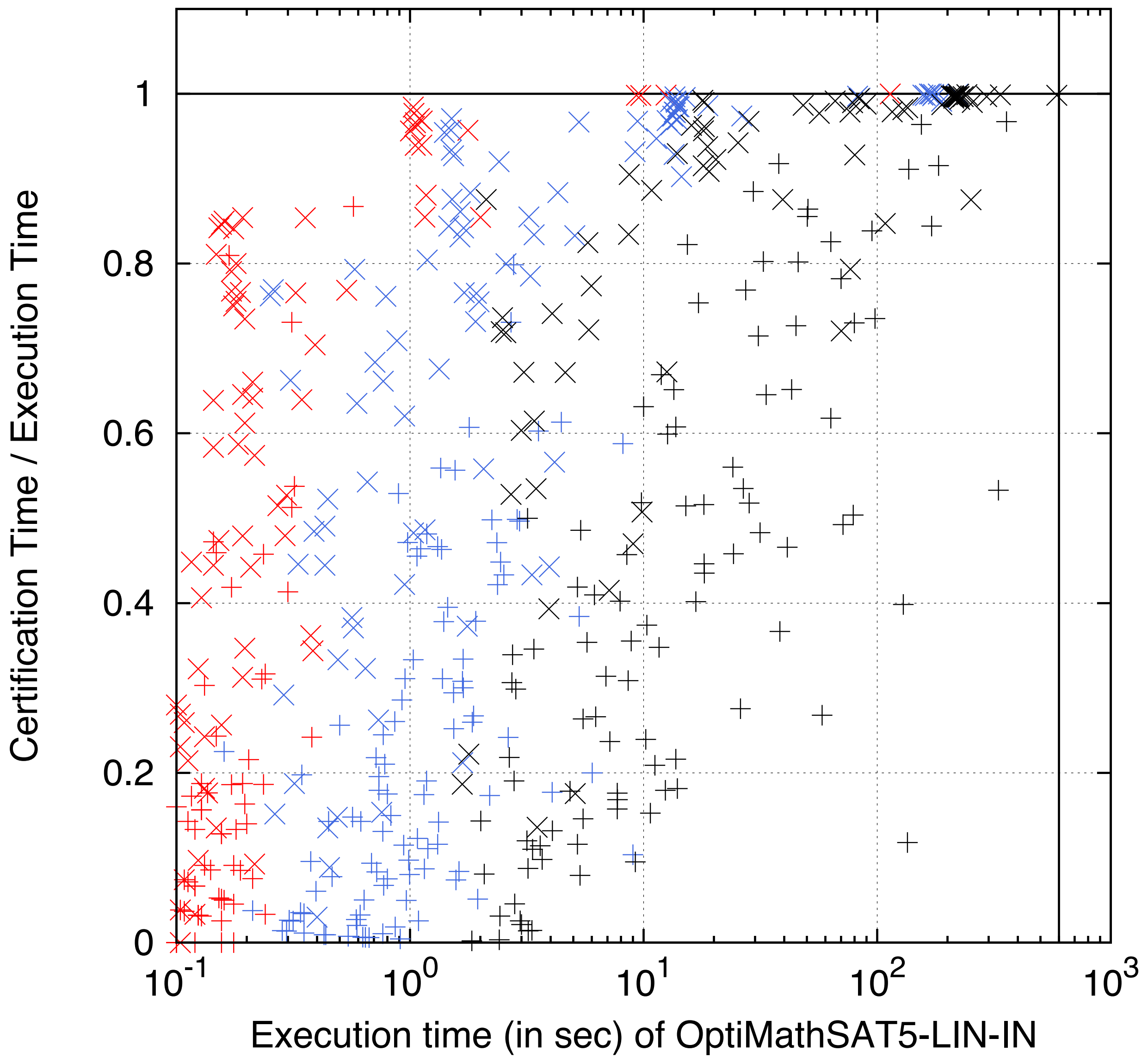

Figure 6.15: Comparing certification time with the execution time of OPTIMATHSAT-LIN-IN on "directly generated" instances of strip-packing.

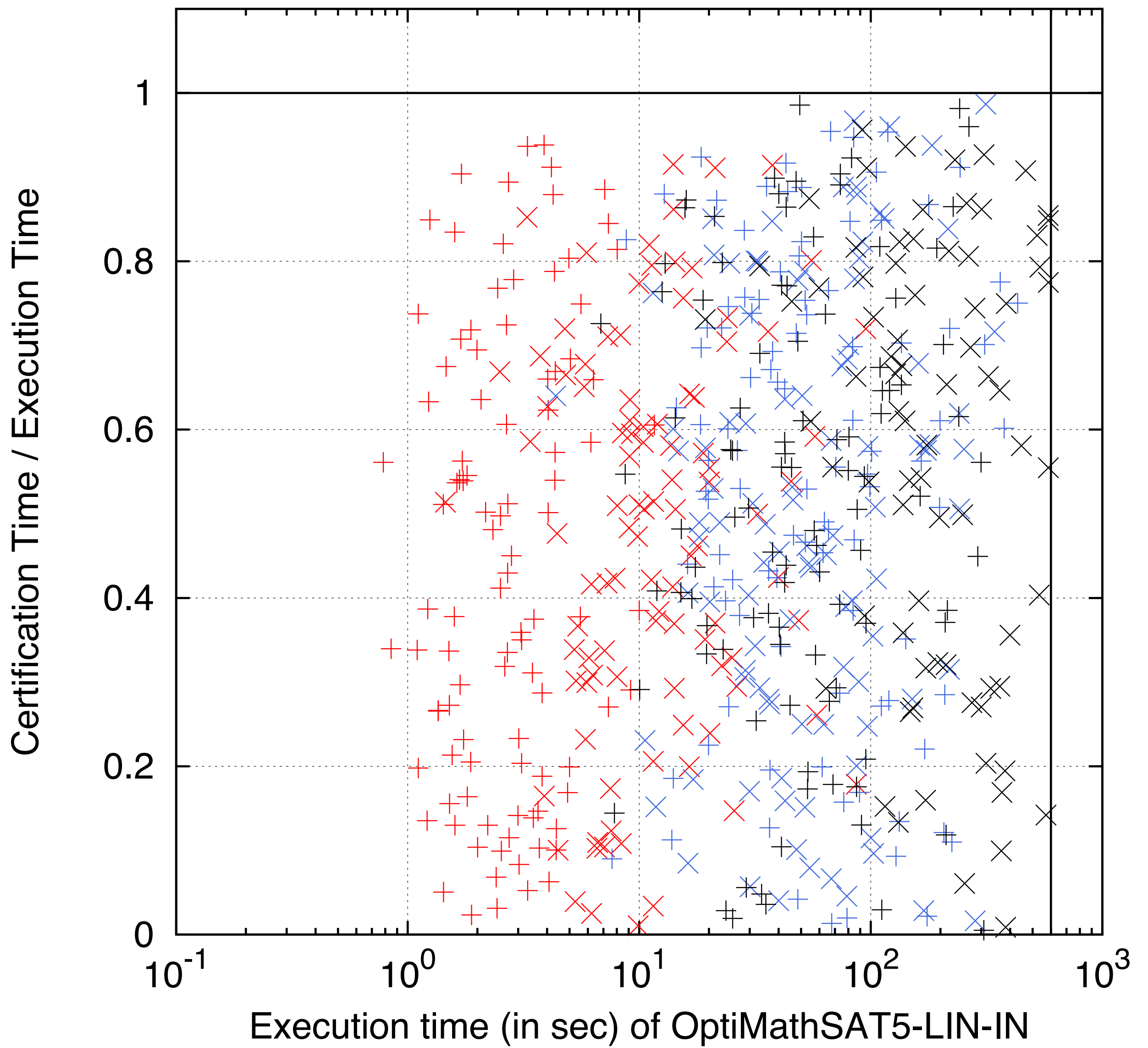

Figure 6.16: Comparing certification time with the execution time of OPTIMATHSAT-LIN-IN on "directly generated" instances of job-shop.

- 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 that 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 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.

6.3 Comparison on SMT-LIB Problems

As a second comparison, in Figures 6.17-6.21 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 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`. [17] 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

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

back, to be fed to OPTIMATHSAT-LIN-IN (4th and 5th row). We notice that this caused 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 3 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, whilst 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 4 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 above-mentioned `sal` instance with SMT2LGDP_1, with the default values the BM versions returned a wrong minimum value "0", the CH versions returned "unfeasible", whilst 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 the encodings used. [18]

[18] We also isolated a subproblem, small enough to be solved by hands, 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.

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

Figure 6.17: 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, # of instances terminating wrongly returning "unfeasible".

6.3.1 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 6.1). Notice that, unlike with the LGDP problems in §6.2, SMT-LIB problems do contain occurrences of strict [resp. non-strict] inequalities with positive [resp. negative] polarity.

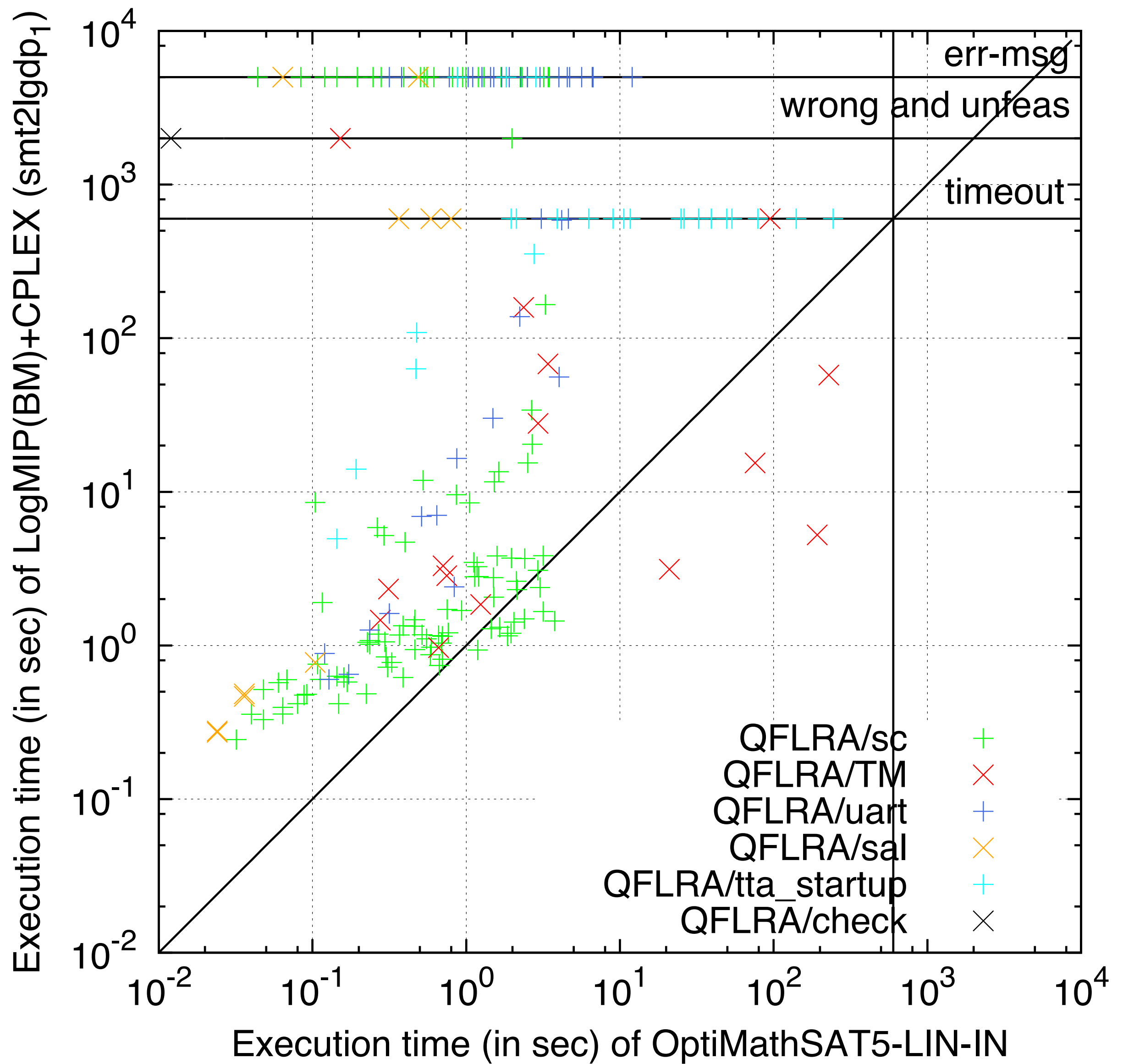

Figure 6.18: Pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable between OPTIMATHSAT-LIN-IN and LOGMIP(BM)+CPLEX instances between OPTIMATHSAT-LIN-IN on SMT2LGDP$_1$ encodings.

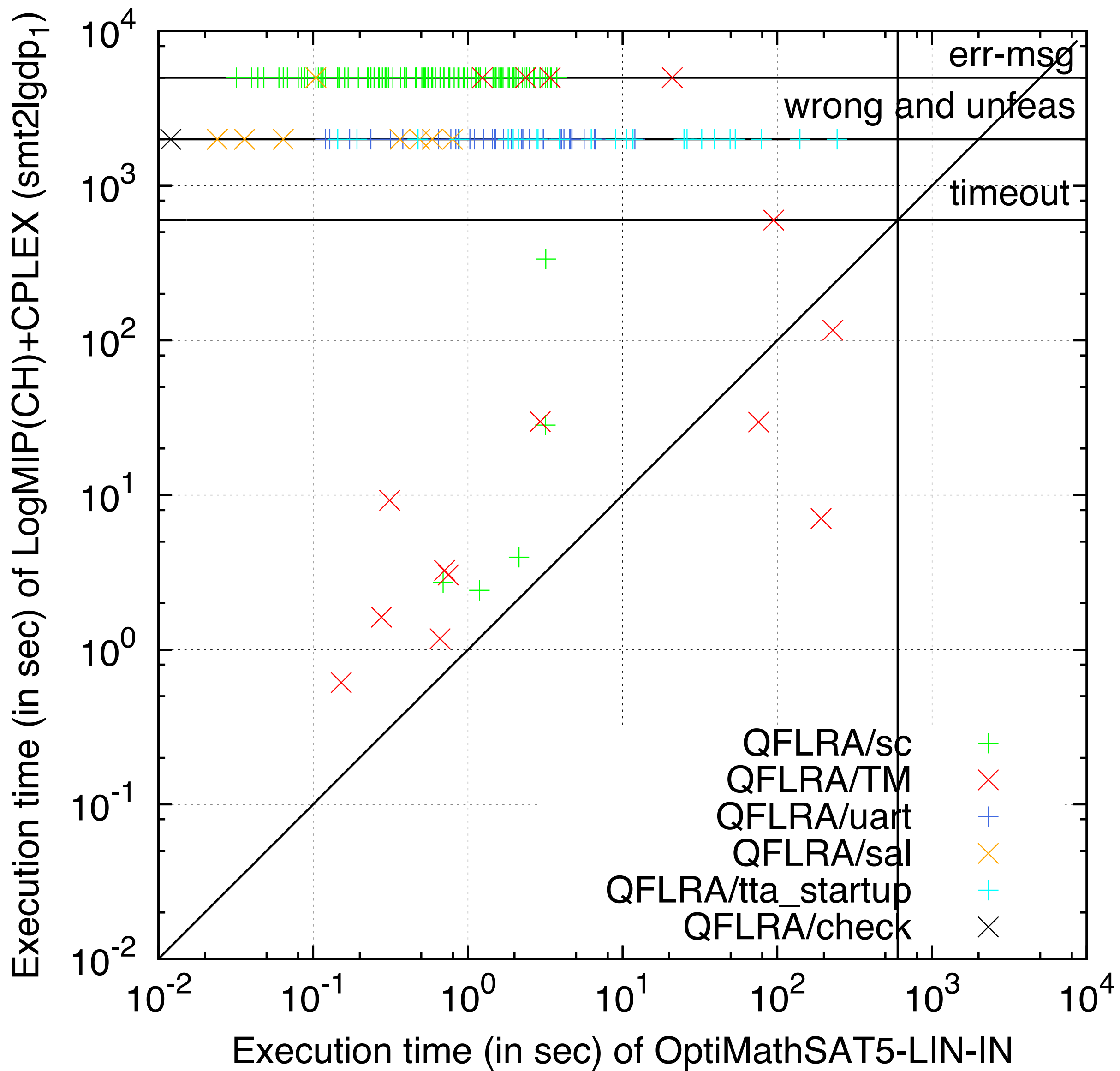

Figure 6.19: Pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable between OPTIMATHSAT-LIN-IN and LOGMIP(CH)+CPLEX instances between OPTIMATHSAT-LIN-IN on SMT2LGDP$_1$ encodings.

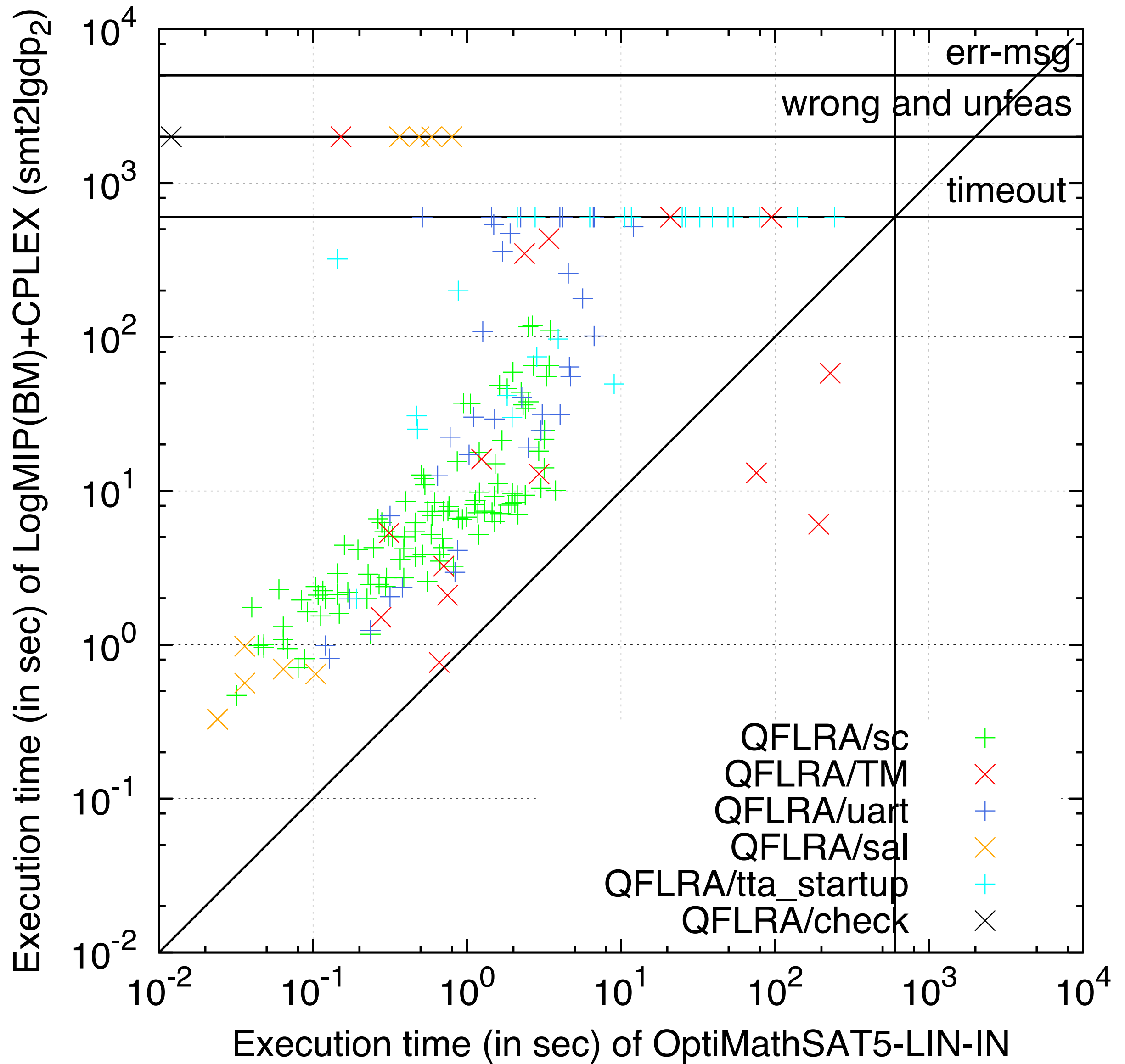

Figure 6.20: Pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable between OPTIMATHSAT-LIN-IN and LOGMIP(BM)+CPLEX instances between OPTIMATHSAT-LIN-IN on SMT2LGDP$_2$ encodings.

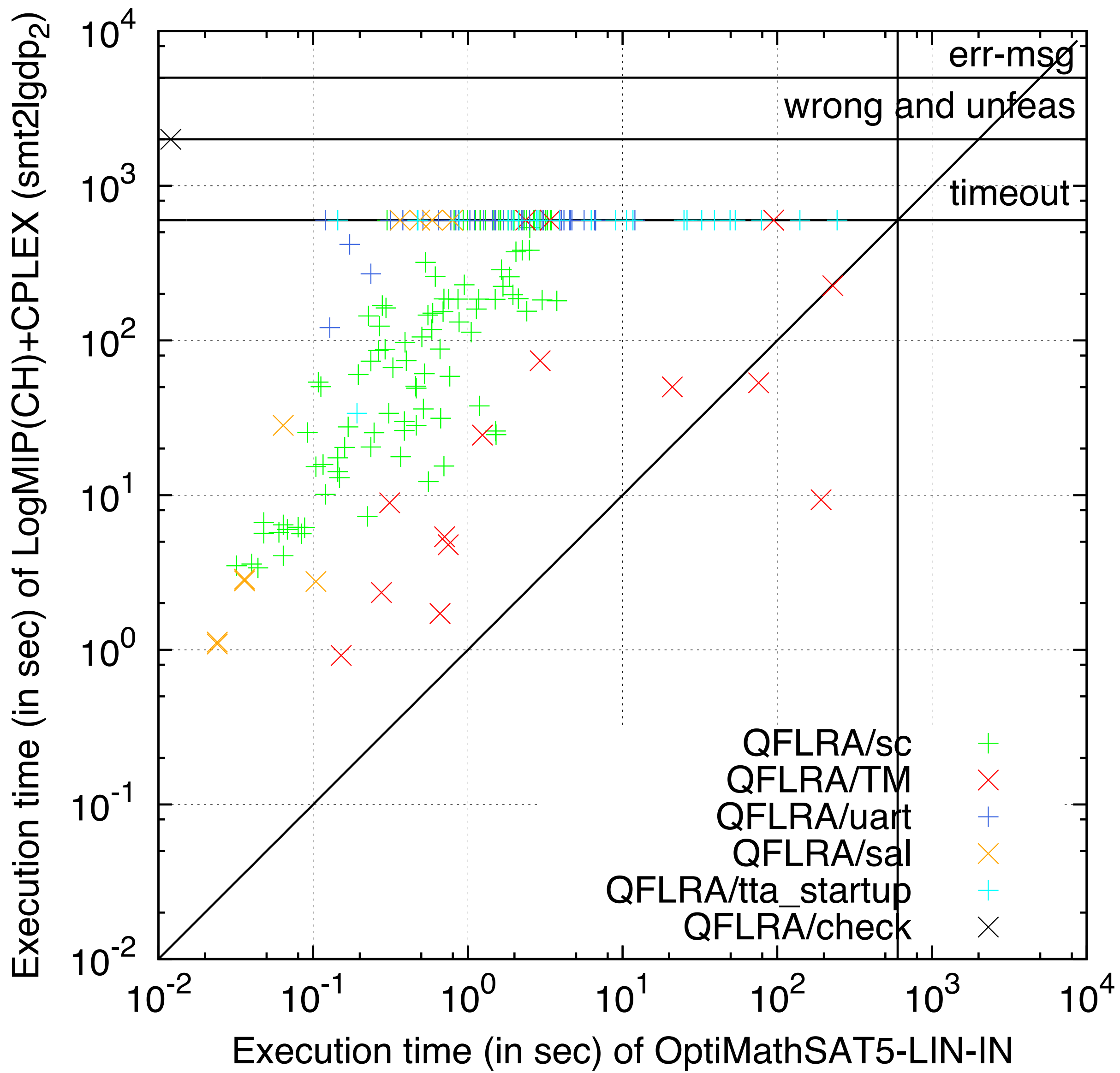

Figure 6.21: Pairwise comparisons on the smt-lib $\mathcal{LA}(\mathbb{Q})$ satisfiable between OPTIMATHSAT-LIN-IN and LOGMIP(CH)+CPLEX instances between OPTIMATHSAT-LIN-IN on SMT2LGDP$_2$ encodings.

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 by more than one order magnitude. We notice that on these problems SMT2LGDP_2 is generally more effective than SMT2LGDP_1 and less prone to errors.

6.4 Comparison on SAL Problems

As a third comparison, in Figures 6.22-6.24 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 [24] and K-Induction (K-IND) [84] — of a well-known parametric timed system, Fisher's Protocol [19].

BMC [resp. K-IND] takes a Finite-State Machine M, an invariant property Ψ and an integer bound k, and produces a propositional formula φ which is satisfiable [resp. unsatisfiable] if and only if there exists a k-step execution violating Ψ [resp. a k-step induction proof that Ψ is always verified]. The approach leverages to real-time systems by producing SMT($\mathcal{LA}(\mathbb{Q})$) formulas rather than purely-propositional ones (see, e.g., [11]).

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

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

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 a set of sufficiently-big values of $k \geq k^*$, [20] 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 which 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.

6.4.1 Discussion

The results are presented in Figures 6.22-6.24. 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 4 samples over 392 and returned 19 solutions with error messages and 1 solution with wrong minimum, whilst 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

[20]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	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

Figure 6.22: 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, # of instances terminating wrongly returning "unfeasible".

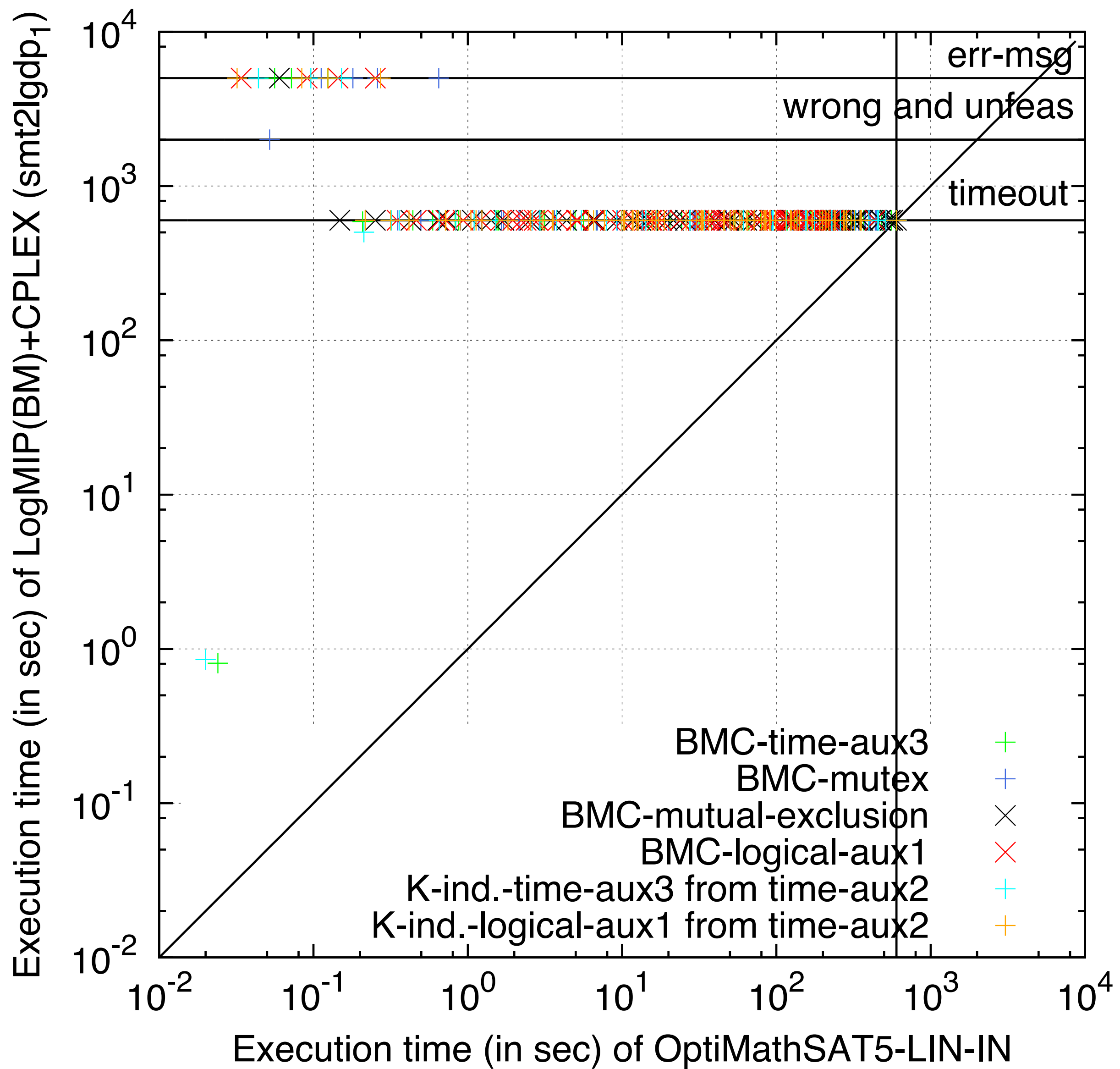

Figure 6.23: Comparison of the best configuration of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX on SMT2LGDP$_1$ encoding.

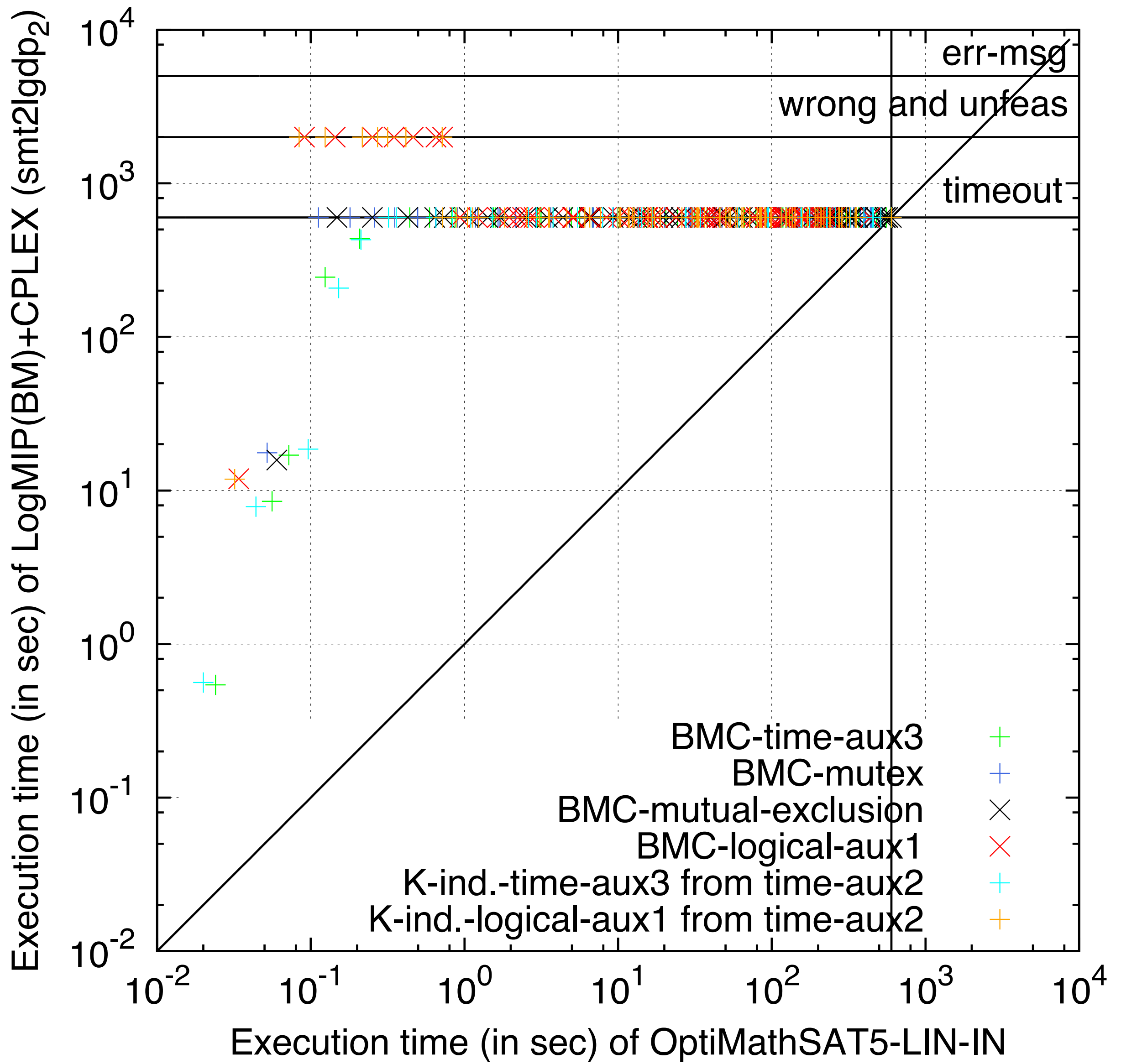

Figure 6.24: Comparison of the best configuration of OPTIMATHSAT (OPTIMATHSAT-LIN-IN) against LOGMIP(BM)+CPLEX on SMT2LGDP$_2$ encoding.

former dramatically outperforms the latter, no matter the encoding used.

6.5 Comparison on Pseudo-Boolean SMT Problems

As a fourth comparison, in Figures 6.25-6.28 we evaluate OPTIMATHSAT on the problem sets used in [31] against the usual GAMS tools and against a recent reimplementation on MATHSAT5 of the tool in [31], namely PB-MATHSAT, for SMT with Pseudo-Boolean constraints (see §4). [21] PB-MATHSAT is tested with both linear search and binary search strategies (denoted with "-LIN" and "-BIN" respectively).

As described in [31], the problems consists of partial weighted MaxSMT problems which 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 (4.7) in §4. These problems are further encoded into OMT($\mathcal{LA}(\mathbb{Q})$) problems by means of the encoding (4.6) in §4, and hence into LGDP problems by means of the usual two encodings.

6.5.1 Discussion

The results are presented in Figures 6.25-6.28. The three versions of OPTIMATHSAT solved respectively 630, 634 and 637 problems out of 675 problems overall, whilst 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 version 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 on, unfeasible results or wrong min-

[21] A comparison against the tool in [31] 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 [32].

Procedure	MaxSMT % SMT+PB generated from SMT-LIB/QF_LRA						
	#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

Figure 6.25: Results for OPTIMATHSAT, PB-MATHSAT and the GAMS tools, on the MaxSMT benchmarks from [31]. 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, # of instances terminating wrongly returning "unfeasible".

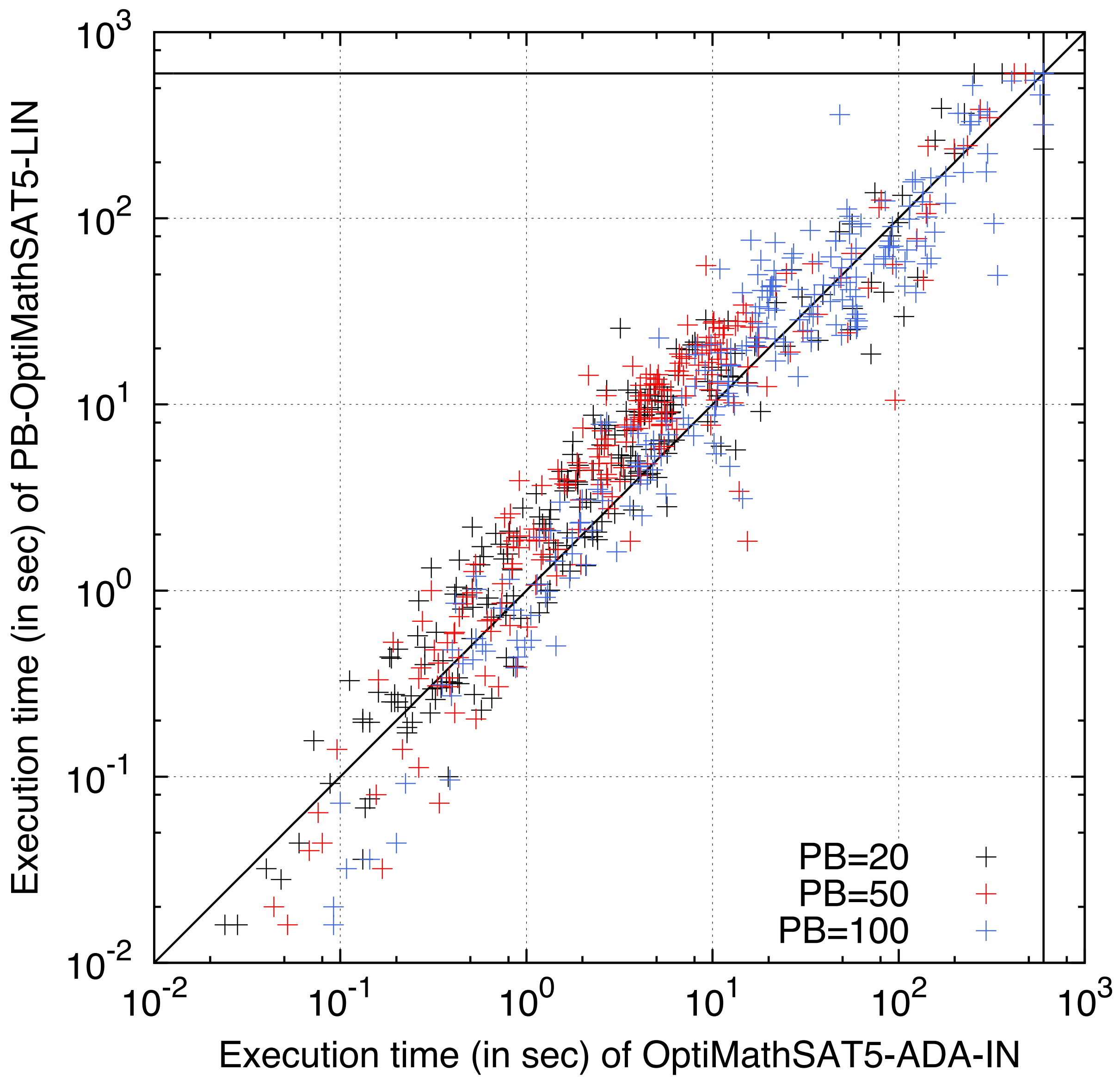

Figure 6.26: Comparison of the best configuration of OPTIMATHSAT, OPTIMATHSAT-ADA-IN, against the best configuration of PB-MATHSAT, PB-MATHSAT-LIN .

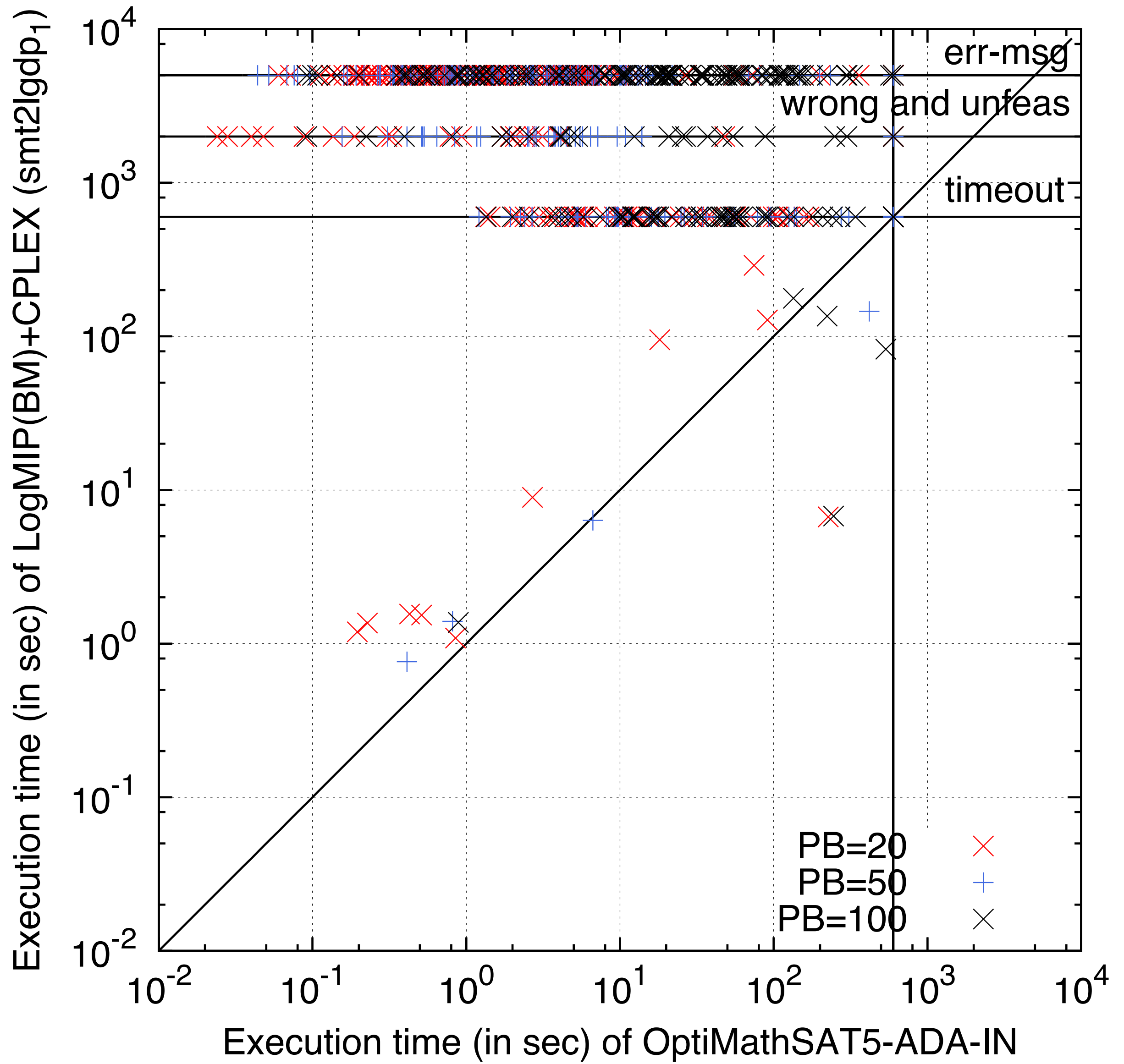

Figure 6.27: Comparison of the best configuration of OPTIMATHSAT, OPTIMATHSAT-ADA-IN, against the best configuration of GAMS tools, LOGMIP(BM)+CPLEX, on SMT2LGDP$_1$ encoding.

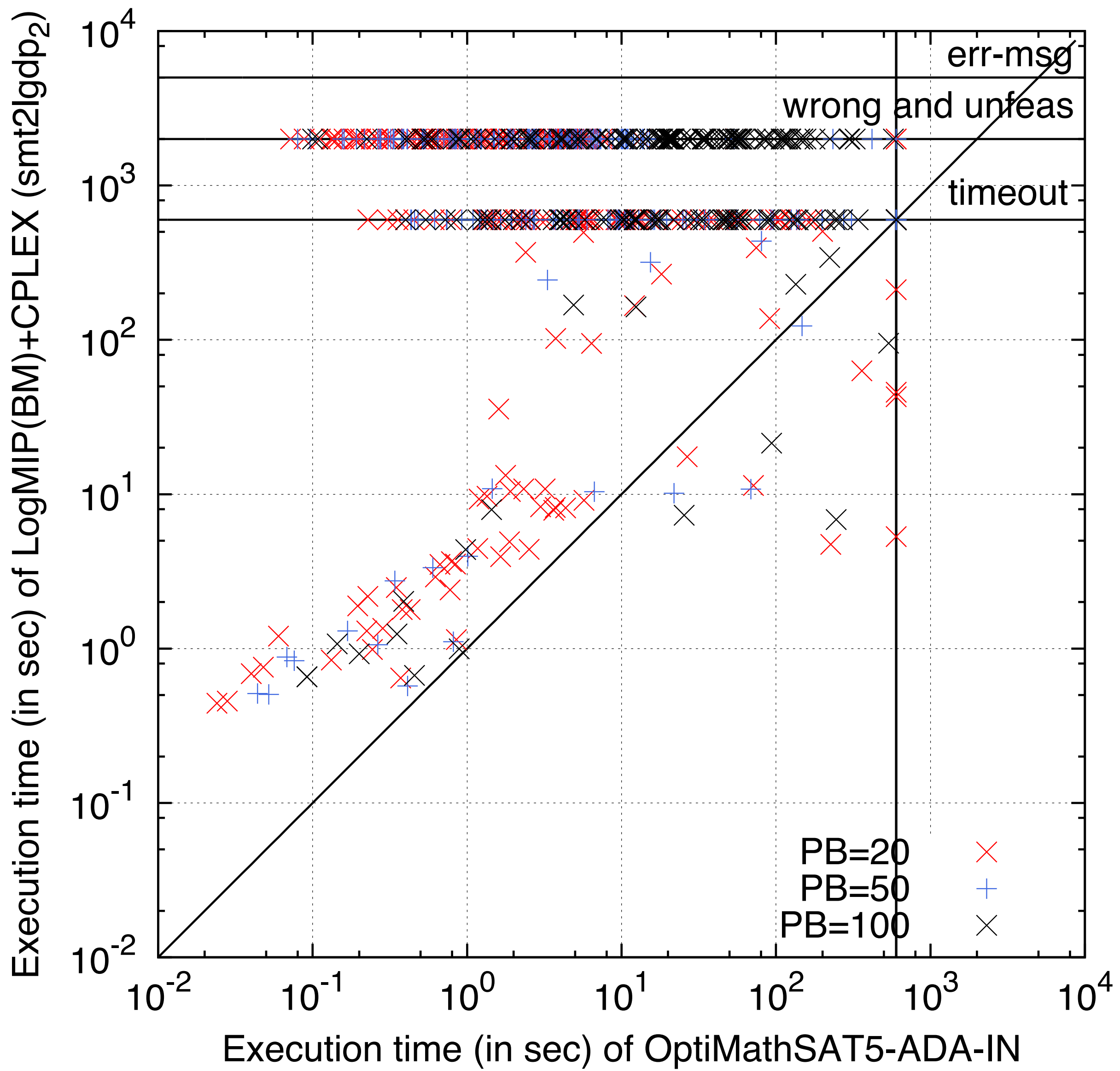

Figure 6.28: Comparison of the best configuration of OPTIMATHSAT, OPTIMATHSAT-ADA-IN, against the best configuration of GAMS tools, LOGMIP(BM)+CPLEX, on SMT2LGDP$_2$ encoding.

imum 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 6.4. Notice that, unlike with LGDP problems (see Remark 6.3) and in part also with SMT-LIB and SAL problems, with Pseudo-Boolean problems the cost variables occurs 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 Figures 6.26-6.28 compare the best version of OPTIMATHSAT with these of PB-MATHSAT and of the GAMS tools. We see that OPTIMATHSAT-ADA-IN performances are analogous to these of PB-MATHSAT-LIN , and drastically superior to these of GAMS tools with both encodings.

As a side note, in [32] 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 of the novel specialized MaxSMT tool presented there. We refer the reader to [32] for details.

6.6 Comparison on SYMBA Problems

As last comparison, in Figures 6.29-6.32, we evaluate the performance of OPTIMATHSAT against SYMBA, which has been presented very recently by Li et al. in [60] (as introduced in §3.3). In our experimentation we consider two versions of SYMBA:

Procedure	Formulas from SYMBA tool [60]						
	#inst.	#term.	#inst. $= -\infty$	time $= -\infty$	#inst. $\neq -\infty$	time $\neq -\infty$	time
OPTIMATHSAT5-LIN-IN	28613	25880	7784	4934	18096	3827	8792
OPTIMATHSAT5-BIN-IN	28613	25880	7784	4932	18096	3826	8759
OPTIMATHSAT5-ADA-IN	28613	25880	7784	4932	18096	3828	8760
SYMBA(100)	28613	28583	10485	31102	18098	9454	40547
SYMBA(40)+OPT-Z3	28613	28613	10515	8163	18098	9128	17291

Figure 6.29: Results for OPTIMATHSAT and SYMBA tools, on the benchmarks from [60]. The columns report respectively: # of instances considered, # of instances terminating within the timeout, # of instances terminating within the timeout with infinite cost, execution time of terminated instances with infinite cost, # of instances terminating within the timeout with finite cost, execution time of terminated instances with finite cost, the execution time.

1. SYMBA(100), which uses the Z3 SMT solver [3] as black-box for satisfiability checking (default configuration of SYMBA);

2. SYMBA(40)+OPT-Z3, which uses Z3 with a modified linear arithmetic solver for performing optimization (OPT-Z3), like our "inline" version (best configuration on the benchmark set in [60]).

The benchmark set consists of formulas used in [60], which were generated from a set of C programs used in the 2013 Software Verification Competition. While Li et al. presented an evaluation on formulas with "multiple-objectives" (by calling OPTIMATHSAT multiple times per benchmark, each time with a different objective), we consider formulas with a single cost variable which are obtained by splitting each formula φ involving costs $\{\mathsf{cost}_1, ..., \mathsf{cost}_k\}$, into k formulas φ_i, each considering one cost variable cost_i (obtaining 28613 instances).

6.6.1 Discussion

The results are presented in Figures 6.29-6.35.

The three versions of OPTIMATHSAT solved 25880 problems out of 28613 problems overall, whilst the two versions of SYMBA solved respectively 28583 and 28613. The table of Figure §6.29 and the scatter plots of Figures 6.30 and 6.31 show that, apparently, the SYMBA tools outperform all versions of OPTIMATHSAT. This difference in performance can be motivated by two reasons.

First, an analysis of the benchmark set reveals a high number ($\approx 37\%$) of formulas whose cost is $-\infty$. Figures 6.34 and 6.35, and Figures 6.32 and 6.33 compare the tools on two set of formulas, respectively, one with finite cost and the other with infinite cost. (We refer to the former kind of formulas as "bounded" and to the latter as "unbounded".) We see that on "bounded" formulas OPTIMATHSAT (Figures 6.34 and 6.35) performs better than SYMBA tools on most benchmarks (only two instances were not solved within the timeout by OPTIMATHSAT). On "unbounded" formulas (Figures 6.32 and 6.33), instead, SYMBA outperforms OPTIMATHSAT (which does not solve $\approx 74\%$ of instances) because the former exploits an ad-hoc heuristics which recognizes infinite costs very effectively (OPTIMATHSAT uses the technique of the standard Simplex algorithm instead).

Second, OPTIMATHSAT is built on top of MATHSAT5, whereas SYMBA uses Z3, which is known to perform better on SMT($\mathcal{LA}(\mathbb{Q})$) formulas[22]. In order to normalize the results wrt. the performance of the underlying SMT solver, we consider the commonly-terminated formulas and plot the ratio of the running time over the time required to the underling solver to check the unsatisfiability of $\varphi_i \wedge (\mathsf{cost}_i < m_i)$, where m_i is the minimum cost. The results are reported in Figures 6.36 and 6.37. We can see that the normalized performance of OPTIMATHSAT is much better than that of SYMBA.

[22] Z3 won the SMT-COMP in 2012 on QF_LRA benchmarks [1]

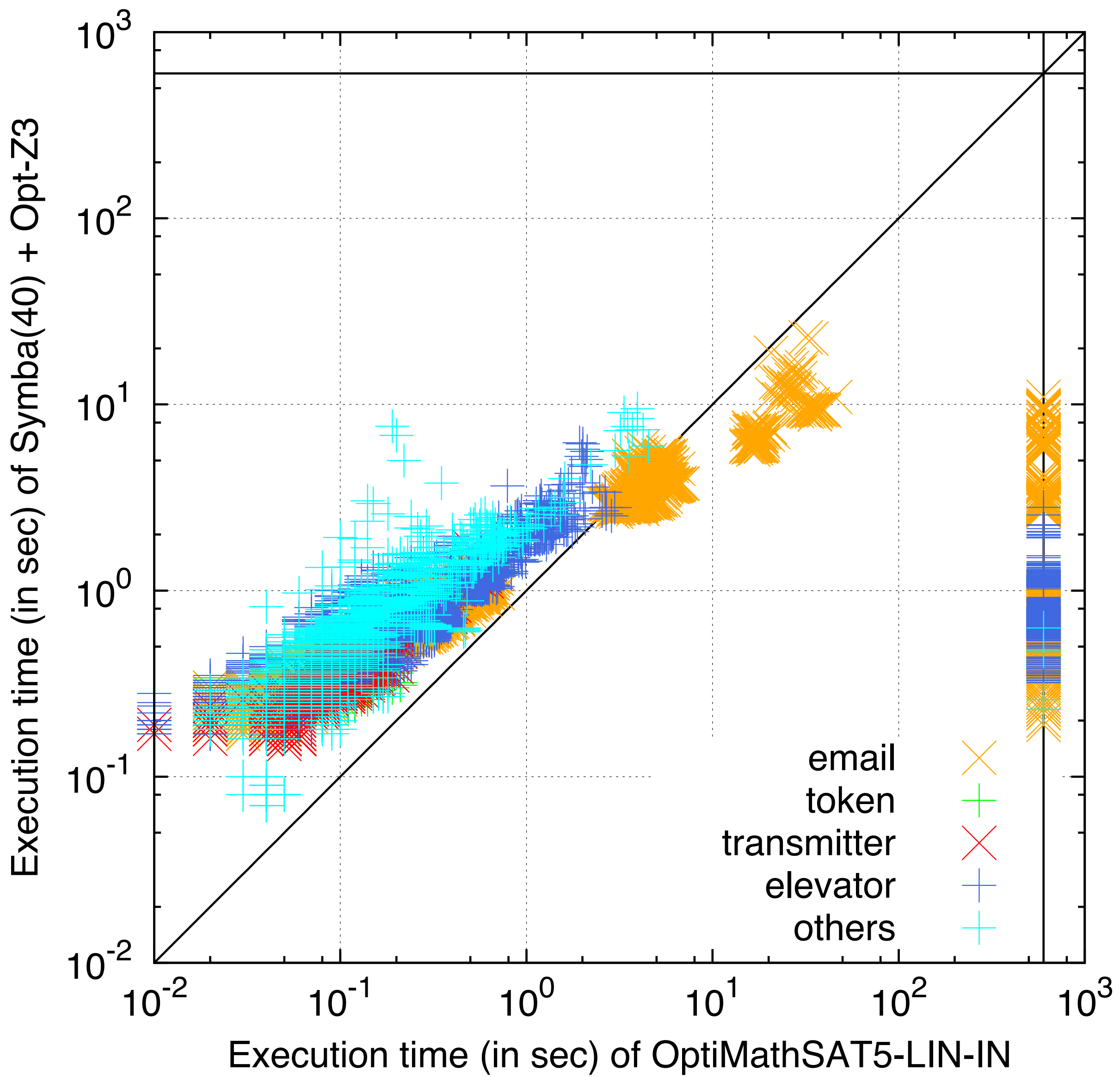

Figure 6.30: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against the best configuration of SYMBA, SYMBA(40)+OPT-Z3.

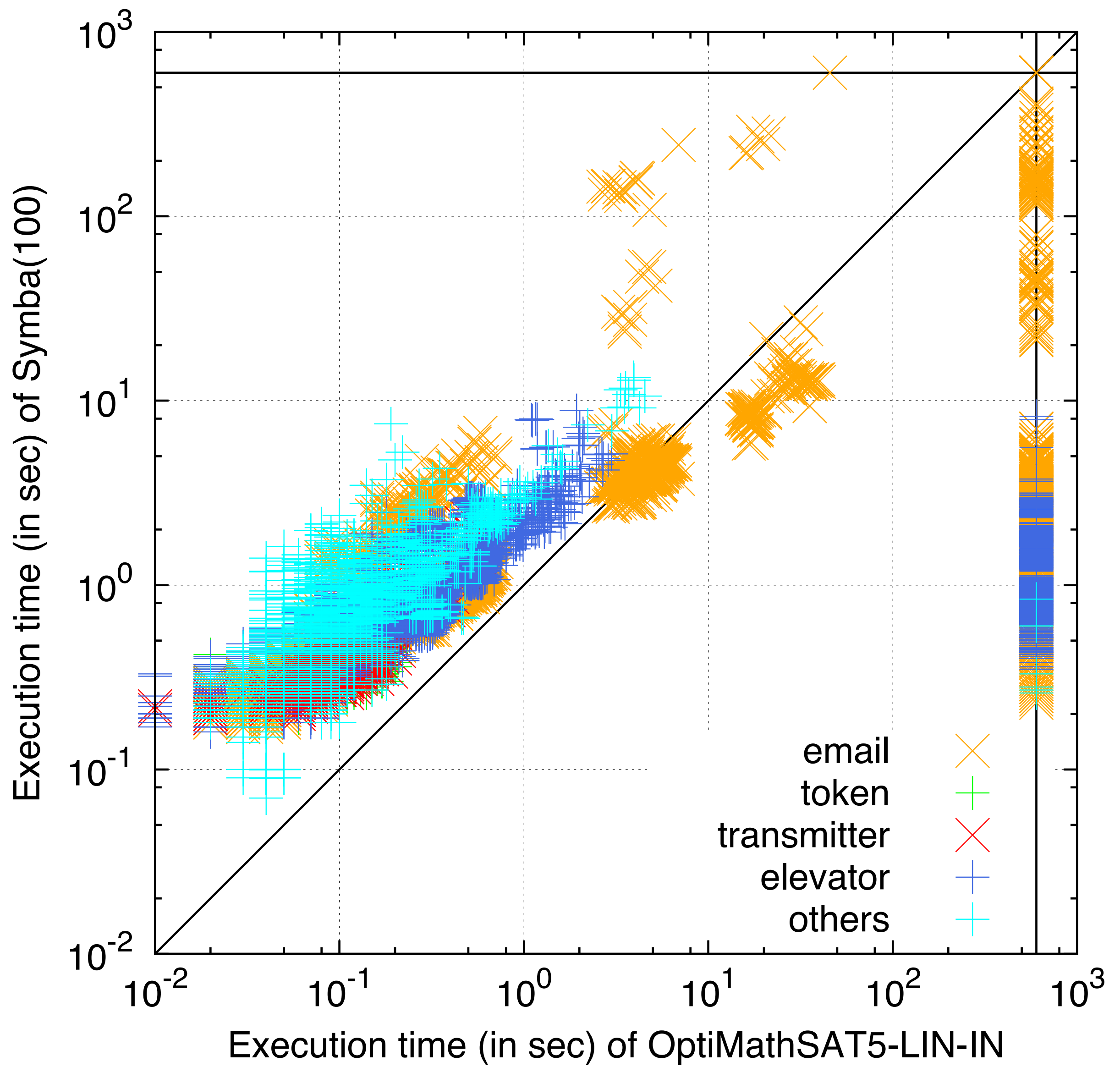

Figure 6.31: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against SYMBA(100).

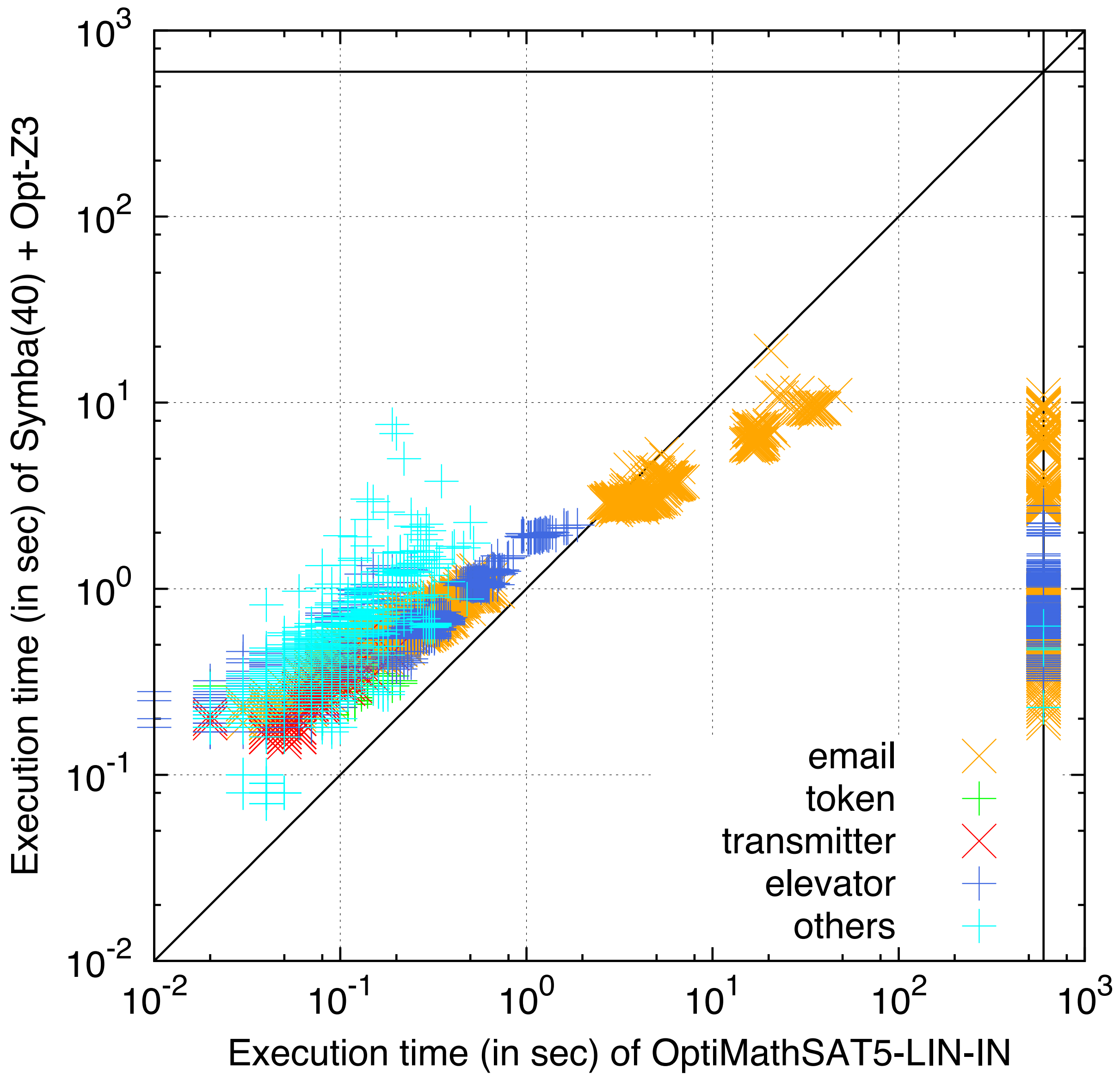

Figure 6.32: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against the best configuration of SYMBA, SYMBA(40)+OPT-Z3, on formulas whose cost is infinite.

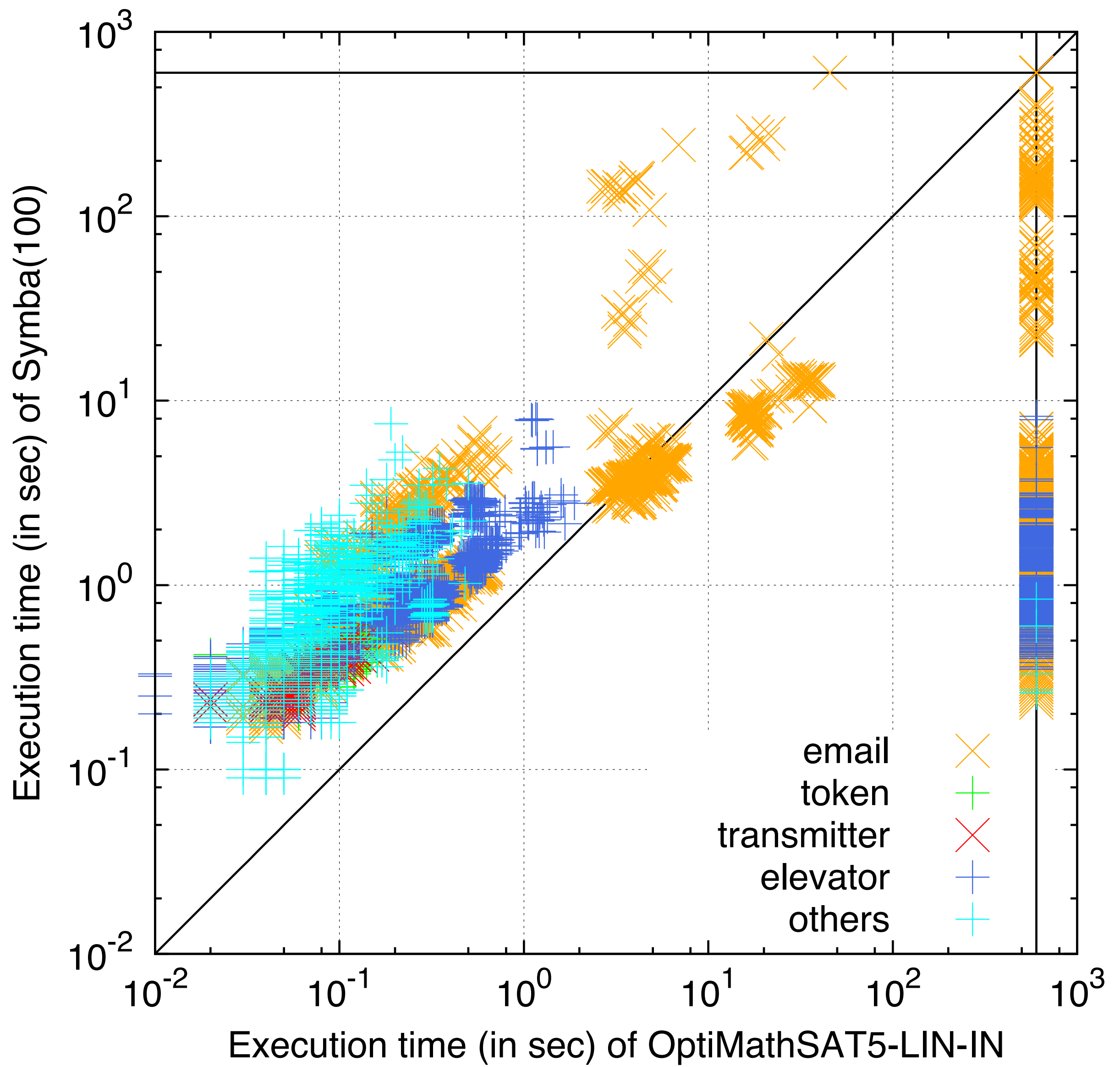

Figure 6.33: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against SYMBA(100), on formulas whose cost is infinite.

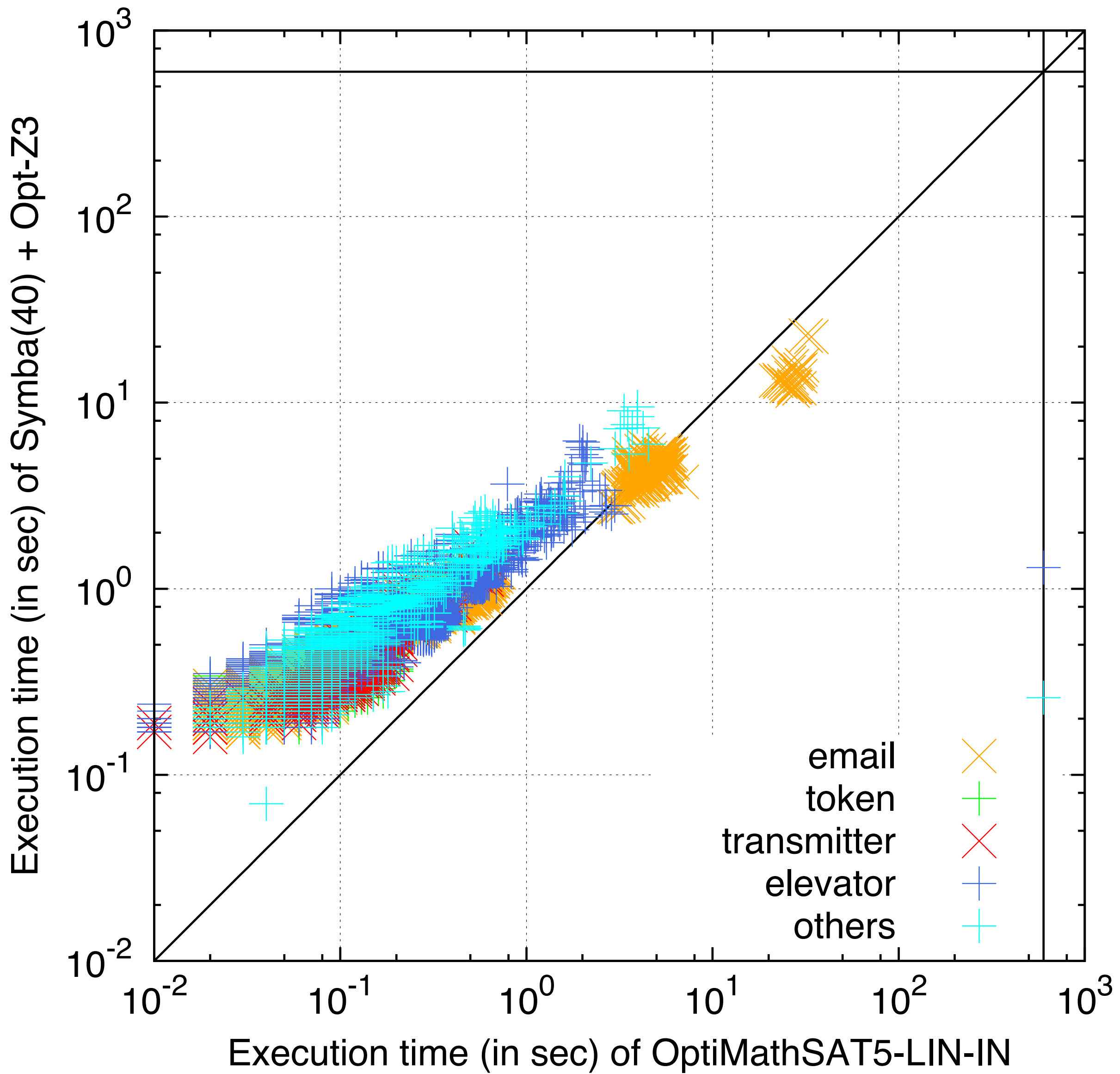

Figure 6.34: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against the best configuration of SYMBA, SYMBA(40)+OPT-Z3, on formulas whose cost is finite.

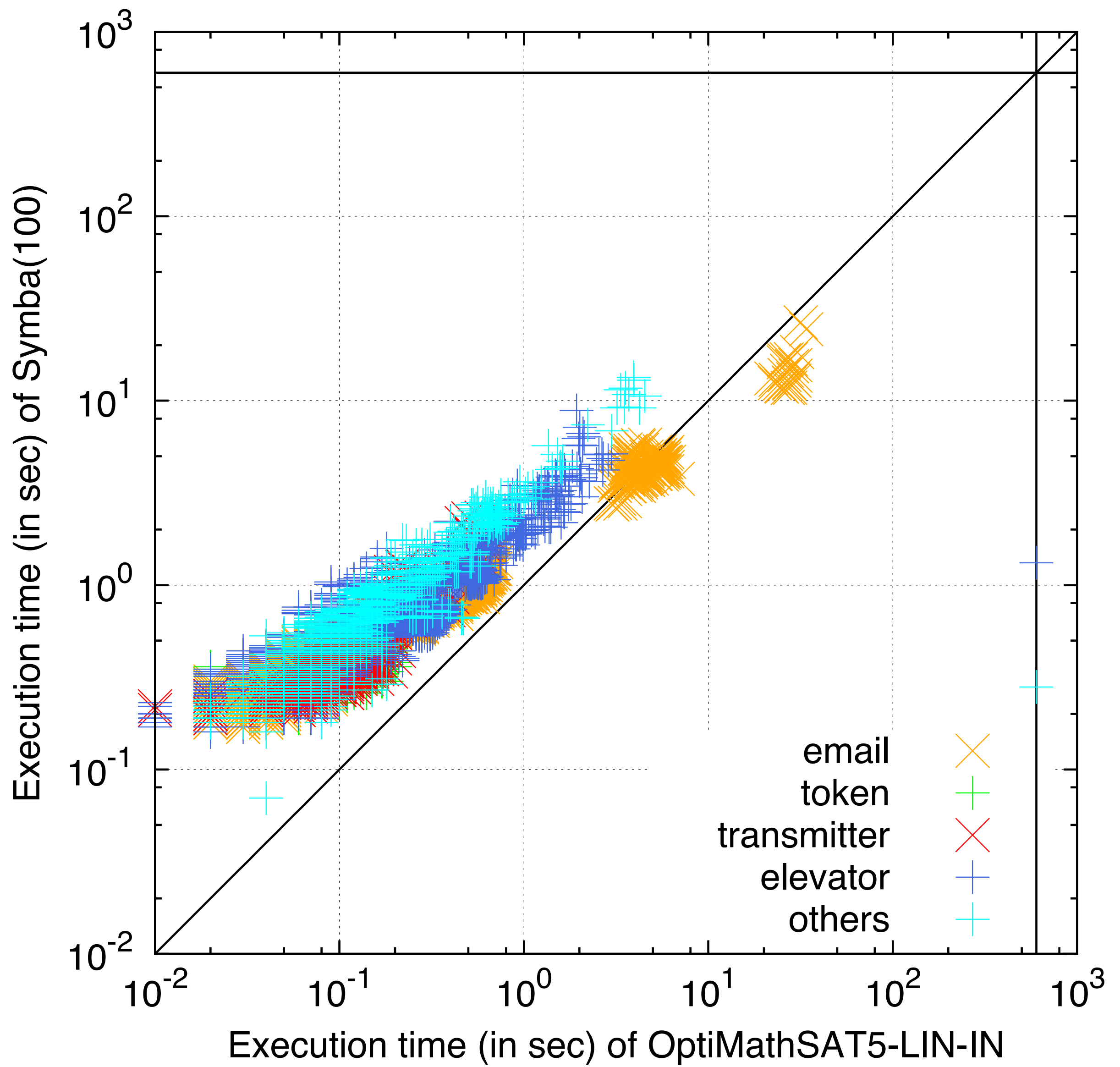

Figure 6.35: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against SYMBA(100), on formulas whose cost is finite.

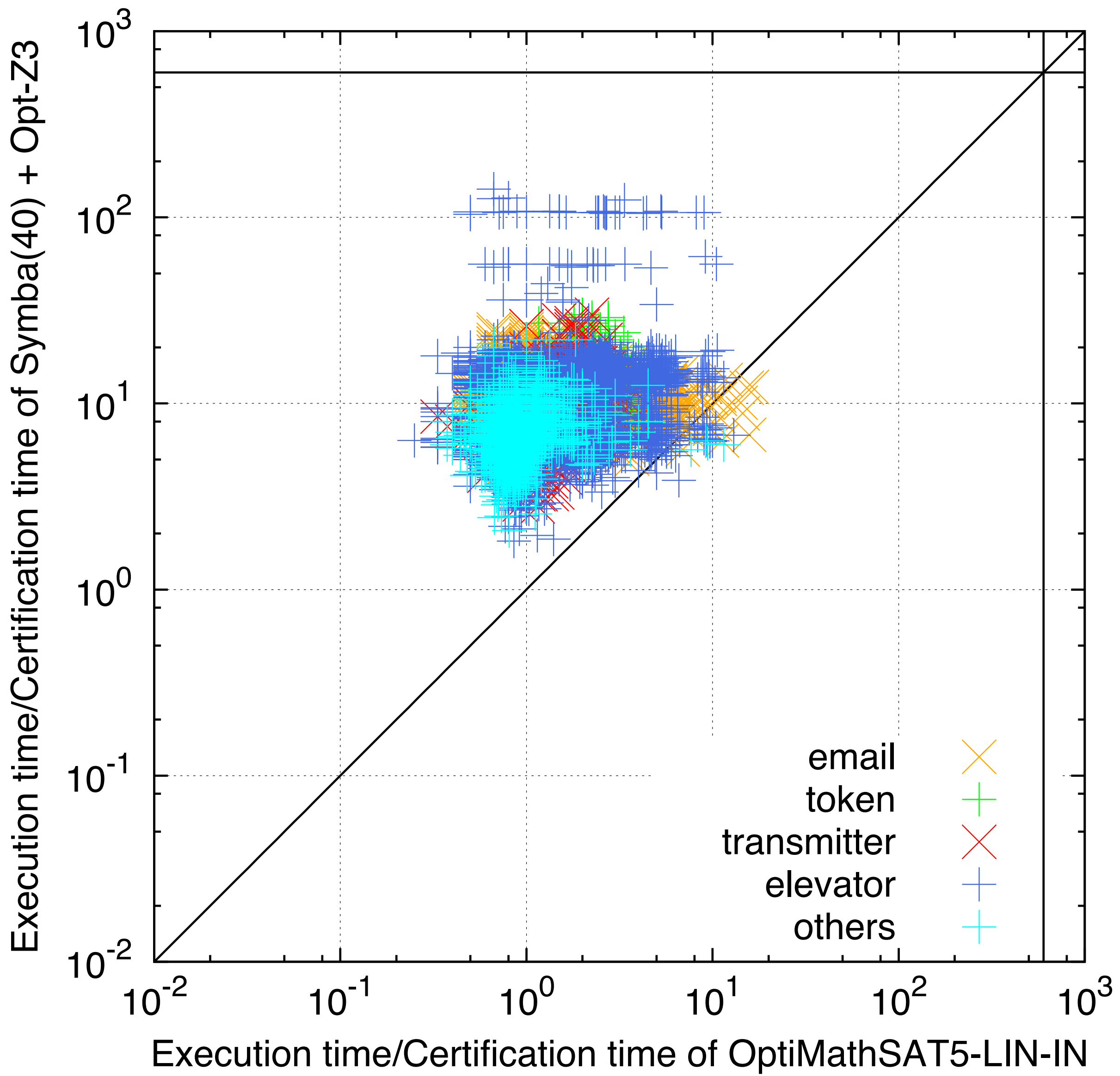

Figure 6.36: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATH-SAT-LIN-IN, against the best configuration of SYMBA, SYMBA(40)+OPT-Z3, on commonly terminated formulas.

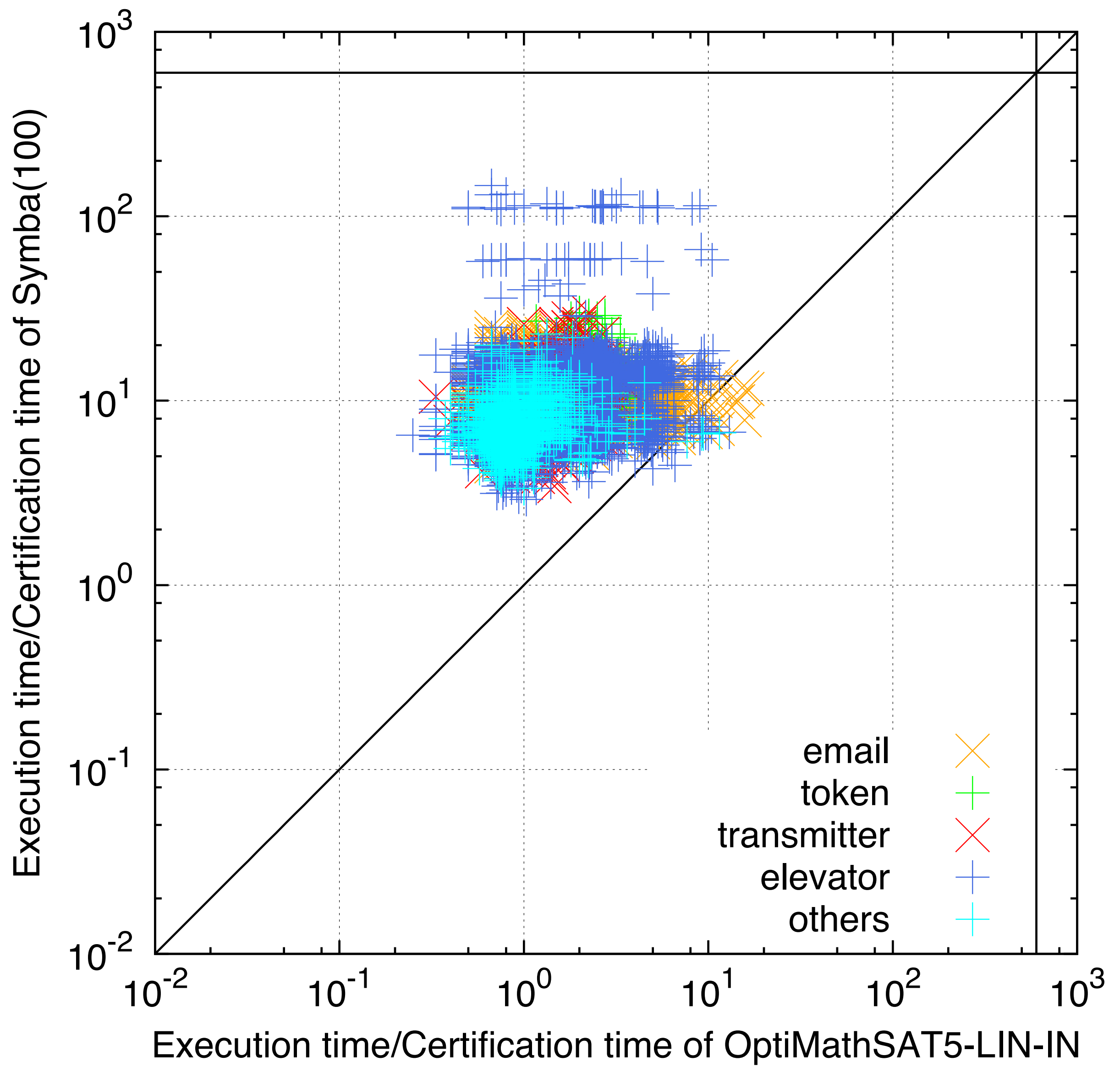

Figure 6.37: Comparison of the linear search configuration of OPTIMATHSAT, OPTIMATHSAT-LIN-IN, against SYMBA(100), on commonly terminated formulas.

Chapter 7

Stochastic Local Search in SMT

Inspired by the idea of "partially-invisible" SAT formulas (§7.1), we conceived a novel and general architecture for integrating a $\mathcal{T}$-Solver with a SLS SAT solver, based on the widely-used WalkSAT algorithm, resulting into a SLS-based SMT solver, which we call WALKSMT. We also analyze the differences between the interaction of a $\mathcal{T}$-solver with a CDCL-based and a SLS-based SAT solver, and we introduce and discuss a group of enhanced techniques aimed at improving the synergy between an SLS solver and the $\mathcal{T}$-solver.

This chapter is structured as follows. Section §7.1 introduces the idea which led to our work, section (§7.2) presents a basic schema of WALKSMT, and the last two sections §7.3 and §7.4 focus on, respectively, discussing and optimizing the integration of the SLS solver and the $\mathcal{T}$-solver.

Disclaimer. The work presented in this chapter (and also in §8) was done in collaboration with Alberto Griggio, Quoc Sang Phan and Roberto Sebastiani and was published in [46, 47].

7.1 Intuition

Our work is based on the following simple observation. In principle, from the perspective of a SAT solver, an SMT problem instance φ can be seen as the problem of solving a *partially-invisible* CNF SAT formula $\varphi^p \wedge \tau^p$, s.t. the "visible" part φ^p is the Boolean abstraction of φ and the "invisible" part τ^p is (the Boolean abstraction of) the set τ of all the $\mathcal{T}$-lemmas providing the obligations induced by the theory $\mathcal{T}$ on the $\mathcal{T}$-atoms of φ [1] (see Example 7.1). Thus, every assignment μ^p s.t. $\mu^p \models \varphi^p$ is $\mathcal{T}$-unsatisfiable iff μ^p falsifies some non-empty set of clauses $\{c_1^p, ..., c_n^p\} \subseteq \tau^p$.

Consequently, we can see a traditional lazy SMT solver as a CDCL SAT solver which knows φ^p but not τ^p: whenever a model μ^p for φ^p is found, it is passed to a $\mathcal{T}$-Solver which (behaves as if it) knows τ^p, and hence checks if μ^p falsifies some clause $c_i^p \in \tau^p$: if this is the case, it returns one (or more) such clause(s) c_i^p, which is then used to drive the future search and which is optionally added to φ^p.

Example 7.1. Consider the following SMT($\mathcal{LA}(\mathbb{Q})$) formula ϕ and its Boolean abstraction ϕ^p. We can see ϕ^p as a "partially-invisible" formula $\phi^p \wedge \tau^p$, where τ^p is the Boolean abstraction of τ which consists of all possible $\mathcal{T}$-lemmas on the $\mathcal{T}$-atoms of φ.

[1] Although not stated explicitly, this idea was exploited in part also in [34].

	ϕ:		ϕ^p:
c_1:	$\{A_1\}$	c_1:	$\{A_1\}$
c_2:	$\{\neg A_1 \vee (x-z>4)\}$	c_2:	$\{\neg A_1 \vee B_1\}$
c_3:	$\{\neg A_3 \vee A_1 \vee (y \geq 1)\}$	c_3:	$\{\neg A_3 \vee A_1 \vee B_2\}$
c_4:	$\{\neg A_2 \vee \neg(x-z>4) \vee \neg A_1\}$	c_4:	$\{\neg A_2 \vee \neg B_1 \vee \neg A_1\}$
c_5:	$\{(x-y \leq 3) \vee \neg A_4 \vee A_5\}$	c_5:	$\{B_3 \vee \neg A_4 \vee A_5\}$
c_6:	$\{\neg(y-z \leq 1) \vee (x+y=1) \vee \neg A_5\}$	c_6:	$\{\neg B_4 \vee B_5 \vee \neg A_5\}$
c_7:	$\{A_3 \vee \neg(x+y=0) \vee A_2\}$	c_7:	$\{A_3 \vee \neg B_6 \vee A_2\}$
c_8:	$\{\neg A_3 \vee (z+y=2)\}$	c_8:	$\{\neg A_3 \vee B_7\}$
τ:	(all possible $\mathcal{T}$-lemmas on the $\mathcal{T}$-atoms of ϕ)	τ^p:	
c_9:	$\{\neg(x+y=0) \vee \neg(x+y=1)\}$	c_9:	$\{\neg B_6 \vee \neg B_5\}$
c_{10}:	$\{\neg(x-z>4) \vee \neg(x-y \leq 3) \vee \neg(y-z \leq 1)\}$	c_{10}:	$\{\neg B_1 \vee \neg B_3 \vee \neg B_4\}$
c_{11}:	$\{(x-z>4) \vee (x-y \leq 3) \vee (y-z \leq 1)\}$	c_{11}:	$\{B_1 \vee B_3 \vee B_4\}$
c_{12}:	$\{\neg(x-z>4) \vee \neg(x+y=1) \vee \neg(z+y=2)\}$	c_{12}:	$\{\neg B_1 \vee \neg B_5 \vee \neg B_7\}$
c_{13}:	$\{\neg(x-z>4) \vee \neg(x+y=0) \vee \neg(z+y=2)\}$	c_{13}:	$\{\neg B_1 \vee \neg B_6 \vee \neg B_7\}$
...	...	...	...

$$B_1 \stackrel{\text{def}}{=} (x-z>4), \quad B_2 \stackrel{\text{def}}{=} (y \geq 1), \quad B_3 \stackrel{\text{def}}{=} (x-y \leq 3), \quad B_4 \stackrel{\text{def}}{=} (y-z \leq 1),$$
$$B_5 \stackrel{\text{def}}{=} (x+y=1), \quad B_6 \stackrel{\text{def}}{=} (x+y=0), \quad B_7 \stackrel{\text{def}}{=} (z+y=2).$$

Consider the formula φ [resp φ^p] which is obtained from ϕ [resp ϕ^p] after pre-processing (see §7.4), we can see that the truth assignment μ_1 is $\mathcal{T}$-unsatisfiable and that its Boolean refinement μ_1^p satisfies the formula φ^p but violates the clauses c_{10} and c_{12} in τ^p (i.e. $\mu_1^p \not\models \varphi^p \wedge \tau^p$).

	φ:		φ^p:
c_2:	$\{(x-z>4)\}$	c_2:	$\{B_1\}$
c_5:	$\{(x-y \leq 3) \vee \neg A_4 \vee A_5\}$	c_5:	$\{B_3 \vee \neg A_4 \vee A_5\}$
c_6:	$\{\neg(y-z \leq 1) \vee (x+y=1) \vee \neg A_5\}$	c_6:	$\{\neg B_4 \vee B_5 \vee \neg A_5\}$
c_7:	$\{A_3 \vee \neg(x+y=0)\}$	c_7:	$\{A_3 \vee \neg B_6\}$
c_8:	$\{\neg A_3 \vee (z+y=2)\}$	c_8:	$\{\neg A_3 \vee B_7\}$
c_9:	$\{\neg(x+y=0) \vee \neg(x+y=1)\}$	c_9:	$\{\neg B_6 \vee \neg B_5\}$

$$\mu_1^p = \{B_1, A_3, \neg A_4, \neg A_5, \neg B_6, B_5, B_3, B_4, B_7\}$$
$$\mu_1 = \{(x-z>4), \neg(x+y=0), (x+y=1), (x-y \leq 3), (y-z \leq 1), (z+y=2)\}$$

Algorithm 6 WALKSMT (φ)

Require: $\langle \varphi, \text{MAX_TRIES}, \text{MAX_FLIPS} \rangle$

1: **for** $i = 1$ to MAX_TRIES **do**
2: $\mu^p \leftarrow$ initial_truth_assignment (φ^p)
3: **for** $j = 1$ to MAX_FLIPS **do**
4: **if** $(\mu^p \models \varphi^p)$ **then**
5: $\langle status, c^p \rangle \leftarrow \mathcal{T} -$ Solver(φ^p, μ^p)
6: **if** $(status ==$ SAT$)$ **then**
7: **return** SAT
8: **end if**
9: $\mu^p \leftarrow$ next_truth_assignment (φ^p, c^p)
10: **else**
11: $c^p \leftarrow$ choose_unsatisfied_clause (φ^p)
12: $\mu^p \leftarrow$ next_truth_assignment (φ^p, c^p)
13: **end if**
14: **end for**
15: **end for**
16: **return** UNKNOWN

7.2 A basic WalkSMT procedure

The observation in §7.1 suggested us a procedure integrating a $\mathcal{T}$-Solver into a SLS algorithm of the WalkSAT family (WALKSMT hereafter). A high-level description of the pseudo-code of WALKSMT is shown in Algorithm 6. It receives in input a $\mathcal{T}$-formula in CNF and applies a WalkSAT scheme (see §2.1.2) to its Boolean abstraction φ^p. (Notice that their underlying heuristics vary with the different variants of WalkSAT adopted.) The only significant difference w.r.t. the schema of WalkSAT is in lines 4-9. Whenever a total truth assignment μ^p is found s.t. $\mu^p \models \varphi^p$, it is passed to $\mathcal{T}$-Solver.

- If the set of $\mathcal{T}$-literals corresponding to μ^p is $\mathcal{T}$-satisfiable (i.e., $\mu^p \models \varphi^p \wedge \tau^p$), the procedures ends returning SAT (line 9).

- Otherwise, $\mathcal{T}$-Solver returns CONFLICT and a $\mathcal{T}$-lemma c^p which is used

by next_truth_assignment as "selected" unsatisfied clause for driving the flipping of the variable (line 7).

Notice that the last case corresponds to say that $\mu^p \not\models \varphi^p \wedge \tau^p$, and that c^p is one of the (possibly-many) clauses in $\varphi^p \wedge \tau^p$ which are falsified by μ^p. Consequently, $\mathcal{T}$-Solver plays the role of choose_unsatisfied_clause on $\varphi^p \wedge \tau^p$ when no unsatisfied clause is found in φ^p (see also "Multiple Learning" in §7.4).

Example 7.2. Suppose WALKSMT is invoked on the formula φ^p in Example 7.1, generating the total truth assignment μ_1^p that satisfies φ^p. Then $\mathcal{T}$-Solver is invoked on μ_1, which is $\mathcal{T}$-inconsistent due to the the literals $\{(x - z > 4), (x + y = 1), (z + y = 2)\}$, returning UNSAT and the conflict clause $c_1^p = \{\neg B_1 \vee \neg B_5 \vee \neg B_7\}$ (i.e. c_{12} in τ^p). Then next_truth_assignment will flip one of the literals B_1, B_5 or B_7.

7.3 Efficient $\mathcal{T}$-solvers for local search.

In CDCL-based SMT solvers, the interaction with $\mathcal{T}$-Solver$s$ is *stack-based*: the truth assignment μ is incrementally extended when performing unit propagation, $\mathcal{T}$-propagation, and when picking an unassigned literal for branching, and it is partly undone upon backtracking, when the most-recently-assigned literals are removed from it. Consequently, $\mathcal{T}$-Solver$s$ designed for interaction with a CDCL SAT solver are typically optimized for such stack-based invocation; in particular, they are typically *incremental* and *backtrackable* (see §2.2.2 for details).

In local search, instead, a new assignment μ' is obtained from the previous one μ by flipping *an arbitrary* literal (according to some heuristics). In this setting, the conventional backtrackability feature of $\mathcal{T}$-Solver$s$ is of little use, since there is no notion of most-recently-assigned literals to remove. Instead, it is very desirable to be able to remove *arbitrary* literals from a $\mathcal{T}$-Solver without

the need of resetting its internal state. Such requirement might seem unrealistic, or at least difficult to fulfill. However, at least two state-of-the-art $\mathcal{T}$-Solver$s$ have this capability: the $\mathcal{T}$-Solver for $\mathcal{DL}$ of [36] and the $\mathcal{T}$-Solver for $\mathcal{LA}(\mathbb{Q})$ of [41], which are therefore natural candidates for integration with a SLS-based SAT solver. The MATHSAT solver implements both.

7.4 Enhancements to the basic WalkSMT procedure

The basic WALKSMT procedure in Algorithm §6 is very naive (and very inefficient, see experimental evaluation in §8). In what follows we analyze the interaction of a $\mathcal{T}$-solver with a SLS SAT solver, and we present a group of enhanced techniques aimed at improving the synergy of their interaction. Algorithm 7 shows the basic WALKSMT procedure augmented with proposed enhancements.

7.4.1 Preprocessing

Before starting the search process, we apply a *preprocessing* step to the input formula φ in order to make it simpler to solve (lines 1-3 in Algorithm 7). This preprocessing consists mainly of two techniques: *Initial BCP* and *Static Learning*.

Initial BCP Often SMT formulas contain lots of “structural” atomic propositions whose truth value is assigned deterministically (e.g., when the formula derives from a CNF-ization step). Unlike a CDCL solver, an SLS one cannot handle them efficiently. Thus, during preprocessing we first perform a run of BCP (see §2.1.1 for details) to the input formula, simplifying the formula accordingly. In order to preserve correctness, we keep as unit clauses the $\mathcal{T}$-literals $l_1, .., l_n$ which have been assigned to true by BCP. If during this process one of the clauses of ϕ^p is falsified, or if the set

Algorithm 7 WALKSMT (φ)

```
Require: ⟨φ, MAX_TRIES, MAX_FLIPS⟩
 1: if (T-PREPROCESS (φ) == CONFLICT) then
 2:     return UNSAT
 3: end if
 4: for i = 1 to MAX_TRIES do
 5:     μ^p ← initial_truth_assignment (φ^p)
 6:     for j = 1 to MAX_FLIPS do
 7:         if (μ^p ⊨ φ^p) then
 8:             ⟨status, c^p⟩ ← T − Solver(φ^p, μ^p)
 9:             if (status == SAT) then
10:                 return SAT
11:             end if
12:             c^p ← UNIT-SIMPLIFICATION(φ^p, c^p)
13:             φ^p ← φ^p ∧ c^p
14:             μ^p ← next_truth_assignment (φ^p, c^p)
15:         else
16:             c^p ← choose_unsatisfied_clause (φ^p)
17:             μ^p ← next_truth_assignment (φ^p, c^p)
18:         end if
19:     end for
20: end for
21: return UNKNOWN
```

of $\mathcal{T}$-literals $l_1, .., l_n$ above is $\mathcal{T}$-inconsistent, the algorithm can exit returning UNSAT. Otherwise, $l_1, .., l_n$ are tagged "unflippable", so that the SLS engine initially assigns them to true and never flips their value.

Static Learning During preprocessing we also conjoin to the formula φ/φ^p short and "obvious" $\mathcal{T}$-lemmas on the atoms occurring in φ, which can be generated without explicitly invoking the $\mathcal{T}$-solver. Some examples of such $\mathcal{T}$-lemmas are shown in Example 7.1: mutual-exclusion lemmas like c_9 and lemmas encoding the transitivity of $\leq, <, \geq, >$, like c_{10} and c_{11} (see also [79]). (Notice that if previously unit-propagated $\mathcal{T}$-atoms occur

in such $\mathcal{T}$-lemmas, then the $\mathcal{T}$-lemmas are simplified accordingly.) The $\mathcal{T}$-Solver is invoked on an assignment μ only if μ^p verifies also these $\mathcal{T}$-lemmas (line 7 in Algorithm 6). This prevents WALKSMT from invoking $\mathcal{T}$-Solver on obviously-$\mathcal{T}$-inconsistent assignments.

Example 7.3. Consider the formulas ϕ and φ of Example 7.1, the preprocessing step generates φ from ϕ. In fact, BCP unit-propagates the literals $A_1, B_1, \neg A_2$, simplifying clause c_7 and eliminating clauses c_1, c_3 and c_4. Clause c_2 survives as an unit clause because B_1 is (the label of) a $\mathcal{T}$-literal. Notice that the $\mathcal{T}$-atom $B_2 \stackrel{\text{def}}{=} (y \geq 0)$ disappears from the formula because c_3 is satisfied by the unit-propagation of A_1. The $\mathcal{T}$-lemma c_9 is then added to the simplified formula by static learning. Notice that c_{10} and c_{11} are not added to φ^p because they are subsumed in φ^p by the unit clause c_2.

7.4.2 Single and Multiple Learning

Inspired by CDCL-based SMT solvers, we integrate the learning technique within WALKSMT and present the following techniques: *Learning*, *Unit Resolution* and *Multiple Learning*.

Learning SLS SAT solvers typically do not implement learning. This is potentially a major problem with SLS-based SMT, because the SLS solver may generate many total assignments $\mu_1^p, ..., \mu_k^p$ each containing the same $\mathcal{T}$-inconsistent subset η^p, causing thus $k-1$ useless calls to $\mathcal{T}$-Solver. Thus, like in standard CDCL-based SMT solvers, we conjoin to φ^p the $\mathcal{T}$-lemma c^p returned by the $\mathcal{T}$-solver (line 13 in Algorithm 7). Henceforth $\mathcal{T}$-Solver is no more invoked on assignments violating c^p.

Unit Resolution Before learning a $\mathcal{T}$-lemma c, we remove from it all the $\mathcal{T}$-literals whose negation occurs as unit clauses in the input problem (line 12 in Algorithm 7). (Notice that after this step c may be no longer a $\mathcal{T}$-lemma.) We do this in both static and dynamic learning.

Example 7.4. Consider the scenario of Example 7.2, assuming learning is implemented. Because of the unit clause c_2 of φ^p, we remove from the conflict clause c_1^p the literal $\neg B_1$, obtaining $c_1^{p\prime} \stackrel{\text{def}}{=} \{\neg B_5 \vee \neg B_7\}$ (i.e., a unit-resolved version of c_{12} in τ^p.), which we add to φ^p. Then next_truth_assignment will flip one of the literals B_5 or B_7. $\mathcal{T}$-Solver will never be invoked again on assignments containing both B_5 and B_7.

Multiple Learning Unlike with CDCL-based SMT solvers, which typically use some form of early pruning to check partial truth assignments for $\mathcal{T}$-consistency, in an SLS-based approach $\mathcal{T}$-solvers operate always on *complete* truth assignments μ. In this setting, it is likely that μ contains many different $\mathcal{T}$-inconsistent subsets, often independent from each another. This is the idea at the basis of our *multiple learning* technique, which allows for learning more than one $\mathcal{T}$-lemma for every $\mathcal{T}$-inconsistent assignment. When a conflict set η is found (and simplified via unit-resolution), a given percentage p of its literals are randomly removed from μ, and $\mathcal{T}$-Solver is invoked again on the resulting set. This process is repeated until no more conflict is found. We then learn all the $\mathcal{T}$-lemmas $c_1^p, ..., c_k^p$ generated during the process. Also, if $k > 1$, then one clause c^p among $c_1^p, ..., c_k^p$ is chosen by choose_unsatisfied_clause to be fed to next_truth_assignment.

Example 7.5. Consider the scenario of Example 7.2 and 7.4, assuming multiple learning is implemented, with $p = 100\%$. After learning the clause $c_1^{p\prime}$, we drop B_5, B_7 from μ_1^p and re-invoke $\mathcal{T}$-Solver on the set of $\mathcal{T}$-literals $\mu_2 \stackrel{\text{def}}{=} \mu_1 \setminus \{(x + y = 1), (z + y = 2)\}$, returning UNSAT and the conflict clause $c_2^p \stackrel{\text{def}}{=} \{\neg B_1 \vee \neg B_3 \vee \neg B_4\}$, from which $\neg B_1$ is removed by unit-resolution, so that also the clause $c_2^{p\prime} \stackrel{\text{def}}{=} \{\neg B_3 \vee \neg B_4\}$ is learned (a unit-resolved version of clause c_{10}). After further removing B_3 and B_4 from μ_2 the set of $\mathcal{T}$-literals is found $\mathcal{T}$-consistent by $\mathcal{T}$-Solver, so that no further clause is learned. Then $c_1^{p\prime}, c_2^{p\prime}$ are fed to choose_unsatisfied_clause which selects one and feed it

to next_truth_assignment, which flips one literal among B_5, B_7, B_3 and B_4.

7.4.3 Filterings

Pure-literal Filtering If some $\mathcal{T}$-atoms occur only positively [resp. negatively] in the original formula (learned clauses and statically-learned clauses are not considered), then we can safely drop every negative [resp. positive] occurrence of them from the assignment μ to be checked by the $\mathcal{T}$-solver (see §2.2.3 for details). (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 is twofold:

(i) reduces the workload for the $\mathcal{T}$-Solver by feeding it smaller sets;

(ii) increases the chance of finding a $\mathcal{T}$-consistent satisfying assignment by removing "useless" $\mathcal{T}$-literals which may cause the $\mathcal{T}$-inconsistency of μ.

Example 7.6. Consider the formula φ^p in Example 7.1 and the total truth assignment

$$\mu_4^p = \{B_1, \neg A_3, \neg A_4, \neg A_5, \neg B_6, \neg B_5, B_3, B_4, \neg B_7\}$$

that satisfies φ^p, but is $\mathcal{T}$-inconsistent because of its subset $\{B_1, B_3, B_4\}$ (clause c_{10} in τ^p). Without pure-literal filtering, $\mathcal{T}$-Solver detects the inconsistency, WALKSMT learns the clause and looks for another assignment. If pure-literal filtering is implemented, instead, since the $\mathcal{T}$-literals $\neg B_5$, B_4 and $\neg B_7$ occur only negatively in the original formula ϕ, they are filtered out from μ_4^p, resulting in the *partial* assignment

$$\eta_4^p = \{B_1, \neg A_3, \neg A_4, \neg A_5, \neg B_6, B_3\},$$

which still satisfies φ^p. $\mathcal{T}$-Solver is invoked on the corresponding set of $\mathcal{T}$-literals:

$$\eta_4 = \{(x - z > 4), \neg(x + y = 0), (x - y \leq 3)\}.$$

which is $\mathcal{T}$-consistent, from which we can conclude that φ (and ϕ) is $\mathcal{T}$-consistent.

Ghost-literal Filtering We further enforce the benefits of pure-literal filtering as follows. When a truth assignment μ is found s.t. $\mu^p \models \varphi^p$, before invoking $\mathcal{T}$-Solver on μ, we check whether any $\mathcal{T}$-atom occurring only positively [resp. negatively] in the original formula and being assigned true [resp. false] in μ can be flipped without falsifying any clause. (This test can be performed very efficiently inside an SLS solver.) If this is the case, then the atom is flipped. This step is repeated until no more such atoms are found, after which the resulting set μ is passed to $\mathcal{T}$-Solver. This allows for further removing useless $\mathcal{T}$-literals from μ by pure-literal filtering. (Since such literals are a particular case of "ghost literals" [79], we call this enhancement *ghost-literal filtering*.)

Example 7.7. Consider the formula φ^p in Example 7.1 and the total truth assignment

$$\mu_5^p = \{B_1, A_3, \neg A_4, \neg A_5, \neg B_6, \neg B_5, B_3, \neg B_4, B_7\}$$

that satisfies φ^p. If we apply pure-literal filtering on μ_5^p, then we can filter out only the literal $\neg B_5$ before invoking $\mathcal{T}$-Solver. By ghost-literal filtering, the literals B_3, $\neg B_4$ and $\neg B_6$ are flipped without falsifying φ^p, resulting in the total truth assignment:

$$\mu_5^{p\prime} = \{B_1, A_3, \neg A_4, \neg A_5, B_6, \neg B_5, \neg B_3, B_4, B_7\}.$$

Now, by pure-literal filtering, we remove from $\mu_5^{p\prime}$ the literals B_3, $\neg B_4$, $\neg B_5$ and $\neg B_6$.

Chapter 8

Experimental evaluation for WalkSMT

In this chapter we propose an implementation of WALKSMT with the techniques described in §7.4, which is based on the integration of the UBCSAT [89] and UBCSAT++ [23] SLS solvers with the $\mathcal{LA}(\mathbb{Q})$-solver of MATHSAT4 [30]. We present an extensive experimental evaluation of our implementation by considering satisfiable industrial problems coming from the SMT-LIB, and we evaluate the effects of the various optimization techniques, also comparing them against MATHSAT4. We also compare WALKSMT and MATHSAT on randomly-generated unstructured problems, obtaining small differences in performances.

This chapter is divided in three sections: §8.1 describes tools, configurations and environment used in our experimental evaluation and §8.2 and §8.3 present the results for SMT-LIB benchmarks and random generated formulas, respectively.

8.1 Environment and Settings

We have implemented two versions of the WALKSMT procedure described in §7 to work for the $\mathcal{LA}(\mathbb{Q})$ theory. The implementation is done on top of MATHSAT4 [30], using part of its preprocessor its $\mathcal{LA}(\mathbb{Q})$-solver [41] and lots of its features. We have implemented two versions, each using one between

two SLS-based SAT solvers: UBCSAT [1] [89] and UBCSAT++ [2] [23]. UBCSAT is a SLS platform providing a very-wide range of SLS algorithms for SAT (including the WalkSAT family), with a very flexible architecture which made the integration of the $\mathcal{LA}(\mathbb{Q})$-solver of MATHSAT4 relatively easy. Among the various SLS procedures provided by UBCSAT, we have chosen to use the Adaptive Novelty$^+$ variant of the WalkSAT family because it was the best-performing in a previous extensive empirical evaluation [87]. UBCSAT++ is built on top of UBCSAT and extends its implementation of Adaptive Novelty$^+$ with the Trimming Variable Selection and Literal Commitment Strategy techniques described in §2.1.2. We partition the enhancements of WALKSMT of §7.4 into three groups:

- *Preprocessing and Learning (PL)*, including preprocessing (Initial BCP and Static Learning), Learning and Unit Resolution;
- *Multiple Learning (ML)*;
- *Filtering (FI)*, including both Pure-Literal and Ghost-Literal filterings.

Notationally, we use a "+" [resp. "–"] symbol to denote that an option is enabled [resp. disabled]: e.g., "UBCSAT++ BASIC+PL-ML+FI" denotes WALKSMT based on UBCSAT++ with PL and FI enabled and ML disabled. (Notice that ML requires PL, so that we cannot have "...-PL+ML..." configurations.)

In this section, we evaluate the performance of WALKSMT by comparing its two versions (those based on UBCSAT and UBCSAT++ respectively) against the CDCL-based SMT solver MATHSAT4. We ran MATHSAT4 with all the optimizations enabled (the most important ones are early pruning and $\mathcal{T}$-propagation). [3] We performed our comparison over two distinct sets of in-

[1] UBCSAT was developed by Tompkins and Hoos and is publicly available at `http://www.satlib.org/ubcsat/`.

[2] UBCSAT++ was kindly provided to us by the developers, Belov and Stachniak.

[3] Although more efficient SMT ($\mathcal{LA}(\mathbb{Q})$) solvers exist, including the recent MATHSAT5, here the choice of MATHSAT4 is aimed at minimizing the differences in performance due to the implementation, because

Solver	SMT-LIB Instances						Total
	sc	uart	sal	TM	tta	miplib	
Total # of Instances	108	36	11	24	24	22	225
WalkSMT UBCSAT Basic–PL–ML–FI	0	0	0	0	0	0	0
WalkSMT UBCSAT++ Basic–PL–ML–FI	0	0	0	0	0	1	1
WalkSMT UBCSAT Basic+PL–ML–FI	59	10	6	13	5	3	96
WalkSMT UBCSAT++ Basic+PL–ML–FI	46	6	7	17	10	1	87
WalkSMT UBCSAT Basic+PL+ML–FI	103	15	6	12	6	3	145
WalkSMT UBCSAT++ Basic+PL+ML–FI	61	6	7	15	9	1	99
WalkSMT UBCSAT Basic+PL–ML+FI	59	32	10	14	9	3	127
WalkSMT UBCSAT++ Basic+PL–ML+FI	62	12	8	18	10	1	111
WalkSMT UBCSAT Basic+PL+ML+FI	78	35	10	14	9	3	149
WalkSMT UBCSAT++ Basic+PL+ML+FI	63	14	8	19	10	2	116
MATHSAT4	108	36	11	21	24	8	208

Figure 8.1: Comparison of the number of instances solved within the 600s timeout by the various configurations of WALKSMT and MATHSAT4.

stances, which are described in the next two sections: the first consists of the set of all satisfiable $\mathcal{LA}(\mathbb{Q})$ formulas in the SMT-LIB 1.2 (`www.smtlib.org`), whereas the second is composed of randomly-generated problems. All tests were executed on 2.66 GHz Xeon machines running Linux, using a timeout of 600 seconds. The correctness of the models found by WALKSMT have been cross-checked by MATHSAT4. In order to make the experiments reproducible, the full-size plots, the tools, the problems, and the results are available [4].

8.2 WALKSMT on SMT-LIB Instances

In the first part of our experiments, we compare WALKSMT against MATHSAT on all the satisfiable $\mathcal{LA}(\mathbb{Q})$-formulas (QF_LRA) in the SMT-LIB 1.2.

WALKSMT is implemented on top of MATHSAT4 (in particular it uses its preprocessor and $\mathcal{T}$-solver for $\mathcal{LA}(\mathbb{Q})$), so that to better highlight the differences between SLS- and CDCL-based approaches.

[4] `http://disi.unitn.it/~rseba/frocos11/tests.tar.gz`

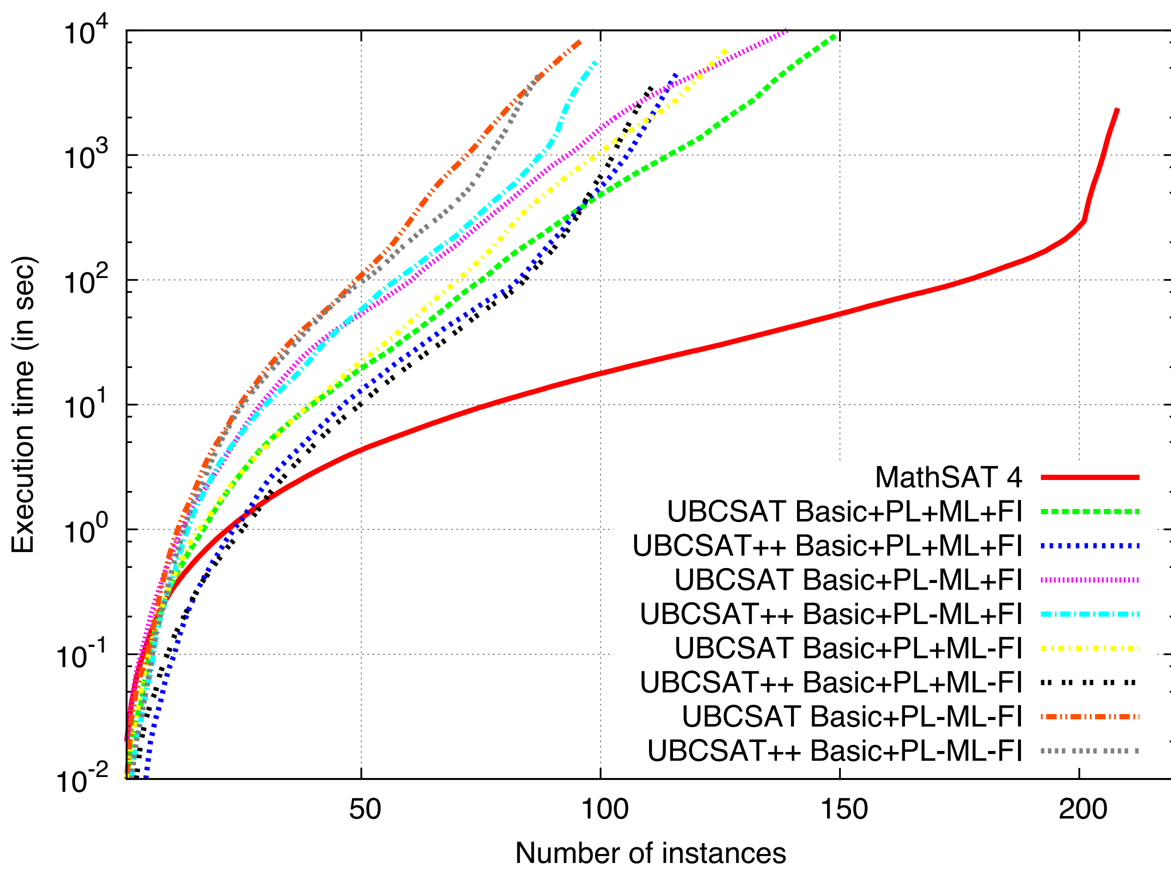

Figure 8.2: Cumulative plots of WALKSMT and MATHSAT4 on all SMT-LIB instances.

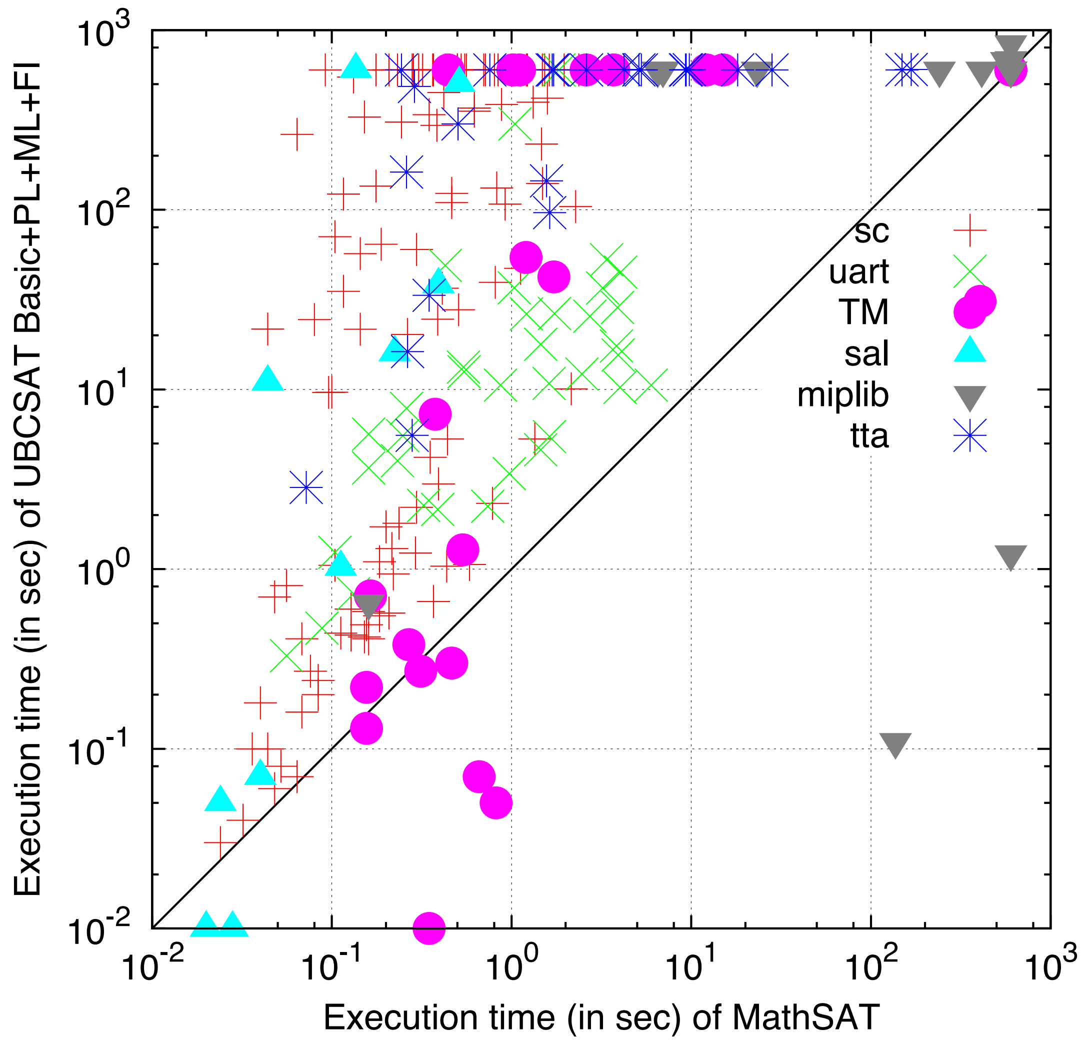

Figure 8.3: Comparison of the best configurations of WALKSMT with UBCSAT (BASIC+PL+ML+FI) against MATHSAT4 on SMT-LIB instances.

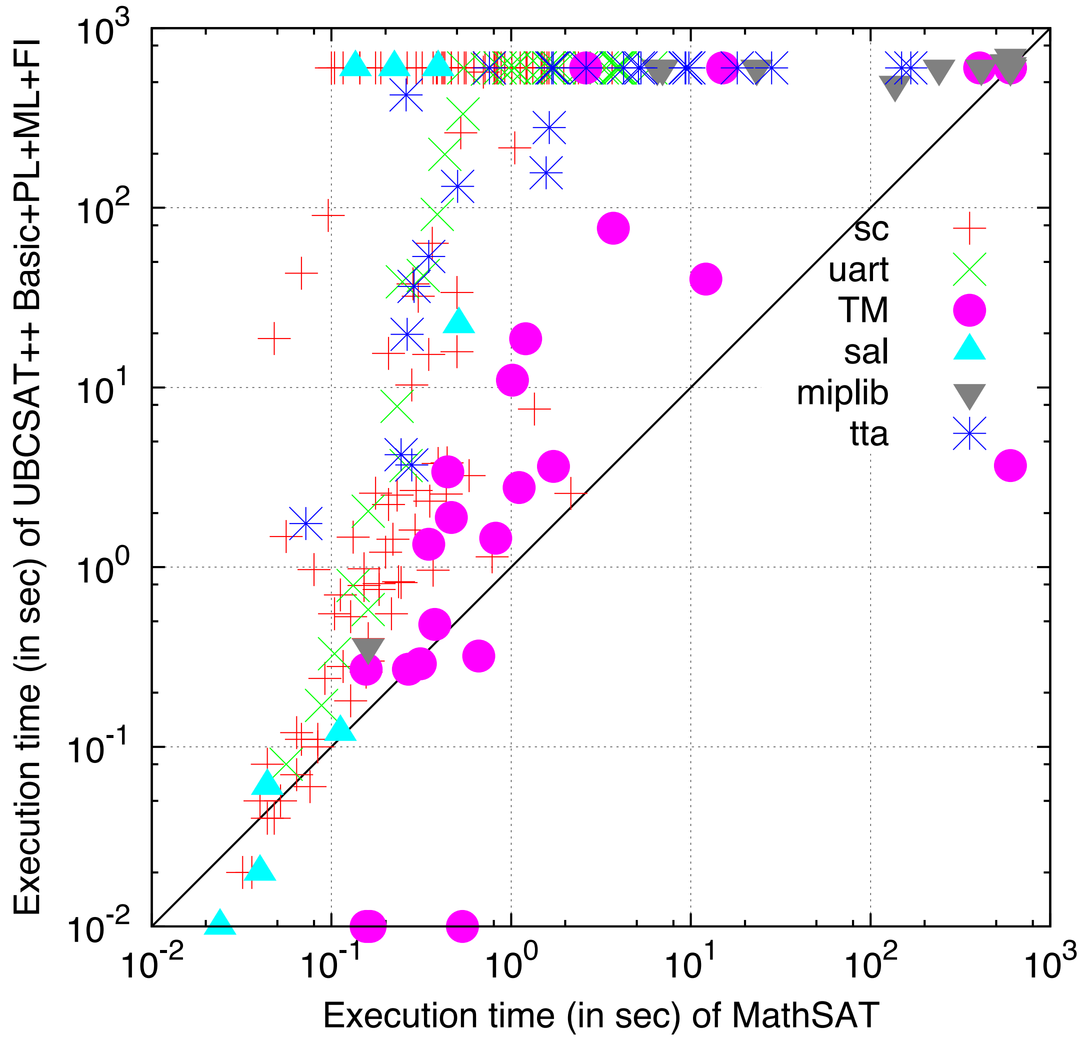

Figure 8.4: Comparison of the best configurations of WALKSMT with UBCSAT++ (i.e. BASIC+PL+ML+FI) against MATHSAT4 on SMT-LIB instances.

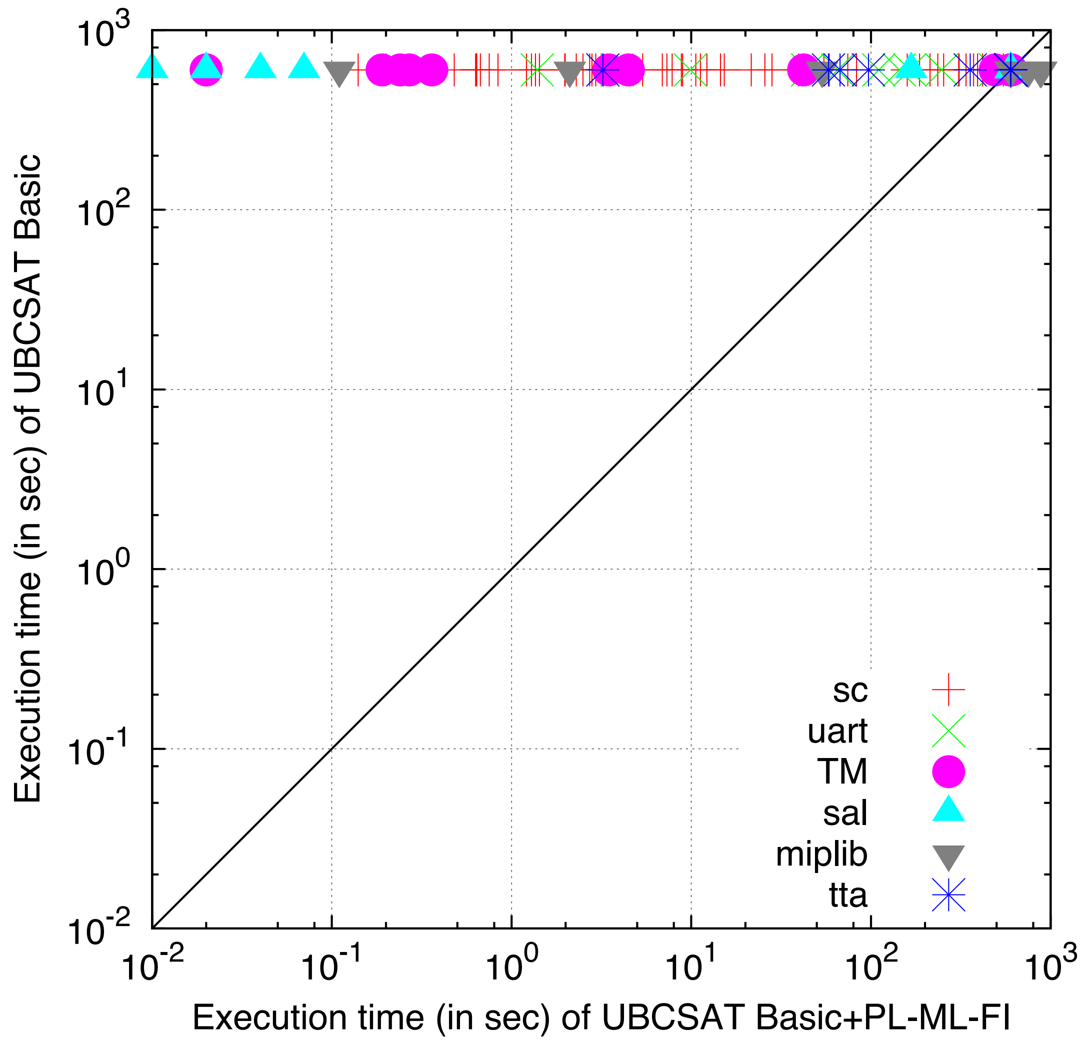

Figure 8.5: Comparison of configurations of WALKSMT with UBCSAT: BASIC-PL-ML-FI vs. BASIC+PL-ML-FI (benefits of adding PL to Basic).

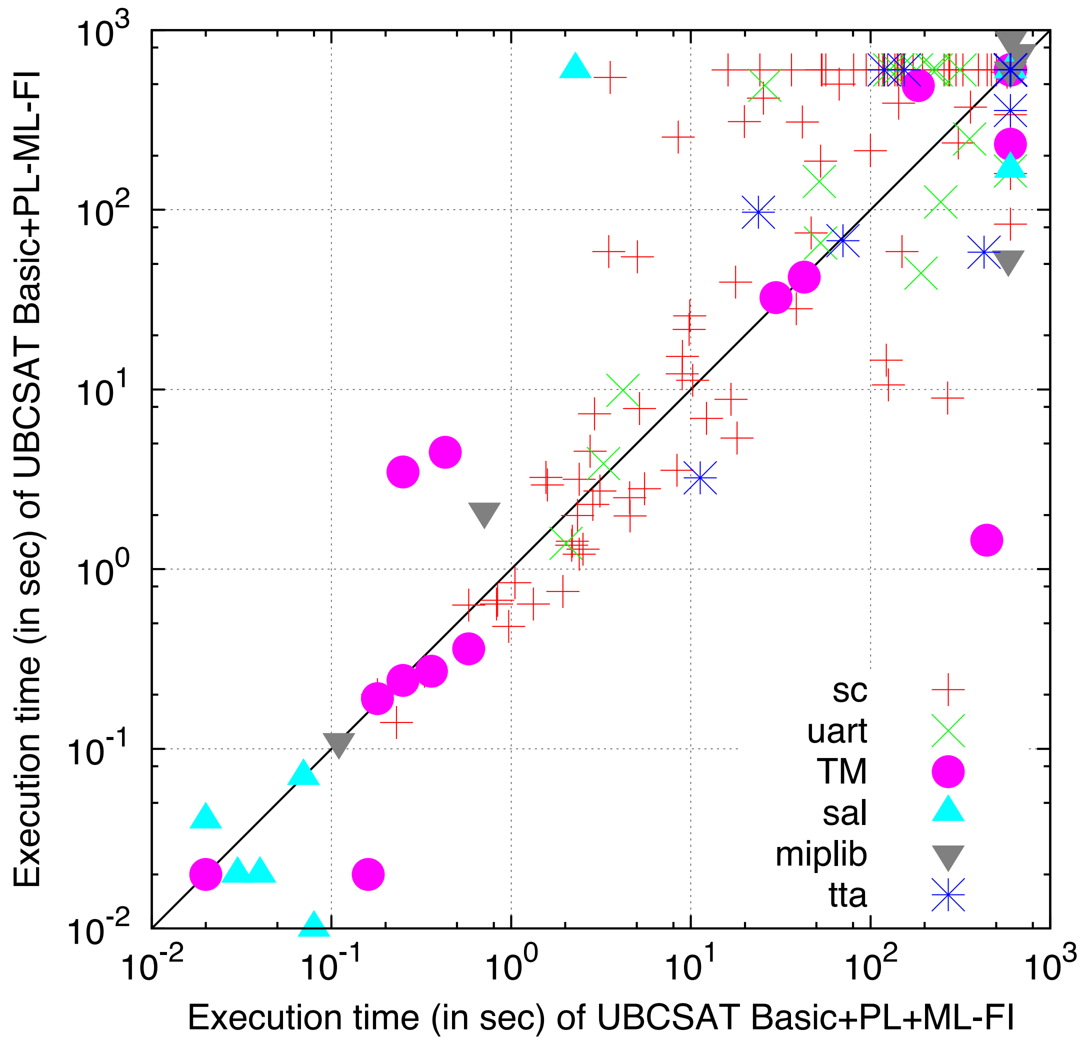

Figure 8.6: Comparison of configurations of WALKSMT with UBCSAT: BASIC+PL-ML-FI vs. BASIC+PL+ML-FI (benefits of further adding ML).

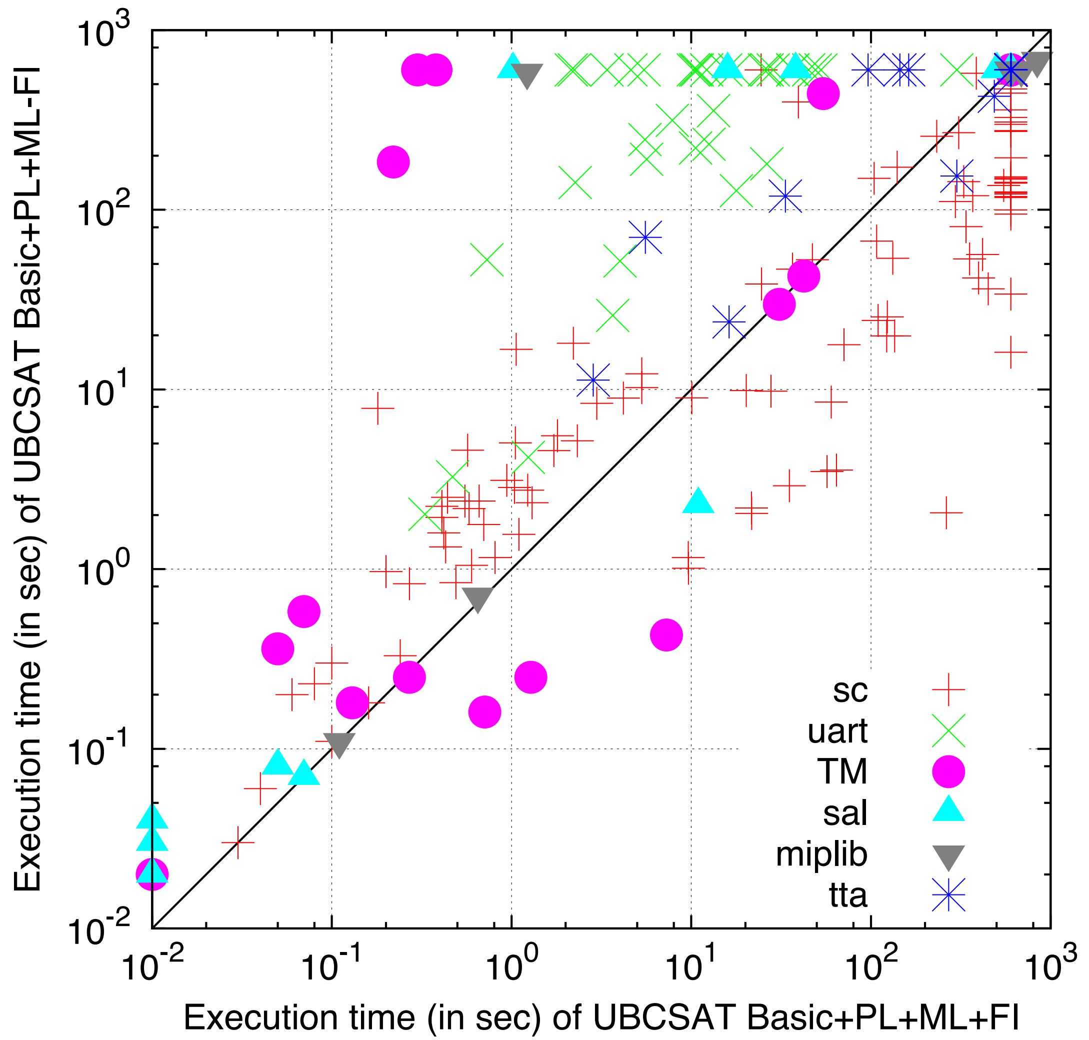

Figure 8.7: Comparison of configurations of WALKSMT with UBCSAT: BASIC+PL+ML-FI vs. BASIC+PL+ML+FI (benefits of further adding FI).

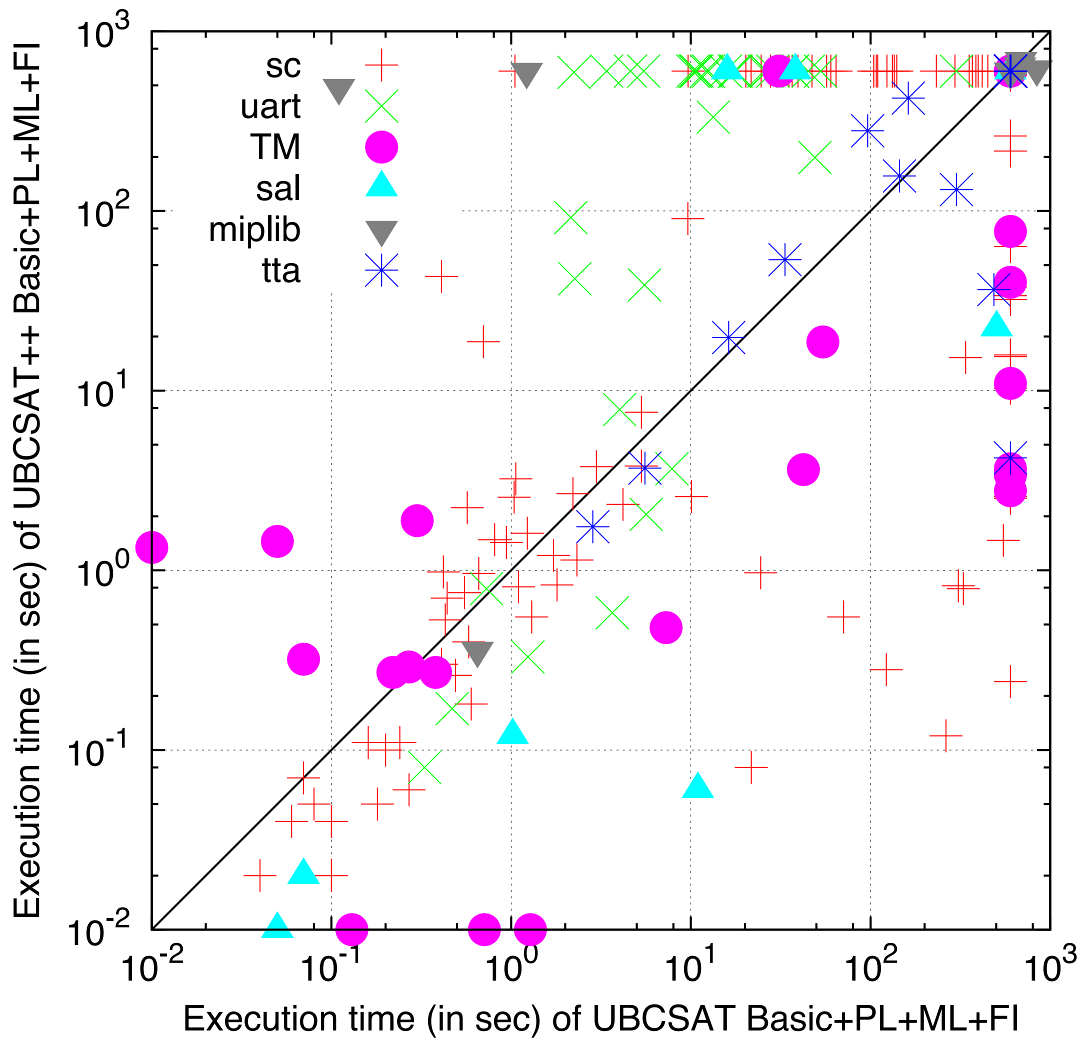

Figure 8.8: Comparison between the CPU time of WALKSMT UBCSAT and WALKSMT UBCSAT++ on BASIC+PL+ML+FI versions.

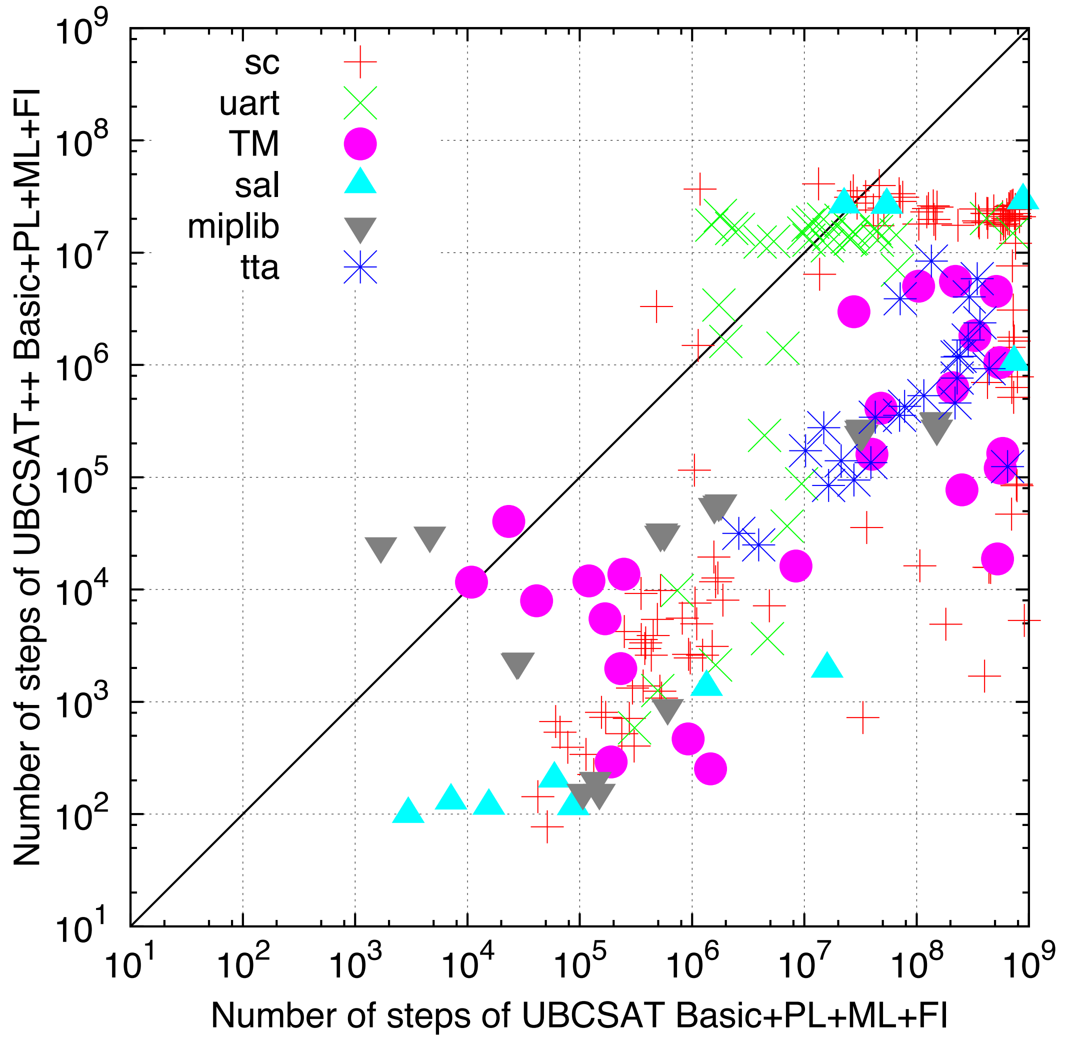

Figure 8.9: Comparison between the number of flips (flips#) of WALKSMT UBCSAT and WALKSMT UBCSAT++ on BASIC+PL+ML+FI versions (on commonly solved instances).

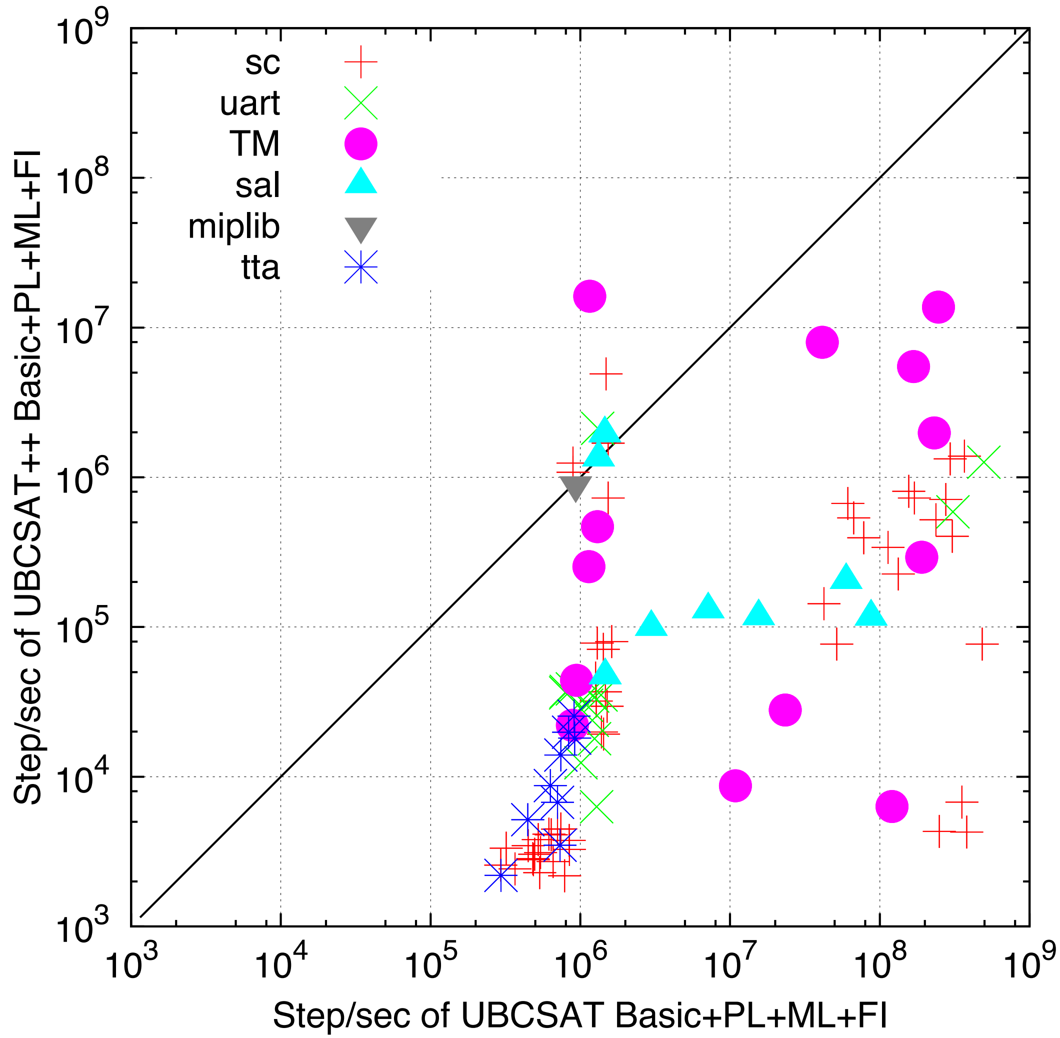

Figure 8.10: Comparison between the average ratio flips#/sec of WALKSMT UBCSAT and WALKSMT UBCSAT++ on BASIC+PL+ML+FI versions (on commonly solved instances).

These instances are all classified as "industrial", because they come from the encoding of different real-world problems in formal verification, planning and optimization, and they are divided into six categories: `sc`, `uart`, `sal`, `TM`, `tta_startup` ("`tta`" hereafter), and `miplib`. [5] Since SLS has a "random" component, we have run WALKSMT 5 times on each benchmark with different seeds and then we have taken the median value. The results of the experiments are reported in Figures 8.1-8.10. Figure 8.1 shows the number of instances solved by all tools. Figure 8.2 shows the cumulative plots of the execution time for the different configurations of WALKSMT and MATHSAT4 on SMT-LIB instances. (The plots for BASIC-PL-ML-FI are not reported since no formula was solved within the timeout.) Figures 8.3-8.4 compare the best configurations of WALKSMT (BASIC+PL+ML+FI) with UBCSAT and with UBCSAT++, respectively, against MATHSAT4 on all instances. Figures 8.5-8.7 show the relative effects of the different optimizations for WALKSMT with UBCSAT. Figures 8.8-8.10 compare WALKSMT UBCSAT against WALKSMT UBCSAT++ on BASIC+PL+ML+FI versions. The results suggest a list of considerations.

First, the optimizations described in §7.4 lead to dramatic improvements in performance, sometimes by orders of magnitude. Without them, WALKSMT times out on all instances (see Figures 8.1-8.2 and 8.5-8.7.):

- PL is crucial for performance, since with PL disabled almost no problem is solved within the timeout. In particular, from our data we see that a key role is played by learning. (Which perhaps is not surprising from an SMT perspective, but we believe may be of interest from an SLS perspective.)

- ML produces significant improvements overall, except for a few cases where it may worsen performances (e.g., with `miplib`).

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

- FI produces strong improvements in performance in all problem categories, (apparently with the exception of the `sc` benchmarks).

Second, globally WALKSMT seems to perform better with UBCSAT than with UBCSAT++, with some exceptions (`TM`, `tta`). From Figures 8.8-8.10, considering the problems solved by both configurations, we see that the total number of flips performed by UBCSAT++ is dramatically smaller than that performed by UBCSAT, but the average cost of each flip is dramatically higher.

Third, globally MATHSAT4 performs much better than WALKSMT, often by orders of magnitude. This mirrors the typical performance gap between CDCL and SLS SAT solvers on industrial benchmarks.

8.3 WALKSMT on Random Instances

Unlike with SAT, in SMT there is very-limited tradition in testing on random problems (e.g., [8, 9]). However, for a matter of scientific curiosity and/or to leverage to SMT a popular test for SLS SAT procedures, here we present also a brief comparison of WALKSMT vs. MATHSAT4 on randomly-generated, unstructured 3-CNF $\mathcal{LA}(\mathbb{Q})$-formulas. Each 3-CNF formula is randomly generated according to three integer parameters $\langle m, n, a\rangle$ as follows. First, a distinct $\mathcal{T}$-atoms $\psi_1, ..., \psi_a$ are created, s.t. each atom ψ_j is in the form $(\sum_{i=1}^{4} c_{ji}x_{ji} \leq c_j)$, it is generated by randomly picking four distinct variables x_{ji} out of n variables $\{x_1, ..., x_n\}$, and five integer values $c_{j1}, ..., c_{j4}, c_j$ in the interval $[-100, 100]$. Then, m 3-CNF clauses are randomly generated, each by randomly picking 3 distinct $\mathcal{T}$-atoms in $\{\psi_1, ..., \psi_a\}$, negating each with probability 0.5.

Figures 8.11-8.12 shows the run times of several versions of WALKSMT and MATHSAT4 on the generated formulas, for $n = 20$. Each graph shows curves for WALKSMT (in particular, UBCSAT and UBCSAT++ with the best configuration BASIC+PL+ML+FI) and MATHSAT4 on a group of instances with a

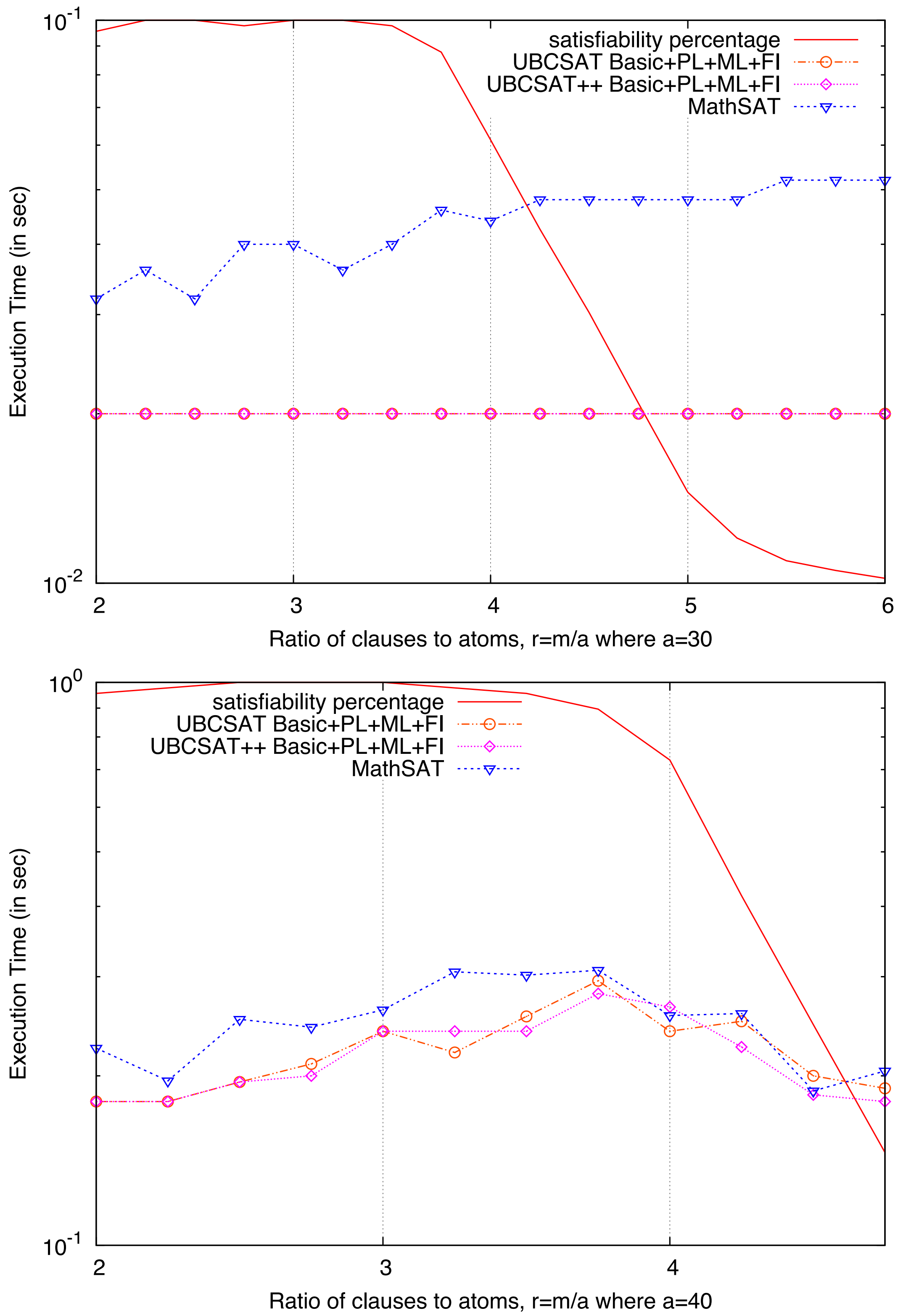

Figure 8.11: Comparison of different configurations of WALKSMT and MATHSAT4 on randomly-generated instances with 20 theory variables and atoms $a = 30, 40$.

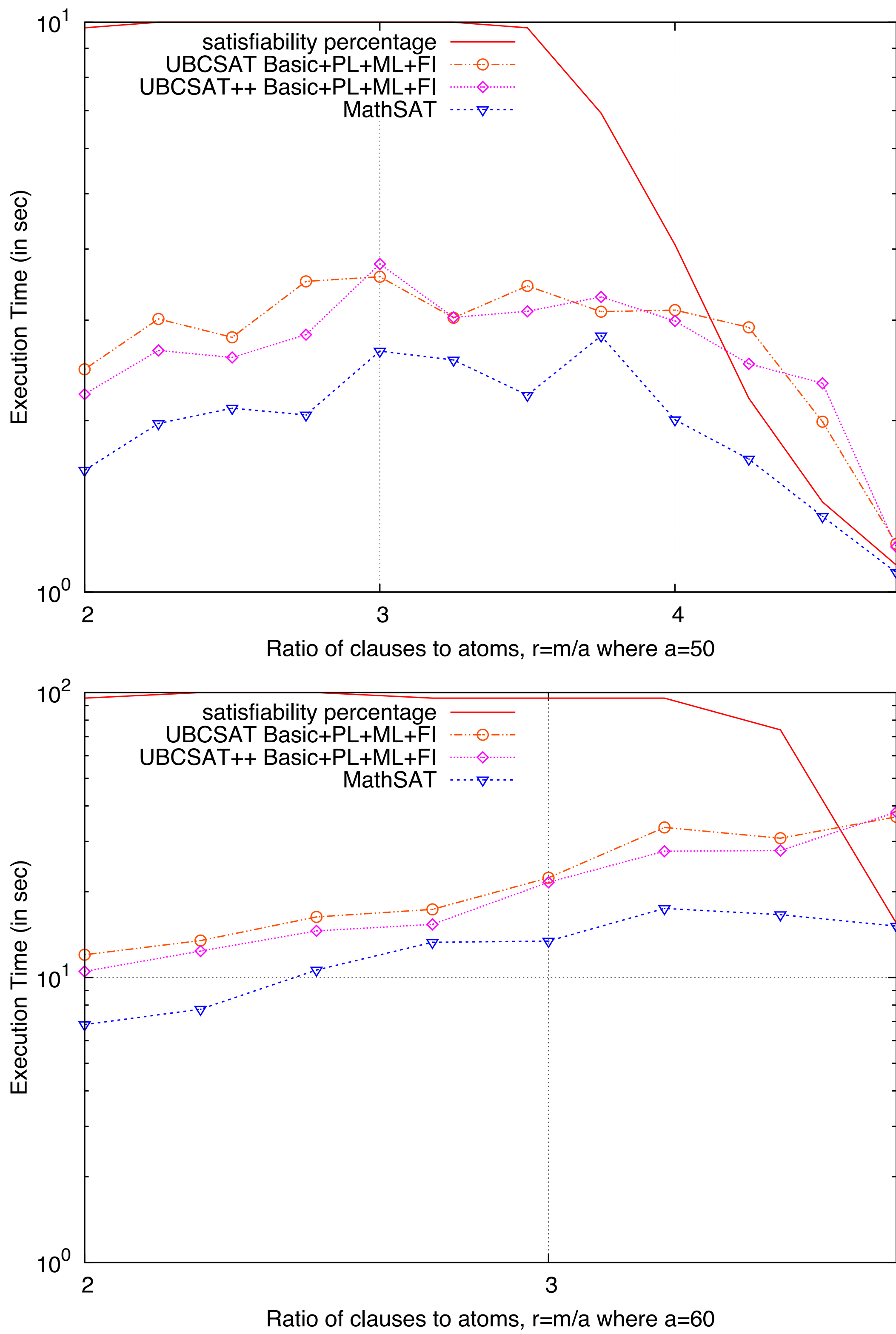

Figure 8.12: Comparison of different configurations of WALKSMT and MATHSAT4 on randomly-generated instances with 20 theory variables and atoms $a = 50, 60$.

fixed number a of $\mathcal{T}$-atoms, for $a = 30, 40, 50, 60$. The plots represent the execution time versus the ratio $r = m/a$ of clauses/$\mathcal{T}$-atoms. Each point in the graphs corresponds to the median run-time of each algorithm on 100 different instances of the same size. (For WALKSMT, each value is itself a median value of 3 runs with different seeds.) The plots show also the satisfiability percentage of each group of instances, defined as the ratio between the satisfiable instances generated and the total number of instances generated, for each value of r. E.g., in the plot in the first column of the first row of Figures 8.11-8.12 the percentage 0.01% for $r = 6$ means that we had to generate and test 10514 formulas (using MATHSAT4 with a timeout of 600 seconds) in order to obtain 100 satisfiable instances.

The results show that, unlike with SMT-LIB formulas, on randomly-generated instances there is a very small difference between the performance of UBCSAT BASIC+PL+ML+FI, UBCSAT++ BASIC+PL+ML+FI and MATHSAT4.

8.4 Discussion

Overall, we observe the following facts:

1. the basic "naive" version of WALKSMT was not able to solve any problem within the timeout;
2. the improved techniques drastically improve the performances of the basic version but WALKSMT cannot beat MATHSAT4.
3. WALKSMT performance are still very far from those of MATHSAT.

Chapter 9

Conclusions and Future Research Directions

In this thesis we have presented two main contributions.

First, 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 proposed are very-promising tools for a variety of optimization problems. This research opens the possibility for several interesting future directions:

1. The efficiency and the applicability of OPTIMATHSAT could be further improved by, e.g., extending the experimentation to novel sets of problems possibly investigating ad-hoc customization.

2. The OMT procedures can be extended to $\mathcal{LA}(\mathbb{Z})$ and mixed $\mathcal{LA}(\mathbb{Q}) \cup \mathcal{LA}(\mathbb{Z})$, by exploiting the solvers which are already present in MATHSAT [45] (Roberto Sebastiani e Patrick Trentin are currently working on this topic [90]).

3. It could be interesting to investigate the feasibility of extending the technique to deal with non-linear constraints, possibly using MINLP tools as

$\mathcal{T}$-Solver/Minimize.

Second, we have investigated the possibility of using an SLS SAT solver instead of a conventional CDCL-based one as propositional engine for a lazy SMT solver. We have presented and discussed several optimizations to the basic architecture proposed, which allowed WALKSMT to solve a significant amount of industrial SMT problems, although it is still much less efficient that the corresponding CDCL-based SMT solver. We believe that the latter fact is not surprising, since optimization techniques for CDCL-based SMT solvers have been investigated and optimized for the last ten years, whilst to the best of our knowledge this is the first attempt of building a SLS-based one. Initially, we have planned to investigate the use of SLS techniques for solving/approximating optimization problems (e.g. Max-SMT) but, considering the results, we decided to focusing on the lazy SMT paradigm. However, this work represents the foundations for SLS-based SMT solver and it constitutes a starting points for future researches. For example:

1. Exploration of the possibility of tightening the synergy between the SLS SAT solver and $\mathcal{T}$-solvers, for instance by better exploiting information that can be provided by $\mathcal{T}$-solvers when deciding which variables to flip, or by considering architectures in which the search is more driven by the theory part of the formula rather than by the SAT engine.

2. Improving the integration/combination between SLS-based and CDCL-based SMT both using a portfolio-like approach and investigating more tightly-coupled solutions.

3. Extending the presented work to cover other theories typically used in SMT (e.g., “hard” theories such as $\mathcal{LA}(\mathbb{Z})$).

Bibliography

[1] SMT-COMP 2012. http://smtcomp.sourceforge.net/2012/.

[2] Yices. http://yices.csl.sri.com/.

[3] Z3. http://research.microsoft.com/en-us/um/redmond/projects/z3/ml/z3.html.

[4] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: a new approach to integrate CP and MIP. In *Proceedings of the 5th international conference on Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, CPAIOR 2008*, LNCS, pages 6–20. Springer, 2008.

[5] T. Alsinet, F. Manya, and J. Planes. Improved branch and bound algorithms for max-sat and weighted max-sat. In *Sixth Catalan Conference on Artificial Intelligence*, 2003.

[6] C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *Symposium on Abstraction, Reformulation, and Approximation, SARA*, 2011.

[7] C. Ansótegui, M. L. Bonet, and J. Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[8] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proc. European Conference on Planning, CP-99*, 1999.

[9] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proceedings of The International Conference on Automated Deduction, CADE-18*, volume 2392 of *LNAI*. Springer, July 2002.

[10] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proc. of International Workshop on Bounded Model Checking, BMC 2004*, volume 119 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2005.

[11] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. of International Conference on Formal Techniques for Distributed Objects, Components and Systems, FORTE 2002*, volume 2529 of *LNCS*. Springer, November 2002.

[12] G. Audemard, J.-M. Lagniez, B. Mazure, and L. Sais. Boosting local search thanks to cdcl. In *Proc. of International Conference on Logic for Programming Artificial Intelligence and Reasoning, LPAR (Yogyakarta)*, pages 474–488, 2010.

[13] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3 – 44, 1998.

[14] E. Balas. Integer programming. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1617–1624. Springer US, 2009.

[15] E. Balas and P. Bonami. New variants of lift-and-project cut generation from the lp tableau: Open source implementation and testing. In *Proc. of Integer Programming and Combinatorial Optimization, IPCO*, pages 89–103, 2007.

[16] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

[17] C. Barrett and S. Berezin. Cvc lite: A new implementation of the cooperating validity checker. In R. Alur and D. Peled, editors, *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518. Springer Berlin Heidelberg, 2004.

[18] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. In Biere et al. [25], February 2009.

[19] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostak's method for combining decision procedures. In *Frontiers of Combining Systems (FROCOS)*, LNAI. Springer-Verlag, April 2002. Santa Margherita Ligure, Italy.

[20] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[21] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Max-Planck-Institut fur Informatik, Im Stadtwald, D-66123 Saarbrucken, Germany, January 1995.

[22] P. Barth. *Logic-based 0-1 constraint programming*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[23] A. Belov and Z. Stachniak. Improving variable selection process in stochastic local search for propositional satisfiability. In *Proc. of International Conference on Theory and Applications of Satisfiability Testing, SAT 2009*, LNCS. Springer, 2009.

[24] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 1999*, pages 193–207, 1999.

[25] 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.

[26] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[27] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10):1493–1525, 2006.

[28] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10):1493–1525, 2006.

[29] A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS - A User's Guide*. GAMS Development Corporation, Washington, DC, USA, 2011.

[30] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *Proc. of International Conference on Computer Aided Verification, CAV*, volume 5123 of *LNCS*. Springer, 2008.

[31] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.

[32] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A Modular Approach to MaxSAT Modulo Theories. In *International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 7962 of *LNCS*, July 2013.

[33] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT 5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13.*, volume 7795 of *LNCS*, pages 95–109. Springer, 2013.

[34] A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. In *Proc. of International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 4501 of *LNCS*, pages 334–339. Springer, 2007.

[35] A. Cimatti, A. Griggio, and R. Sebastiani. Computing Small Unsatisfiable Cores in SAT Modulo Theories. *Journal of Artificial Intelligence Research, JAIR*, 40:701–728, April 2011.

[36] S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In *Proc. of International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 4121 of *LNCS*. Springer, 2006.

[37] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13, pages 339–347. John Wiley & Sons Inc., New York, N. Y., 1951.

[38] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[39] L. de Moura and N. Bjørner. Model-based theory combination. *Electronic Notes in Theoretical Computer Science*, 198(2):37–49, May 2008.

[40] I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken. Minimum Satisfying Assignments for SMT. In *Proc. of Proc. of International Conference on Computer Aided Verification, CAV*, pages 394–409, 2012.

[41] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc of International Conference on Computer Aided Verification, CAV*, volume 4144 of *LNCS*, 2006.

[42] M. X. Goemans and D. P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[43] C. Gomes, W.-J. van Hoeve, and L. Leahu. The power of semidefinite programming relaxations for max-sat. In J. Beck and B. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin / Heidelberg, 2006.

[44] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[45] A. Griggio. A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT*, 8:1–27, 2012.

[46] A. Griggio, Q. S. Phan, R. Sebastiani, and S. Tomasi. Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT. In *Frontiers of Combining Systems, FroCoS'11*, LNAI. Springer, 2011.

[47] A. Griggio, R. Sebastiani, and S. Tomasi. Stochastic Local Search for SMT: a Preliminary Report. Proceedings SMT'09.

[48] F. Heras, J. Larrosa, and A. Oliveras. Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research (JAIR)*, 31:1–32, 2008.

[49] F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In W. Burgard and D. Roth, editors, *AAAI*. AAAI Press, 2011.

[50] M. J. H. Heule and H. van Maaren. *Look-Ahead Based SAT Solvers*, chapter 5, pages 155–184. In Biere et al. [25], February 2009.

[51] H. H. Hoos. An adaptive noise mechanism for walksat, 2002.

[52] H. H. Hoos and T. Stutzle. Local Search Algorithms for SAT: An Empirical Evaluation. *Journal of Automated Reasoning*, 24(4), 2000.

[53] H. H. Hoos and T. Stutzle. *Stochastic Local Search Foundation And Application*. Morgan Kaufmann, 2005.

[54] IBM. *IBM ILOG CPLEX Optimizer*, 2010. Available at `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

[55] H. Jaygarl, S. Kim, T. Xie, and C. K. Chang. Ocat: Object capture-based automated testing. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*, ISSTA '10, pages 159–170, New York, NY, USA, 2010. ACM.

[56] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[57] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960.

[58] E. L. Lawler and D. W. Wood. Branch and bound methods: A survey. *Operations Research*, pages 699–719, 1966.

[59] C. M. Li and F. Manyà. *MaxSAT, Hard and Soft Constraints*, chapter 19, pages 613–631. In Biere et al. [25], February 2009.

[60] Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *Symposium on Principles of Programming Languages, POPL*, 2014. To appear.

[61] A. Lodi. Mixed Integer Programming Computation. In *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer-Verlag, 2009.

[62] I. Lynce and J. Marques-Silva. On Computing Minimum Unsatisfiable Cores. In *7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[63] P. Manolios and V. Papavasileiou. Ilp modulo theories. In *Proc. of International Conference on Computer Aided Verification, CAV*, pages 662–677, 2013.

[64] J. P. Marques-Silva. On computing minimum size prime implicants. In *in International Workshop on Logic Synthesis*, 1997.

[65] J. P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. In Biere et al. [25], February 2009.

[66] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58(1), 1992.

[67] A. Morgado, F. Heras, and J. Marques-Silva. Improvements to core-guided binary search for maxsat. In *Proc. of International Conference on Theory and Applications of Satisfiability Testing, SAT*, pages 284–297, 2012.

[68] C. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems, TOPLAS*, 1(2):245–257, 1979.

[69] G. Nelson and D. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.

[70] R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006.

[71] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[72] D. C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12:291–302, 1980.

[73] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, Feb. 1991.

[74] D. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[75] R. Raman and I. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18(7):563 – 578, 1994.

[76] O. Roussel and V. Manquinho. *Pseudo-Boolean and Cardinality Constraints*, chapter 22, pages 695–733. In Biere et al. [25], February 2009.

[77] N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computers and Chemical Engineering*, 29(9):1891–1913, 2005.

[78] 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.

[79] R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3(3-4):141–224, 2007.

[80] R. Sebastiani and S. Tomasi. Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012. Available at `http://disi.unitn.it/~rseba/publist.html`.

[81] R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transaction on Computational Logics*, January 2014. Submitted.

[82] M. Sellmann and S. Kadioglu. Dichotomic Search Protocols for Constrained Optimization. In *International Conference on Principles and Practice of Constraint Programming*, volume 5202 of *LNCS*. Springer, 2008.

[83] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. AAAI*. MIT Press, 1994.

[84] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In *Proc. Formal Methods in Computer-Aided Design, FMCAD07*, LNCS, pages 108–125. Springer, 2000.

[85] R. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31:1–12, 1984.

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

[87] S. Tomasi. Stochastic Local Search for SMT. Technical report, DISI-10-060, DISI, University of Trento, 2010. Available at `http://eprints.biblio.unitn.it/`.

[88] D. Tompkins and H. Hoos. Novelty+ and Adaptive Novelty+. *SAT 2004 Competition Booklet*, 2004.

[89] D. Tompkins and H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In *Proc. of International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 3542 of *LNCS*. Springer, 2004.

[90] P. Trentin. Linear Integer Optimization with SMT. Master's thesis, Computer Science School, DISI, University of Trento, Italy, March 2014.

[91] D. van Dalen. *Logic and structure (2. ed.).* Universitext. Springer, 1989.

[92] A. Vecchietti, 2011. Personal communication.

[93] A. Vecchietti and I. Grossmann. Computational experience with logmip solving linear and nonlinear disjunctive programming problems. In *Proc. of The Foundations of Computer-Aided Process Design, FOCAPD*, pages 587–590, 2004.

[94] S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. of International Joint Conferences on Artificial Intelligence, IJCAI*, 1999.

[95] Z. Xing and W. Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2):47 – 80, 2005.

[96] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press.