Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive, 14 - 38123 POVO, Trento - Italy
http://disi.unitn.it

Automated Parameter Configuration for an SMT Solver

Duy Tin Truong

November 2010

Technical Report # DISI-10-057 Truong Duy Tin

UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea Specialistica in Informatica

Tesi dal Titolo

Automated Parameter Configuration for an SMT Solver

Relatore:
Roberto Sebastiani

Laureando:
Duy Tin Truong

Anno Accademico 2009/2010

Contents

Chapter 1

Introduction

In this thesis, we use an Automatic Configuration Framework (implemented in *ParamILS*) to find the most suitable techniques for a set of popular theories solved by Satisfiability Modulo Theories (SMT). The techniques that we investigate are the most effective techniques of interest for the lazy SMT and which have been proposed in various communities and implemented in the *MathSAT* tool.

The ultimate goal of this thesis is to provide the guidelines about the choice of optimized techniques for solving popular theories using SMT.

1.1 Satisfiability Modulo Theories and MathSAT

Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to some decidable first-order theory $\mathcal{T}(SMT(\mathcal{T}))$. These problems are typically not handled adequately by standard automated theorem provers. SMT is being recognized as increasingly important due to its applications in many domains in different communities, in particular in formal verification.

Typical *SMT($\mathcal{T}$)* problems require testing the satisfiability of formulas which are Boolean combinations of atomic propositions and atomic expressions in $\mathcal{T}$, so that heavy Boolean reasoning must be efficiently combined with expressive theory-specific reasoning. The dominating approach to *SMT($\mathcal{T}$)*, called *lazy approach*, is based on the integration of a SAT solver (widely used is a modern conflict-driven DPLL solver) and of a decision procedure able to handle sets of atomic constraints in $\mathcal{T}$ ($\mathcal{T}$-solver), handling respectively the Boolean and the theory-specific components of reasoning.

An amount of papers with novel and very efficient techniques for optimizing the integration of DPLL and $\mathcal{T}$-solver has been published in the last years, and some very efficient SMT tools are now available. However, it is still very difficult

to decide which technique is the most suitable one for a theory, or even harder, which combination of techniques is the best choice for a theory. Therefore, in this thesis, we use *MathSAT*, one of the efficient SMT tools, which implements most of these techniques to compare the effectiveness of techniques on different theories.

1.2 Automatic Configuration Framework

The identification of performance-optimizing parameter settings is an important part of the development and application of algorithms. Usually, we start with some parameter configuration, and then modify a single parameter. If the result is improved after tuning, we keep this new result. We repeat this job until some termination criteria is satisfied. This approach is very expensive in term of human time and the performance is also very poor. Fortunately, Frank Hutter and Holger H. Hoos has proposed *ParamILS*, an automatic configuration framework, to solve this problem automatically and effectively. Experiments on many algorithms like SAPS, SPEAR, CPLEX with different benchmarks Graph colouring (Gent et.el, 1999), Quasigroup completion (Gomes and Selman, 1997), etc. has proved that the configurations found by ParamILS outperform the default configurations in all cases, especially faster than 50 times for some special cases. Therefore, in this thesis, we use ParamILS to search for the best possible configurations of MathSAT on different theories.

1.3 Summary of Contributions

The main contribution of this thesis is a comprehensive study of the most effective SMT techniques by mean of the MathSAT and ParamILS tool. This includes an empirical analysis approach to study the characteristics of the MathSAT configuration scenario, two experimental groups on eight and five theories to determine the best possible configurations for MathSAT on the SMT-COMP 2009 and SMT-LIB benchmark. Here, we describe these in more detail:

- We do many experiments on one theory to determine the most suitable scenario for MathSAT before starting experiments on a set of theories. The main parameters in a scenario are the ParamILS strategy (Basic or Focused), the timeout of ParamILS (tunerTimeout), the timeout of each MathSAT run(cutoff_time), the effect of ParamILS random seeds, the determinism of MathSAT.

- Then, we start ParamILS on eight theories using the SMT-COMP 2009

benchmark. In these experiments, we use the same dataset of SMT-COMP 2009 for training and testing phases in order to check whether we can have better configurations than the default configurations and the configurations used in SMT-COMP 2009 (smt-comp configurations, these configurations can be ***changed*** according to different problem classes based on a statistics module in MathSAT). In three of five cases, the number solved tests of the optimized configurations increases significantly compared with the smt-comp configurations. In two other cases, although the number of solved tests are equal to the number of tests solved by the smt-comp configurations, the mean runtime is reduced approximately by *half* and by *a factor of eight*. In addition, in training and testing phases, we also obtain sets of MathSAT bugs on three theories and report them to the MathSAT team.

- Next, we use the benchmark selection tool of SMT-COMP 2009 to extract from the SMT-LIB benchmark different training and testing datasets for five successfully tested theories (no errors found in training and testing phases for these theories in Chapter 5) to find general optimized MathSAT configurations. In all cases, the number of tests solved by the optimized configurations is much larger than the number of tests solved by the default configurations.

Part I

BACKGROUND AND STATE OF THE ART

Chapter 2

SMT Techniques and MathSAT

The description in this chapter is mostly taken from [4].

2.1 Lazy SMT in MathSAT

2.1.1 Satisfiability Modulo Theories - SMT

Satisfiability Modulo Theories is the problem of deciding the satisfiability of a first-order formula with respect to some decidable first-order theory $\mathcal{T}(SMT(\mathcal{T}))$. Examples of theories of interest are those of *Equality and Uninterpreted Functions* $(\mathcal{EUF})$, *Linear Arithmetic* $(\mathcal{LA})$, both over the reals $(\mathcal{LA}(\mathbb{Q}))$ and the integers $(\mathcal{LA}(\mathbb{Z}))$, its subclasses of *Difference Logic* $(\mathcal{DL})$ and *Unit-Two-Variable-Per-Inequality* $(\mathcal{UTVPI})$, the theories of *bit-vectors* $(\mathcal{BV})$, of *arrays* $(\mathcal{AR})$ and of *lists* $(\mathcal{LI})$. These problems are typically not handled adequately by standard automated theorem provers - like, e.g., those based on resolution calculus - because the latter cannot satisfactorily deal with the theory-specific interpreted symbols (i.e., constants, functions, predicates).

SMT is being recognized as increasingly important due to its applications in many domains in different communities, ranging from resource planning [95] and temporal reasoning [19] to formal verification, the latter including verification of pipelines and of circuits at Register-Transfer Level (RTL) [39, 75, 33], of proof obligations in software systems [76, 52], of compiler optimizations [29], of real-time embedded systems [23, 45, 22].

An amount of papers with novel and very efficient techniques for SMT has been published in the last years, and some very efficient SMT tools are now available (e.g., Ario [81], BarceLogic [72], CVCLite/CVC3 [25], DLSAT [67], haRVey [76], MathSAT [34], Sateen [64], SDSAT [53] Simplify [46], TSAT++ [20], UCLID [65], Yices [47], Verifun [51], Zapato [24]), Z3 [43]. An amount of bench-

marks, mostly derived from verification problems, is available at the SMT-LIB official page [77, 78]. A workshop devoted to SMT and an official competition on SMT tools are run yearly.

For a complete survey, please refer [4] and [12].

2.1.2 Lazy SMT = SAT + $\mathcal{T}$-Solvers

All applications mentioned above require testing the satisfiability of formulas which are (possibly-big) Boolean combinations of atomic propositions and atomic expressions in some theory $\mathcal{T}$, so that heavy Boolean reasoning must be efficiently combined with expressive theory-specific reasoning.

On the one hand, in the last decade we have witnessed an impressive advance in the efficiency of propositional satisfiability techniques, SAT [85, 30, 69, 58, 49, 48]). As a consequence, some hard real-world problems have been successfully solved by encoding them into SAT. SAT solvers are now a fundamental tool in most formal verification design flows for hardware systems, both for equivalence, property checking, and ATPG [31, 68, 89]; other application areas include, e.g., the verification of safety-critical systems [88, 32], and AI planning in its classical formulation [63], and in its extensions to non-deterministic do-mains [40, 59]. Plain Boolean logic, however, is not expressive enough for representing many other real-world problems (including, e.g., the verification of pipelined microprocessors, of real-time and hybrid control systems, and the analysis of proof obligations in software verification); in other cases, such as the verification of RTL designs or assembly-level code, even if Boolean logic is expressive enough to encode the verification problem, it does not seem to be the most effective level of abstraction (e.g., words in the data path are typically treated as collections of unrelated Boolean variables).

On the other hand, decision procedures for much more expressive decidable logics have been conceived and implemented in different communities, like, e.g., automated theorem proving, operational research, knowledge representation and reasoning, AI planning, CSP, formal verification. In particular, since the pioneering work of Nelson and Oppen [70, 71, 73, 74] and Shostak [83, 84], efficient procedures have been conceived and implemented which are able to check the consistency of sets/conjunctions of atomic expressions in decidable F.O. theories. (We call these procedures, Theory Solvers or $\mathcal{T}$-solvers.) To this extent, most effort has been concentrated in producing $\mathcal{T}$-solvers of increasing expressiveness and efficiency and, in particular, in combining them in the most efficient way (e.g., [70, 71, 73, 74, 83, 84, 50, 28, 80]). These procedures, however, deal only with conjunctions of atomic constraints, and thus cannot handle the Boolean component of reasoning.

In the last ten years new techniques for efficiently integrating SAT solvers

with logic- specific or theory-specific decision procedures have been proposed in different communities and domains, producing big performance improvements when applied (see, e.g., [57, 61, 19, 95, 45, 27, 21, 93, 67, 51, 54, 34, 82]). Most such systems have been implemented on top of SAT techniques based on variants of the DPLL algorithm [42, 41, 85, 30, 69, 58, 49, 48].

In particular, the dominating approach to SMT ($\mathcal{T}$), which underlies most state-of-the-art SMT ($\mathcal{T}$) tools, is based on the integration of a SAT solver and one (or more) $\mathcal{T}$-solver(s), respectively handling the Boolean and the theory-specific components of reasoning: the SAT solver enumerates truth assignments which satisfy the Boolean abstraction of the input formula, whilst the $\mathcal{T}$-solver checks the consistency in $\mathcal{T}$ of the set of literals corresponding to the assignments enumerated. This approach is called *lazy*, in contraposition to the eager approach to SMT ($\mathcal{T}$), consisting on encoding an SMT formula into an equivalently-satisfiable Boolean formula, and on feeding the result to a SAT solver (see, e.g., [94, 38, 92, 91, 79]). All the most extensive empirical evaluations performed in the last years [54, 44, 72, 35, 26, 86, 87] confirm the fact that currently all the most efficient SMT tools are based on the lazy approach.

2.1.3 MathSAT

MathSAT is a DPLL-based decision procedure for the SMT problem for various theories, including those of Equality and Uninterpreted Function (EUF), Difference Logics (DL), Linear Arithmetic over the Reals (LA(R)) and Linear Arithmetic over the Integers (LA(Z)). MathSAT is based on the approach of integrating a state-of-the-art SAT solver with a hierarchy of dedicated solvers for the different theories, and implements several optimization techniques. MathSat pioneers a lazy and layered approach, where propositional reasoning is tightly integrated with solvers of increasing expressive power, in such a way that more expensive layers are called less frequently. MathSAT has been applied in different real-world application domains, ranging from formal verification of infinite state systems (e.g. timed and hybrid systems) to planning with resources, equivalence checking and model checking of RTL hardware designs. For more detail, please visit the MathSAT website *http://mathsat4.disi.unitn.it/*.

2.2 SMT Techniques Implemented in MathSAT

2.2.1 Concept of the section

Before presenting SMT techniques in detail, we present the following basic concepts that are used throughout this section.

Let Σ be a first-order signature containing function and predicate symbols with their arities, and $\mathcal{V}$ be a set of variables. A 0-ary function symbol c is called a *constant*. A 0-ary predicate symbol A is called a *Boolean atom*. A Σ-term is either a variable in $\mathcal{V}$ or it is built by applying function symbols in Σ to Σ-terms. If $t_1, ..., t_n$ are Σ-terms and P is a predicate symbol, then $P(t_1, ..., t_n)$ is a Σ-atom. A Σ-literal is either a Σ-atom (a positive literal) or its negation (a negative literal). The set of Σ-atoms and Σ-literals occurring in φ are denoted by $Atoms(\varphi)$ and $Lits(\varphi)$ respectively.

Given a decidable first-order theory $\mathcal{T}$, we call a *theory solver for* $\mathcal{T}$, $\mathcal{T}$-solver, any tool able to decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of ground atomic formulas and their negations - *theory literals or* $\mathcal{T}$*-literals* - in the language $\mathcal{T}$.

We will often use the prefix "$\mathcal{T}$-" to denote "in the theory $\mathcal{T}$": e.g., we call a "$\mathcal{T}$-formula" a formula in (the signature of) $\mathcal{T}$, "$\mathcal{T}$-model" a model in $\mathcal{T}$, and so on. We also use the bijective function $\mathcal{T}2B$ ("Theory-to-Boolean") and its inverse $\mathcal{B}2\mathcal{T} := \mathcal{T}2\mathcal{B}^{-1}$ ("Boolean-to-Theory"), s.t. $\mathcal{T}2\mathcal{B}$ maps Boolean atoms into themselves and non-Boolean $\mathcal{T}$-atoms into fresh Boolean atoms - so that two atom instances in φ are mapped into the same Boolean atom iff they are syntactically identical - and distributes with sets and Boolean connectives.

For the combination of theories, we use the concept of *interface equalities*, that is, equalities between variables appearing in atoms of different theories (*interface variables*).

2.2.2 Integration of DPLL and $\mathcal{T}$-Solver

Several procedures exploiting the integration schema have been proposed in different communities and domains (see, e.g., [57, 95, 19, 21, 51, 54, 36]). In this integration schema, $\mathcal{T}$-DPLL is a variant of the DPLL procedure, modified to work as an enumerator of truth assignments, whose $\mathcal{T}$-satisfiability is checked by a $\mathcal{T}$-solver.

Procedure 1 represents the schema of a $\mathcal{T}$-DPLL procedure based on a modern DPLL engine. The input φ and μ are a $\mathcal{T}$-formula and a reference to an (initially empty) set of $\mathcal{T}$-literals respectively. The DPLL solver embedded in $\mathcal{T}$-DPLL reasons on and updates φ^p and μ^p, and $\mathcal{T}$-DPLL maintains some data structure encoding the set Lits(φ) and the bijective mapping $\mathcal{T}2\mathcal{B}/\mathcal{B}2\mathcal{T}$ on literals.

$\mathcal{T}$-preprocess simplifies φ into a simpler formula, and updates μ if it is the case, so that to preserve the $\mathcal{T}$-satisfiability of $\varphi \wedge \mu$. If this process produces some conflict, then $\mathcal{T}$-DPLL returns Unsat. $\mathcal{T}$-preprocess combines most or all the Boolean preprocessing steps with some theory-dependent rewriting steps on the $\mathcal{T}$-literals of φ.

$\mathcal{T}$-decide_next_branch implements the key non-deterministic step in DPLL, for which many heuristic criteria have been conceived. Old-style heuristics like

Procedure 1 SatValue $\mathcal{T}$-DPLL ($\mathcal{T}$-formula φ, $\mathcal{T}$-assignment &μ)

```
1: if (T-preprocess(φ, μ) == Conflict) then
2:    return Unsat;
3: end if
4: φ^p = T2B(φ); μ^p = T2B(μ);
5: while (1) do
6:    T-decide_next_branch(φ^p, μ^p)
7:    while (1) do
8:       status = T-deduce(φ^p, μ^p)
9:       if (status == Sat) then
10:         μ = B2T(μ^p)
11:         return Sat
12:      else if (status == Conflict) then
13:         blevel = T-analyze_conflict(φ^p, μ^p)
14:         if (blevel == 0) then
15:            return Unsat
16:         else
17:            T-backtrack(blevel, φ^p, μ^p)
18:         end if
19:      else
20:         break
21:      end if
22:   end while
23: end while
```

MOMS and Jeroslow- Wang [62] used to select a new literal at each branching point, picking the literal occurring most often in the minimal-size clauses (see, e.g., [60]). The heuristic implemented in SATZ [66] selects a candidate set of literals, performs Boolean Constraint Propagation (BCP), chooses the one leading to the smallest clause set; this maximizes the effects of BCP, but introduces big overheads. When formulas derive from the encoding of some specific problem, it is sometimes useful to allow the encoder to provide to the DPLL solver a list of ”privileged” variables on which to branch first (e.g., action variables in SAT-based planning [56], primary inputs in bounded model checking [90]). Modern conflict-driven DPLL solvers adopt evolutions of the VSIDS heuristic [69, 58, 49], in which decision literals are selected according to a score which is updated only at the end of a branch, and which privileges variables occurring in recently-learned clauses; this makes $\mathcal{T}$-decide_next_branch state-independent (and thus much faster, because there is no need to recomputing the scores at each decision) and allows it to take into account search history, which makes search

more effective and robust. In addition, $\mathcal{T}$-decide_next_branch takes into consideration also the semantics in $\mathcal{T}$ of the literals to select.

$\mathcal{T}$-deduce 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:

(i) μ^p propositionally violates φ^p ($\mu^p \wedge \varphi^p \models_p \bot$). If so, $\mathcal{T}$-deduce returns *Conflict*.

(ii) μ^p propositionally satisfies $\varphi^p (\mu^p \models_p \varphi^p)$. If so, $\mathcal{T}$-deduce invokes $\mathcal{T}$-solver on $\mathcal{B}2\mathcal{T}(\mu^p)$: if the latter returns *Sat*, then $\mathcal{T}$-deduce returns *Sat*; otherwise, $\mathcal{T}$-deduce returns *Conflict*.

(iii) no more literals can be deduced. If so, $\mathcal{T}$-deduce returns *Unknown*. A slightly more elaborated version of $\mathcal{T}$-deduce can invoke $\mathcal{T}$-solver on $\mathcal{B}2\mathcal{T}(\mu^p)$ also at this intermediate stage: if $\mathcal{T}$-solver returns *Unsat*, then $\mathcal{T}$-deduce returns *Conflict*.

$\mathcal{T}$-analyze_conflict: if the conflict produced by $\mathcal{T}$-deduce is caused by a Boolean failure (case (i) above), then $\mathcal{T}$-analyze_conflict produces a Boolean conflict set η^p and the corresponding value of blevel; if instead the conflict is caused by a $\mathcal{T}$-inconsistency revealed by $\mathcal{T}$-solver (case (ii) or (iii) above), then $\mathcal{T}$-analyze_conflict produces as a conflict set the Boolean abstraction η^p of the theory conflict set η produced by $\mathcal{T}$-solver (i.e., $\eta^p := \mathcal{T}2\mathcal{B}(\eta)$), or computes a mixed Boolean+theory conflict set by a backward-traversal of the implication graph starting from the conflicting clause $\neg\mathcal{T}2\mathcal{B}(\mu)$. 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. Once the conflict set η^p and blevel have been computed, **$\mathcal{T}$-backtrack** adds the clause $\neg\eta^p$ to φ^p and backtracks up to blevel.

2.2.3 Adaptive Early Pruning

In its simplest form, Early Pruning (EP) is based on the empirical observation that most assignments which are enumerated by $\mathcal{T}$-DPLL, and which are found Unsat by $\mathcal{T}$-solver, are such that their $\mathcal{T}$-unsatisfiability is caused by much smaller subsets. Thus, if the $\mathcal{T}$-unsatisfiability of an assignment μ is detected during its construction, then this prevents checking the $\mathcal{T}$-satisfiability of all the up to $2^{|Atoms(\phi)|-|\mu|}$ total truth assignments which extend μ. However, as EP may cause useless calls to $\mathcal{T}$-solver, the benefits of the pruning effect may be partly counterbalanced by the overhead introduced by the extra EP calls [4].

A standard solution for this problem, adopted by several SMT solvers, is to use incomplete but fast $\mathcal{T}$-solvers for EP calls, performing the complete but

potentially-expensive check only when absolutely necessary (i.e. when a truth assignment which propositionally satisfies the input formula is found). This technique is usually called Weak (or Approximate) Early Pruning.

In MathSAT, a different approach, which we call Adaptive Early Pruning (AEP), is implemented. The main idea of AEP is that of controlling the frequency of EP calls, by adapting the rate at which $\mathcal{T}$-solvers are invoked according to some measure of the usefulness of EP: the more EP calls contribute to pruning the search by detecting $\mathcal{T}$-conflicts or $\mathcal{T}$-deductions, the more frequently $\mathcal{T}$-solvers are invoked [5].

In MathSAT, the parameter of this technique is *aep*. We experiment on the following values: {yes: enable, no: disable}.

2.2.4 $\mathcal{T}$-propagation

T -propagation was introduced in its simplest form (plunging, see [4]) by [19] for DL; [21] proposed an improved technique for LA; however, T -propagation showed its full potential in [93, 54, 72], where it was applied aggressively.

As discussed in [4], for some theories it is possible to implement $\mathcal{T}$-solver so that a call to $\mathcal{T}$-solver(μ) returning Sat can also perform one or more deduction(s) in the form $\eta \models_T l$, s.t. $\eta \subseteq \mu$ and l is a literal on a not-yet-assigned atom in ϕ. If this is the case, then $\mathcal{T}$-solver can return l to $\mathcal{T}$-DPLL, so that $\mathcal{T} \in \mathcal{B}(l)$ is unit-propagated. This may induce new literals to be assigned, new calls to $\mathcal{T}$-solver, new assignments deduced, and so on, possibly causing a beneficial loop between $\mathcal{T}$-propagation and unit-propagation. Notice that $\mathcal{T}$-solver can return the deduction(s) performed $\eta \models_T l$ to $\mathcal{T}$-DPLL, which can add the deduction clause $\mathcal{T} 2 \mathcal{B}(\eta \rightarrow l)$ to φ^p, either temporarily and permanently. The deduction clause will be used for the future Boolean search, with benefits analogous to those of $\mathcal{T}$-learning (see [4]).

In MathSAT, the parameter of this technique is *deduction*. This parameter is used to set the deduction level of theories and we experiment on the following values: {0,1,2,3}.

2.2.5 Dual Rail Encoding

We would like to reduce the number of literals sent to the bit-vector theory solver, since each theory solver call is potentially very expensive. One way to do this is to have the boolean enumerator enumerate minimal models. In [13], Roorda and Claessen uses a technique based on a dual-rail encoding which gives minimal models for the SAT problem, and the same technique lifts into SMT.

In a dual rail encoding of a formula, each propositional atom P is replaced by two fresh atoms $P^{\top}$ and $P^{\bot}$. These are used to encode a three valued semantics of

$P^{\intercal}$	P^{τ}	Meaning
False	False	No value
False	True	False
True	False	True
True	True	Illegal

Table 2.1: Three value logic semantic of dual rail encoding

propositional logic according to table 2.1. To translate a formula in CNF to dual rail, all positive literals A are replaced with $A^{\intercal}$, and all negative literals $\neg A$ are replace with $A^{\perp}$. To rule out the illegal value, for every atom A the clause $\{\neg A^{\intercal}$, $\neg A^{\perp}\}$ is added to the CNF.

To see why this encoding would help in enumerating minimal models, we can notice that in DPLL, if the decision heuristic always assigns false to decision variables, then any model μ for a set of clauses Γ has the minimal number of positive literals. This means that it is not possible to negate any of the positive literals in μ and still have $\mu \models \Gamma$. We say that such a model is (positive) sign-minimal. The reverse is true if the decision heuristic always assigns true to decision variables, and we call such models negative sign-minimal. See [6] for the full proof.

In MathSAT, the parameter of this technique is *dual_rail*. We experiment on the following values:

- off: disables dual rail encoding.
- circuit: ensures enumerating minimal models for the original formula.
- cnf: a "lighter" version that introduces less clauses, but only ensures that the enumerated models are minimal w.r.t. the CNF-conversion of the original problem (i.e., they might not be minimal for the original formula).

2.2.6 Dynamic Ackermann Expansion

When the theory $\mathcal{T}$ solved is combination of many theories, and one of the theories $\mathcal{T}_i$ is $\mathcal{EUF}$, one further approach to the $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$ problem is to eliminate uninterpreted function symbols by means of Ackermanns expansion [18] so that to obtain an $SMT(\mathcal{T})$ problem with only one theory. The method works by replacing every function application occurring in the input formula φ with a fresh variable and then adding to φ all the needed functional congruence constraints. The new formula φ' obtained is equisatisfiable with φ, and contains no uninterpreted function symbols.

However, the traditional congruence-closure algorithm misses the propagation rule $f(x) \neq f(y) \rightsquigarrow x \neq y$ which has a dramatic performance benefit on many

problems. An approach called *Dynamic Ackermannization* is proposed to cope with this problem [10].

In MathSAT, the parameter of this technique is *dyn_ack*. We experiment on the following values: {yes: enable, no: disable}.

Besides, this parameter is used together with two following parameters:

- *dyn_ack_limit* Maximum number of clauses added by dynamic Ackermann's expansion. We experiment this parameter only on the value of 0 which means unlimited.

- *dyn_ack_threshold* Number of times a congruence must be used before activating its dynamic Ackermann's expansion. We experiment this parameter on the following values: {0, 10,50}.

2.2.7 Boolean Conflict Clause Minimization

Let C and C' be clauses, $\otimes_x$ the resolution operator on variable x. If $C \otimes_x C' \subseteq C$ then C is said to be self-subsumed by C' w.r.t. x. In effect, C' is used to remove x (or x) from C by the fact that C is subsumed by $C \otimes_x C'$. A particularly useful and simple place to apply self-subsumption is in the conflict clause generation. The following 5-line algorithm can easily be added to any clause recording SAT solver [11]:

strengthenCC(*Clause C*) - C is the conflict clause
for each $p \in C$ **do**
if $(reason(\bar{p}) \setminus \{p\} \subseteq C)$ **then**
mark p
end if
remove all marked literals in C
end for

By $reason(p)$ we denote the clause that became unit and propagated $p = True$.

In MathSAT, the parameter of this technique is *expensive_ccmin*. We experiment on the following values: {yes: enable, no: disable}.

2.2.8 Learned Clauses Deleting

In the $\mathcal{T}$*-learning* technique (see [4]), when a conflict set η is found, the clause $\mathcal{T}2\mathcal{B}(\neg\eta)$ is added in conjunction to φ^p. Since then, $\mathcal{T}$-DPLL will never again generate any branch containing η. In fact, as soon as $|\eta| - 1$ literals in η are

assigned to true, the remaining literal will be immediately assigned to false by unit-propagation on $\mathcal{T}2\mathcal{B}(\neg\eta)$. However, *$\mathcal{T}$-learning* must be used with some care, because it may cause an explosion in size of φ. To avoid this, one has to introduce techniques for discarding learned clauses when necessary [30].

In MathSAT, the parameter of this technique is *frequent_reduce_db*. We experiment on the following values: {yes: aggressively delete learned clause that are consider unrelevant, no: not delete aggressively}.

2.2.9 Ghost Filtering

In Lazy SMT, when DPLL decides the next branch, it may select also literals which occur only in clauses which have already been satisfied (which we call "ghost literals"). The technique 'Ghost Filter' prevents DPLL from splitting branches on the ghost literals.

In MathSAT, the parameter of this technique is *ghost_filter*. We experiment on the following values: {yes: enable, no: disable}.

2.2.10 Increase The Initial Weight of Boolean Variables

Increase the initial weight of non-theory atoms which were not introduced by the cnf conversion in the splitting heuristic. It means that the original non-theory atoms have the highest score, and the introduced variables have the lowest score.

In MathSAT, the parameter of this technique is *ibliwi*. We experiment on the following values: {yes: enable, no: disable}.

2.2.11 Threshold for Lazy Explanation of Implications

MathSAT can learn only the clauses whose length less than n (which is called *threshold for lazy explanation of implications*) and other clauses in the conflict set on demand.

In MathSAT, the parameter of this technique is *impl_expl_threshold*. We experiment on the following values: {0: learn all clauses, 1}.

2.2.12 Incremental Theory Solvers

MathSAT can introduce and handle new atoms during search.

In MathSAT, the parameter of this technique is *incr_tsolvers*. We experiment on the following values: {yes: enable this feature, no: disable this feature}.

2.2.13 Mixed Boolean+Theory Conflict Clauses

In the online approach to integrate SAT and $\mathcal{T}$-solver [4], when $\mathcal{T}$-DPLL reaches the status of *Conflict*, it will call *$\mathcal{T}$-analyze-conflict* to analyze the failure. If conflict produced by *$\mathcal{T}$-deduce* is caused by a Boolean failure, then *$\mathcal{T}$-analyze-conflict* produces a Boolean conflict set η^p and the corresponding value of *blevel*, as described in [4] if instead the conflict is caused by a *$\mathcal{T}$-inconsistency* revealed by *$\mathcal{T}$-solver*, then *$\mathcal{T}$-analyze-conflict* produces as a conflict set the Boolean abstraction η^p of the theory conflict set η produced by *$\mathcal{T}$-solver* (i.e., $\eta^p := T2B(\eta)$), or computes a mixed Boolean+theory conflict set by a backward-traversal of the implication graph starting from the conflicting clause $\neg\mathcal{T}2\mathcal{B}(\eta)$ (see [4]). 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. Once the conflict set η^p and *blevel* have been computed, *$\mathcal{T}$-backtrack* behaves analogously to *backtrack* in DPLL: it adds the clause $\neg\eta^p$ to φ^p and backtracks up to *blevel*.

In MathSAT, the parameter of this technique is *mixed_cs*. We experiment on the following values: {yes: enable. no: disable}.

2.2.14 Permanent Theory Lemmas

If $S = \{l_1, ..., l_n\}$ is a set of literals in $\mathcal{T}$, we call *($\mathcal{T}$)-conflict set* any subset η of S which is inconsistent in $\mathcal{T}$. We call $\neg\eta$ a $\mathcal{T}$-lemma [9]. (Notice that $\neg\eta$ is a $\mathcal{T}$-valid clause.) These $\mathcal{T}$-lemma can be deleted to avoid explosion when necessary.

In MathSAT, the parameter of this technique is *permanent_theory_lemmas*. We experiment on the following values: {yes: never delete theory lemmas, no: delete when necessary}.

2.2.15 Pure Literal Filtering

This technique, which we call pure-literal filtering, was implicitly proposed by [95] and then generalized by [55, 21, 35].

The idea is that, if we have non-Boolean $\mathcal{T}$-atoms occurring only positively [resp. negatively] in the input formula, we can safely drop every negative [resp. positive] occurrence of them from the assignment to be checked by $\mathcal{T}$-solver. Moreover, if both $\mathcal{T}$-propagation and pure-literal filtering are implemented, then the filtered literals must be dropped not only from the assignment, but also from the list of literals which can be $\mathcal{T}$-deduced by $\mathcal{T}$-solver, so that to avoid the $\mathcal{T}$-propagation of literals which have been filtered away.

We notice first that pure-literal filtering has the same two benefits described for reduction to prime implicants [4]. Moreover, this technique is particularly

useful in some situations. For instance, in DL(Z) and LA(Z) many solvers cannot efficiently handle disequalities (e.g., $(x_1 - x_2 \neq 3)$), so that they are forced to split them into the disjunction of strict inequalities $(x_1 - x_2 > 3) \vee (x_1 - x_2 < 3)$. (This is done either off-line, by rewriting all equalities [resp. disequalities] into a conjunction of inequalities [resp. a disjunction of strict inequalities], or on-line, at each call to T -solver.) This causes an enlargement of the search, because the two disjuncts must be investigated separately.

However, in many problems it is very frequent that many equalities $(t_1 = t_2)$ occur with positive polarity only. If so, pure-literal filtering avoids adding $(t_1 \neq t_2)$ to μ when $\mathcal{T}2\mathcal{B}((t_1 = t_2))$ is assigned to false by $\mathcal{T}$-DPLL, so that no split is needed [21].

In MathSAT, the parameter of this technique is *pure_literal_filter*. This parameter may affect negatively theory deduction, but can be a benefit for complex theories. We experiment on the following values {yes: enable, no: disable}.

2.2.16 Random Decisions

Perform randomly about 5% of the branching decisions.

In MathSAT, the parameter of this technique is *random_decisions*. We experiment on the following values: {yes: enable, no: disable}.

2.2.17 Restart

When searching for a solution, SMT can get stuck in some branches. In that case, the solution is to escape from the current branch by restarting the whole process.

In MathSAT, the parameter of this technique is *restart*. We experiment on the following values:

- Normal: The restart policies implemented in MiniSat. They all observe different search parameters like: conflict level (the height of the search tree (i.e. the number of decisions) when a conflict occurred) or backtrack level (the height of the search tree to which the solver jumped back to resolve the conflict), length of learned clauses (the length of the currently learned clause), and trail size (the total number of assigned variables when a conflict occurred (including variables assigned by unit propagation)) over time and, based on their development, decide whether to perform a restart or not [16].

- Quick: Using a small prototype SAT solver, called *TINISAT*, which implements the essentials of a modern clause learning solver and is designed to facilitate adoption of arbitrary restart policies [7].

- Adaptive: This technique measures the ”agility” of the SAT solver as it traverses the search space, based on the rate of recently flipped assignments. The level of agility dynamically determines the restart frequency. Low agility enforces frequent restarts, high agility prohibits restarts [8].

2.2.18 Static Learning

The technique was proposed by [19] for a lazy SMT procedure for DL. Similar such techniques were generalized and used in [23, 35, 96].

On some specific kind of problems, it is possible to quickly detect a priori short and 'obviously $\mathcal{T}$-inconsistent' assignments to $\mathcal{T}$-atoms in Atoms(φ) (typically pairs or triplets). Some examples are:

- incompatible value assignments (e.g., $x = 0, x = 1$),
- congruence constraints (e.g., $(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))$),
- transitivity constraints (e.g., $(x - y \leq 2), (y - z \leq 4), \neg(x - z \leq 7)$),
- equivalence constraints ($(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)$).

If so, the clauses obtained by negating the assignments (e.g.,$\neg(x = 0) \vee \neg(x = 1)$) can be added a priori to the formula before the search starts. Whenever all but one literal in the inconsistent assignment are assigned, the negation of the remaining literal is assigned deterministically by unit-propagation, which prevents the solver generating any assignment which include the inconsistent one. This technique may significantly reduce the Boolean search space, and hence the number of calls to $\mathcal{T}$-solver, producing very relevant speed-ups [19, 23, 35, 96].

Intuitively, one can think of static learning as suggesting a priori some small and 'obvious' $\mathcal{T}$-valid lemmas relating some $\mathcal{T}$-atoms of ϕ, which drive DPLL in its Boolean search. Notice that, unlike the extra clauses added in 'per-constraint' eager approaches [92, 79] (see [4]), the clauses added by static learning refer only to atoms which already occur in the original formula, so that the Boolean search space is not enlarged, and they are not needed for correctness and completeness: rather, they are used only for pruning the Boolean search space.

In MathSAT, the parameter of this technique is *sl*. This parameter is used to set the level of static learning. And we experiment on the following values: $\{0$: disable, 1, $2\}$.

2.2.19 Splitting of Equalities

This technique rewrites $(x = y)$ into $(x \leq y)\ and\ (y \leq x)$ during preprocessing.

In MathSAT, the parameter of this technique is *split_eq*. We experiment on the following values {yes: enable, no: disable}.

2.2.20 Theory Combination

In many practical applications of SMT, the theory $\mathcal{T}$ is a combination of two or more theories $\mathcal{T}_1, ..., \mathcal{T}_n$. For instance, an atom of the form $f(x + 4y) = g(2xy)$, that combines uninterpreted function symbols (from $\mathcal{EUF}$) with arithmetic functions (from $\mathcal{LA}(\mathbb{Z})$), could be used to naturally model in a uniform setting the abstraction of some functional blocks in an arithmetic circuit (see e.g. [37, 33]).

In MathSAT, the parameter of this technique is *tcomb*. We experiment on the following values:

- off: Do not use theory combination.
- ack: Using Ackermann's expansion as described in section 2.2.6.
- dtc: Delayed Theory Combination (DTC) is a general approach for combining theories in SMT proposed in [14]. With DTC, the solvers for $\mathcal{T}_1$ and $\mathcal{T}_2$ do not communicate directly. The integration is performed by the SAT solver, by augmenting the Boolean search space with up to all the possible interface equalities, so that each truth assignment on both original atoms and interface equalities is checked for consistency independently on both theories [9].
- decide: Use heuristics to select ack or dtc.

2.2.21 Propagation of Toplevel Information

Use top-level equalities to simplify the formula. Example: $(x = 0)\ and\ (y \leq x)$ can be rewritten to $(y \leq 0)$.

In MathSAT, the parameter of this techniques is *toplevelprop*. The set of values that we experiment is: {0: disable, 1: standard, 2: aggressive}.

Chapter 3

ParamILS

Most description in this chapter is extracted from [3].

3.1 An Automatic Configuration Scenario

The algorithm configuration problem can be informally stated as follows: given an algorithm, a set of parameters for the algorithm and a set of input data, find parameter values under which the algorithm achieves the best possible performance on the input data.

To avoid potential confusion between algorithms whose performance is optimized and algorithms used for carrying out that optimization task, we refer to the former as target algorithms and to the latter as configuration procedures (or simply configurators). The setup is illustrated as in Figure 3.1, the automatic configuration scenario includes an algorithm to be configured and a collection of instances. A configuration procedure executes the target algorithm with specified parameter settings on some or all of the instances, receives information about the performance of these runs, and uses this information to decide about what subsequent parameter configurations to evaluate.

3.2 The ParamILS Framework

This section describes an iterated local search framework called ParamILS. To start with, we fix the other two dimensions, using an unvarying benchmark set of instances and fixed cutoff times for the evaluation of each parameter configuration. Thus, the stochastic optimization problem of algorithm configuration reduces to a simple optimization problem, namely to find the parameter configuration that yields the lowest mean runtime on the given benchmark set. Then, in Section 3.3

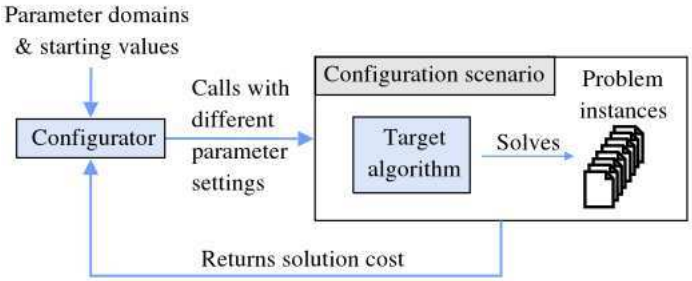


Figure 3.1: An Automated Configuration Scenario

and 3.4, we address the question of how many runs should be performed for each configuration.

Consider the following manual parameter optimization process:

1. begin with some initial parameter configuration;

2. experiment with modifications to single parameter values, accepting new configurations whenever they result in improved performance;

3. repeat step 2 until no single-parameter change yields an improvement.

This widely used procedure corresponds to a manually-executed local search in parameter configuration space. Specifically, it corresponds to an iterative first improvement procedure with a search space consisting of all possible configurations, an objective function that quantifies the performance achieved by the target algorithm with a given configuration, and a neighbourhood relation based on the modification of one single parameter value at a time (i.e., a ”one-exchange” neighbourhood).

Viewing this manual procedure as a local search algorithm is advantageous because it suggests the automation of the procedure as well as its improvement by drawing on ideas from the stochastic local search community. For example, note that the procedure stops as soon as it reaches a local optimum (a parameter configuration that cannot be improved by modifying a single parameter value). A more sophisticated approach is to employ iterated local search [15] to search for performance-optimizing parameter configurations. ILS is a prominent stochastic local search method that builds a chain of local optima by iterating through a main loop consisting of

1. a solution perturbation to escape from local optima,

2. a subsidiary local search procedure and

3. an acceptance criterion to decide whether to keep or reject a newly obtained candidate solution.

ParamILS (given in pseudocode as Procedure 2 and Algorithm Framework 3) is an ILS method that searches parameter configuration space. It uses a combination of default and random settings for initialization, employs iterative first improvement as a subsidiary local search procedure, uses a fixed number (s) of random moves for perturbation, and always accepts better or equally-good parameter configurations, but re-initializes the search at random with probability $p_{restart}$. Furthermore, it is based on a one-exchange neighbourhood, that is, we always consider changing only one parameter at a time. ParamILS deals with conditional parameters by excluding all configurations from the neighbour- hood of a configuration θ that differ only in a conditional parameter that is not relevant in θ.

Procedure 2 IterativeFirstImprovement(θ)

```
repeat
    θ' ← θ;
    for θ'' ∈ Nbh(θ') in randomized order do
        if better(θ'', θ') then
            θ ← θ''; break;
        end if
    end for
until θ' = θ;
return θ;
```

3.3 The BasicILS Algorithm

In order to turn ParamILS as specified in Algorithm Framework 3 into an executable configuration procedure, it is necessary to instantiate the function better that determines which of two parameter settings should be preferred. We will ultimately present several different ways of doing this. Here, we describe the simplest approach, which we call BasicILS. Specifically, we use the term $BasicILS(N)$ to refer to a ParamILS algorithm in which the function $better(\theta_1, \theta_2)$ is implemented as shown in Procedure 4: simply comparing estimates $\hat{c}_N$ of the cost statistics $c(\theta_1)$ and $c(\theta_2)$ that are based on N runs each.

Because BasicParamILS is simple, when benchmark instances are very heterogeneous or when the user can identify a rather small “representative” subset

Algorithm Framework 3 ParamILS($\theta_0, r, p_{restart}, s$)
Outline of iterated local search in parameter conguration space; the specic variants of ParamILS we study, **BasicILS(N)** and **FocusedILS**, are derived from this framework by instantiating procedure *better* (which compares $\theta, \theta' \in \Theta$). *BasicILS(N)* uses $better_N$ (see Procedure 4), while *FocusedILS* uses $better_{Foc}$ (see Procedure 6). The neighbourhood $Nbh(\theta)$ of a configuration θ is the set of all configurations that differ from θ in one parameter, excluding configurations differing in a conditional parameter that is not relevant in θ.

Input: Initial configuration $\theta_0 \in \Theta$, algorithm parameters r, $p_{restart}$, and s.
Output: Best parameter configuration θ found.
for i = 1,...,r **do**
    $\theta \leftarrow random\ \theta \in \Theta$
    if $better(\theta, \theta_0)$ **then**
        $\theta_0 \leftarrow \theta$
    end if
end for
$\theta_{ils} \leftarrow IterativeFirstImprovement(\theta_0)$
while not TerminationCriterion() **do**
    $\theta \leftarrow \theta_{ils}$;
    // ===== Perturbation
    for i = 1,...,s **do**
        $\theta \leftarrow random\ \theta' \in Nbh(\theta)$;
    end for
    // ===== Basic local search
    $\theta \leftarrow IterativeFirstImprovement(\theta)$;
    // ===== AcceptanceCriterion
    if $better(\theta, \theta_{ils})$ **then**
        $\theta_{ils} \leftarrow \theta$;
    end if
    with probability $p_{restart}$ do $\theta_{ils} \leftarrow random\ \theta \in \theta$;
end while
return overall best θ_{inc} found;

Procedure 4 $better_N(\theta_1, \theta_2)$
Procedure used in $BasicILS(N)$ to compare two parameter configurations. Procedure $objective(\theta, N)$ returns the user-defined objective of the target algorithm with the configuration θ on the first N instances, and keeps track of the incumbent solution, θ_{inc}

1: **Input**: Parameter configuration θ_1, parameter configuration θ_2.
2: **Output**: True if θ_1 does better than or equal to θ_2 on the first N instances; false otherwise.
3: **Side Effect**: Adds runs to the global caches of performed algorithm runs R_{θ_1} and R_{θ_2} of configuration θ_1 and θ_2, respectively; potentially updates the incumbent θ_{inc}.
4: $\hat{c}_N(\theta_2) \leftarrow objective(\theta_2, N)$
5: $\hat{c}_N(\theta_1) \leftarrow objective(\theta_1, N)$
6: **return** $\hat{c}_N(\theta_1) \leq \hat{c}_N(\theta_2)$

of instances, this approach can find good parameter configurations with low computational effort. However, BasicILS uses a fixed number of N runs to evaluate each configuration θ. Therefore, this strategy is not flexible in general cases. Because if N is large, evaluating a configuration is very expensive and optimization process is very slow. On the contrary, if N is small, ParamILS can suffer a poor generalisation to independent test runs.

3.4 The FocusedILS Algorithm

FocusedILS is a variant of ParamILS that deals with the problems of BasicParamILS by adaptively varying the number of training samples considered from one parameter configuration to another. We denote the number of runs available to estimate the cost statistic $c(\theta)$ for a parameter configuration θ by $N(\theta)$. Having performed different numbers of runs using different parameter configurations, we face the question of comparing two parameter configurations θ and θ' for which $N(\theta) \leq N(\theta')$. One option would be simply to compute the empirical cost statistic based on the available number of runs for each configuration. However, this can lead to systematic biases if, for example, the first instances are easier than the average instance. Instead, we compare θ and θ' based on $N(\theta)$ runs on the same instances and seeds. This approach leads us to a concept of domination and the domination procedure is presented in Procedure 5.

Procedure 6 shows the procedure $better_{Foc}$ used by FocusedParamILS to compare two parameter configurations. This procedure first acquires one additional sample for the configuration i having smaller $N(\theta_i)$, or one run for both configu-

rations if they have the same number of runs. Then, it continues performing runs in this way until one configuration dominates the other. At this point it returns true if θ_1 dominates θ_2, and false otherwise. We also keep track of the total number of configurations evaluated since the last improving step (i.e., since the last time $better_{Foc}$ returned true); we denote this number as B. Whenever $better_{Foc}(\theta_1, \theta_2)$ returns true, we perform B "bonus" runs for θ_1 and reset B to 0. This mechanism ensures that we perform many runs with good configurations, and that the error made in every comparison of two configurations θ_1 and θ_2 decreases on expectation.

Procedure 5 dominates(θ_1, θ_2)

if $N(\theta_1) < N(\theta_2)$ **then**
  return false
end if
return $objective(\theta_1, N(\theta_2)) \leq objective(\theta_2, N(\theta_2))$

3.5 Usage

ParamILS [1, 2] is a tool for parameter optimization. It works for any parameterized algorithm whose parameters can be discretized. ParamILS searches through the space of possible parameter configurations, evaluating configurations by running the algorithm to be optimized on a set of benchmark instances.

What users need to provide for ParamILS are:

- a parametric algorithm A (executable to be called from the command line),
- all parameters and their possible values (parameters need to be configurable from the command line), and
- a set of benchmark problems, S.

Users can also choose from a multitude of optimization objectives, reaching from minimizing average runtime to maximizing median approximation qualities. ParamILS then executes algorithm A with different combinations of parameters on instances sampled from S, searching for the configuration that yields overall best performance across the benchmark problems. For details, see [2].

3.5.1 ParamILS Configuration

There are a number of configurable parameters the user can set:

Procedure 6 $better_{Foc}(\theta_1, \theta_2)$
Procedure used in FocusedILS to compare two parameter configurations. Procedure $objective(\theta, N)$ returns the user-defined objective of the configuration θ on the first N instances, keeps track of the incumbent solution, and updates R_θ (a global cache of algorithm runs performed with parameter configuration θ). For each θ, $N(\theta) = length(R_\theta)$. B is a global counter denoting the number of configurations evaluated since the last improvement step.

Input: Parameter configuration θ_1, parameter configuration θ_2.
Output: True if θ_1 dominates θ_2, false otherwise.
Side Effect: Adds runs to the global caches of performed algorithm runs R_{θ_1} and R_{θ_2} ; updates the global counter B of bonus runs, and potentially the incumbent θ_{inc}.
if $N(\theta_1) \leq N(\theta_2)$ **then**
  $\theta_{min} \leftarrow \theta_1; \theta_{max} \leftarrow \theta_2$
  if $N(\theta_1) = N(\theta_2)$ **then**
    $B \leftarrow B + 1$
  end if
else
  $\theta_{min} \leftarrow \theta_2; \theta_{max} \leftarrow \theta_1$
end if
repeat
  $i \leftarrow N(\theta_{min}) + 1$
  $\hat{c}_i(\theta_{max}) \leftarrow objective(\theta_{max}, i)$ // If $N(\theta_{min}) = N(\theta_{max})$, adds a new run to $R_{\theta_{max}}$.
  $\hat{c}_i(\theta_{min}) \leftarrow objective(\theta_{min}, i)$ // Adds a new run to $R_{\theta_{min}}$.
until $dominates(\theta_1, \theta_2)$ *or* $dominates(\theta_2, \theta_1)$
if $dominates(\theta_1, \theta_2)$ **then**
  // ===== Perform B bonus runs.
  $\hat{c}_{N(\theta_1)+B}(\theta_1) \leftarrow objective(\theta_1, N(\theta_1) + B)$ // Adds B new runs to R_{θ_1}.
  $B \leftarrow 0$
  return true
else
  return false
end if

- **maxEvals** The number of algorithm executions after which the optimization is terminated.
- **maxIts** The number of ILS iterations after which the optimization is terminated.
- **approach** Use basic for BasicILS, focused for FocusedILS, and random for random search.
- **N** For BasicILS, N is the number of runs to perform to evaluate each parameter configuration. For FocusedILS, it is the maximal number of runs to perform to evaluate a parameter configuration.
- **userunlog** If this parameter is 1 or true, another file ending in -runlog will be placed in the output directory. This file will contain the configurations and results for every algorithm run performed by ParamILS. There are also several internal parameters that control the heuristics in ParamILS.

3.5.2 Tuning-scenario file

Tuning-scenario files such as this define a tuning scenario completely, and also contain some information about where ParamILS should write its results, etc. They can contain the following information:

- **algo** An algorithm executable or a call to a wrapper script around an algorithm that conforms with the input/output format of ParamILS.
- **execdir** Directory to execute ⟨algo⟩ from: 'cd ⟨execdir⟩; ⟨algo⟩'
- **deterministic** Set to 0 for randomized algorithms, 1 for deterministic
- **run_obj** A scalar quantifying how 'good' a single algorithm execution is, such as its required runtime. Implemented examples for this include runtime, runlength, approx (approximation quality, i.e., 1-(optimal quality divided by found quality)), speedup (speedup over a reference runtime for this instance - note that for this option the reference needs to be defined in the instance seed file as covered in Section 6). Additional objectives for single algorithm executions can be defined by modifying function single run objective in file algo_specifics.rb.
- **overall_obj** While run obj defines the objective function for a single algorithm run, overall obj defines how those single objectives are combined to reach a single scalar value to compare two parameter configurations. Implemented examples include mean, median, q90 (the 90% quantile), adj_mean

(a version of the mean accounting for unsuccessful runs: total runtime divided by number of succesful runs), mean1000 (another version of the mean accounting for unsuccessful runs: (total runtime of successful runs + 1000x runtime of unsuccessful runs) divided by number of runs - this effectively maximizes the number of successful runs, breaking ties by the runtime of successful runs), and geomean (geometric mean, primarily used in combination with run_obj = speedup. The empirical statistic of the cost distribution (across multiple instances and seeds) to be minimized, such as the mean (of the single run objectives).

- **cutoff_time** The time after which a single algorithm execution will be terminated unsuccesfully. This is an important parameter: if choosen too high, lots of time will be wasted with unsuccessful runs. If chosen too low the optimization is biased to perform well on easy instances only.
- **cutoff_length** The run length after which a single algorithm execution will be terminated unsuccessfully. This length can, e.g. be defined in flips for an SLS algorithm or decisions for a tree search.
- **tunerTimeout** The timeout of the tuner. Validation of the final best found parameter configuration starts after the timeout.
- **paramfile** Specifies the file with the parameters of the algorithms.
- **outdir** Specifies the directory ParamILS should write its results to.
- **instance_file** Specifies the file with a list of training instances.
- **test_instance_file** Specifies the file with a list of test instances.
- **instance_seed_file** Specifies the file with a list of training instance/seed pairs - this and instance file are mutually exclusive.
- **test instance seed file** Specifies the file with a list of training instance/seed pairs - this and test instance file are mutually exclusive.

Part II

CONTRIBUTIONS

Chapter 4

Determine ParamILS Parameters

The experiments in this Section will clarify which are the most suitable ParamILS parameters for MathSAT. The most important ParamILS parameters are:

- **approach** the strategy that ParamILS uses to find the optimized configurations for ParamILS. We can set the *approach* parameter as *Basic*, *Focused*, and *Random* search. Because the *Basic*, and *Focused* search are better than the *Random* search in all experiments of [2], therefore in this thesis we only check whether *Basic* or *Focused* is the fittest one for MathSAT.

- **deterministic** the way ParamILS evaluates the target algorithm (MathSAT, in this case). If *deterministic* is set to 1 (**DAE**), then ParamILS will only evaluate a single ⟨configuration, instance⟩ pair once (with the seed of -1 for the target algorithm, since the seed is not used by a deterministic target algorithm). Otherwise, the *deterministic* parameter is set to 0 (**RAE**), then a ⟨configuration, instance⟩ pair will be evaluated several times, each with a different seed. This is in order to obtain a more representative picture of the algorithm's expected performance on that instance.

- **cutoff-time** the time after which a single algorithm execution will be terminated unsuccessfully. This is an important parameter because: if it is chosen too high, lots of time will be wasted with unsuccessful runs. If it is chosen too low the optimization is biased to perform well on easy instances only.

- **tunerTimeout** the timeout of the tuner (ParamILS). The validation of the final best found parameter configuration starts after this timeout.

- **numRun** the random seed for ParamILS.

4.1 Two runs of Basic ParamILS using DAE

In theory, two runs of ParamILS with the same configuration will return the same result if the MathSAT runtime on the same test are always exactly the same. Because MathSAT is not fully deterministic (it can perform some random steps) and even if MathSAT is deterministic we still cannot guarantee that the measurement of time is always 100% reliable on a general purpose operating system, therefore we cannot avoid having different results of two ParamILS runs using the same configuration. But we still need to check the stability of ParamILS by considering whether the result are nearly the same or completely different. If the first case happens, we will only run ParamILS once on each test case. Otherwise, we will have to run ParamILS many times on each test case and choose the best result. In addition, although ParamILS has two search strategies *Basic* and *Focused*, the ParamILS stability (when fixing other ParamILS parameters) only depends the way ParamILS evaluates each the MathSAT run (DAE and RAE). So, we only need to check the ParamILS stability on the *Basic* approach using *DAE* and *RAE*.

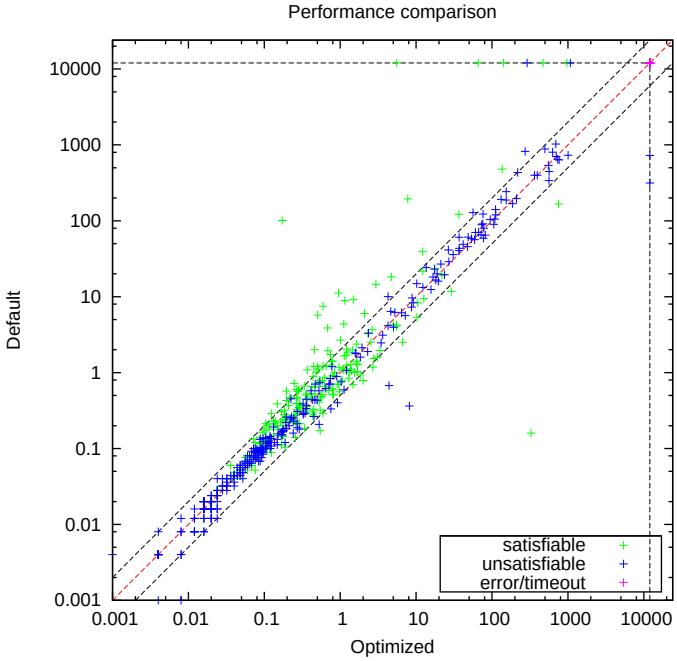
4.1.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Training MathSAT timeout	10s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration 1	Basic, deterministic=1, N=100 run_obj=runtime, overall_obj=mean
ParamILS Configuration 2	Basic, deterministic=1, N=100 run_obj=runtime, overall_obj=mean

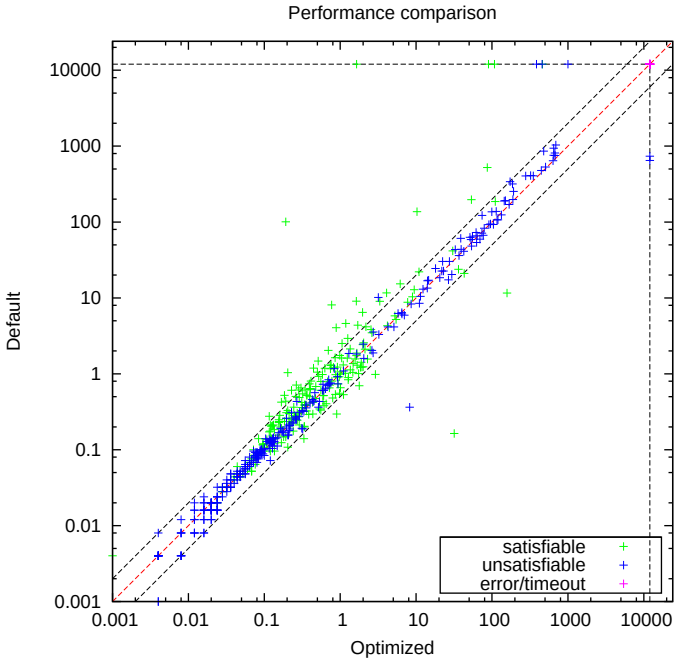
Table 4.1: The experimental setup of Basic ParamILS using DAE

4.1.2 Experimental result

Table 4.2 presents the two configurations found by two runs of ParamILS. Figure 4.1 and Table 4.3 show the MathSAT performance on two configurations found by two ParamILS runs. It can be seen that the two configurations are slightly different, and the MathSAT performance on these configurations are very similar to each other.



(a) The first run



(b) The second run

Figure 4.1: Performance comparison of two runs of Basic ParamILS with deterministic=1

Configuration Parameter	Default	Basic DAE	Basic DAE
aep	yes	yes	yes
deduction	2	2	2
dual_rail	off	off	off
dyn_ack	no	no	yes
dyn_ack_limit	0	0	0
dyn_ack_threshold	1	1	50
expensive_ccmin	yes	no	yes
frequent_reduce_db	no	no	yes
ghost_filter	no	yes	yes
ibliwi	no	yes	yes
impl_expl_threshold	0	0	0
mixed_cs	yes	no	no
permanent_theory_lemmas	yes	yes	yes
pure_literal_filter	no	yes	yes
random_decisions	no	no	no
restarts	normal	adaptive	adaptive
sl	2	2	2
split_eq	no	no	no
tcomb	off	off	ack
toplevelprop	1	0	0
tsolver	euf la	la	euf la

Table 4.2: Experimental result of Basic ParamILS using DAE

Tests solved (Optimized/Default)	**501/496**
Mean runtime(not include TIMEOUT tests)	**30.639/29.202**
Optimized compared with Default	Result
Better runtime tests/The total number of tests	265/543
Equal runtime tests/The total number of tests	92/543
Worse runtime tests/The total number of tests	186/543

(a) The first run

Tests solved (Optimized/Default)	**501/496**
Mean runtime(not include TIMEOUT tests)	**25.978/29.999**
Optimized compared with Default	Result
Better runtime tests/The number of tests	255/543
Equal runtime tests/The number of tests	105/543
Worse runtime tests/The number of tests	183/543

(b) The second run

Table 4.3: Experimental result of Basic ParamILS using DAE

4.2 Two runs of Basic ParamILS using RAE

The experiments in this section accompanied with the experiments in the previous section are used to check the stability of ParamILS by considering whether the results of two ParamILS runs are nearly the same or completely different. If the first case happens, we will only run ParamILS once on each test case. Otherwise, we will have to run ParamILS many times on each test case and choose the best result.

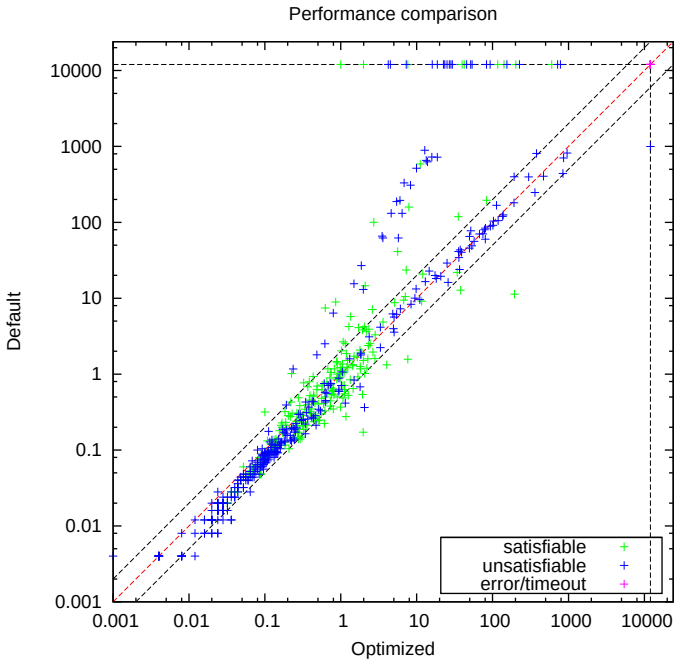
4.2.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Training MathSAT timeout	10s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration 1	Basic, deterministic=0, N=100 run_obj=runtime, overall_obj=mean
ParamILS Configuration 2	Basic, deterministic=0, N=100 run_obj=runtime, overall_obj=mean

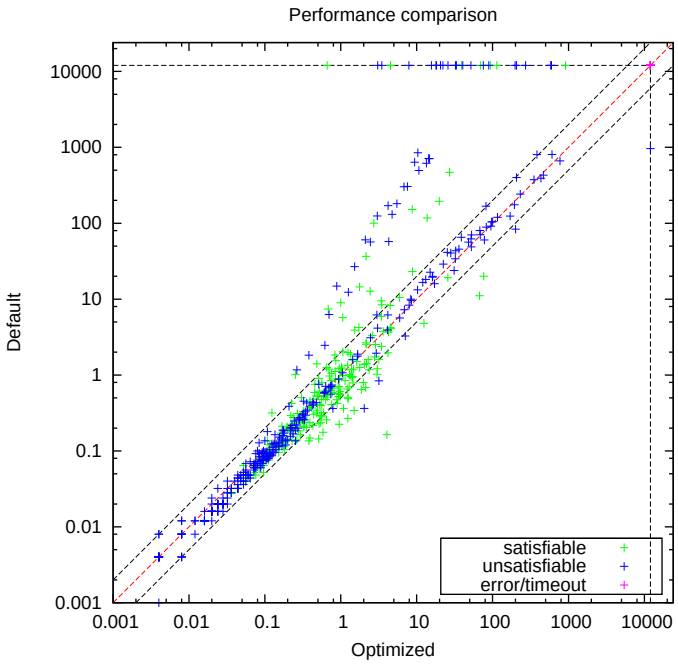
Table 4.4: Experimental setup of two runs of Basic ParamILS using RAE

4.2.2 Experimental result

Table 4.5 presents two configurations found by two ParamILS runs. Figure 4.2 and Table 4.6 show the MathSAT performance on two configurations found by two ParamILS runs. It can be seen that the two configurations are slightly different, and the MathSAT performance on these configurations are very similar to each other.



(a) The first run



(b) The second run

Figure 4.2: Performance comparison of Two runs of Basic ParamILS using RAE

Configuration Parameter	Default	Basic RAE	Basic RAE
aep	yes	no	no
deduction	2	2	2
dual_rail	off	off	off
dyn_ack	no	no	no
dyn_ack_limit	0	0	0
dyn_ack_threshold	1	1	1
expensive_ccmin	yes	yes	yes
frequent_reduce_db	no	no	no
ghost_filter	no	yes	no
ibliwi	no	yes	yes
impl_expl_threshold	0	0	0
mixed_cs	yes	yes	yes
permanent_theory_lemmas	yes	yes	yes
pure_literal_filter	no	yes	no
random_decisions	no	yes	no
restarts	normal	normal	adaptive
sl	2	2	2
split_eq	no	yes	yes
tcomb	off	off	off
toplevelprop	1	0	1
tsolver	euf la	la	la

Table 4.5: Experimental result of two runs of Basic ParamILS using RAE

Tests solved (Optimized/Default)	**524/496**
Mean runtime(not include TIMEOUT tests)	**20.547/29.211**
Optimized compared with Default	Result
Better runtime tests/The number of tests	155/543
Equal runtime tests/The number of tests	36/543
Worse runtime tests/The number of tests	352/543

(a) The first run

Tests solved (Optimized/Default)	**523/496**
Mean runtime(not include TIMEOUT test)	**18.321/28.163**
Optimized compared with Default	Result
Better runtime tests/The number of tests	158/543
Equal runtime tests/The number of tests	55/543
Worse runtime tests/The number of tests	330/543

(b) The second run

Table 4.6: Experimental result of two runs of Basic ParamILS using RAE

4.3 Summary of two Basic ParamILS runs using DAE and RAE

From the summary result in Table 4.7, it can be seen that the MathSAT performances of configurations found by the two ParamILS runs are almost the same. The difference seems not to justify the overhead of running ParamILS several times. Therefore, from now on, we have decided to run ParamILS once on each test case.

	Default	Basic DAE 1	Basic DAE 2	Basic RAE 1	Basic RAE 2
Tests solved	496	501	501	524	523
Mean runtime	28.163	30.639	25.978	20.547	18.321

Table 4.7: The number of solved tests and mean runtime in second (not include TIMEOUT tests) of configurations found by Basic ParamILS using DAE and RAE

4.4 Basic ParamILS using DAE and RAE

The purpose of the experiments in this section is to check which algorithm evaluation (DAE or RAE) Basic ParamILS uses is better for MathSAT.

4.4.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Training MathSAT timeout	10s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration 1	Basic, deterministic=1, N=100 run_obj=runtime, overall_obj=mean
ParamILS Configuration 2	Basic, deterministic=0, N=100 run_obj=runtime, overall_obj=mean

Table 4.8: Experimental setup of Basic ParamILS using DAE and RAE

4.4.2 Experimental result

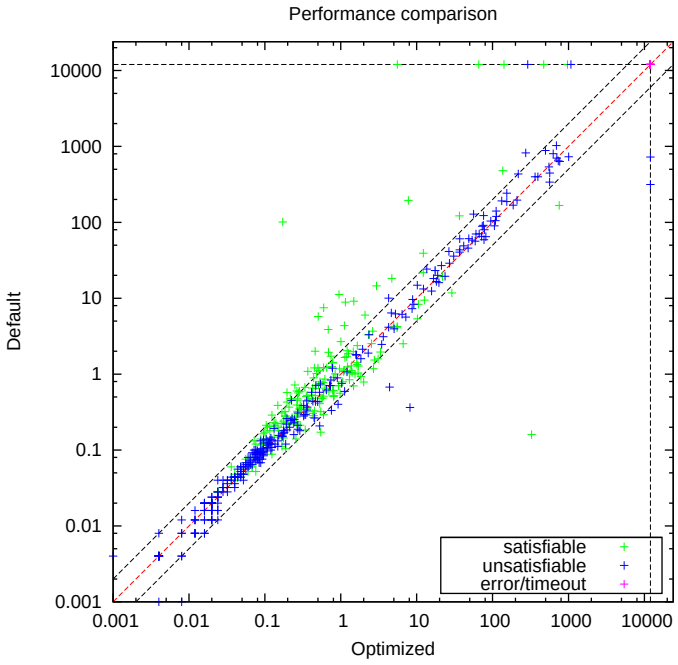
Tests solved (Optimized/Default)	**501/496**
Mean runtime(not include TIMEOUT tests)	**30.639/29.202**
Optimized compared with Default	Result
Better runtime tests/The number of tests	265/543
Equal runtime tests/The number of tests	92/543
Worse runtime tests/The number of tests	186/543

(a) Deterministic

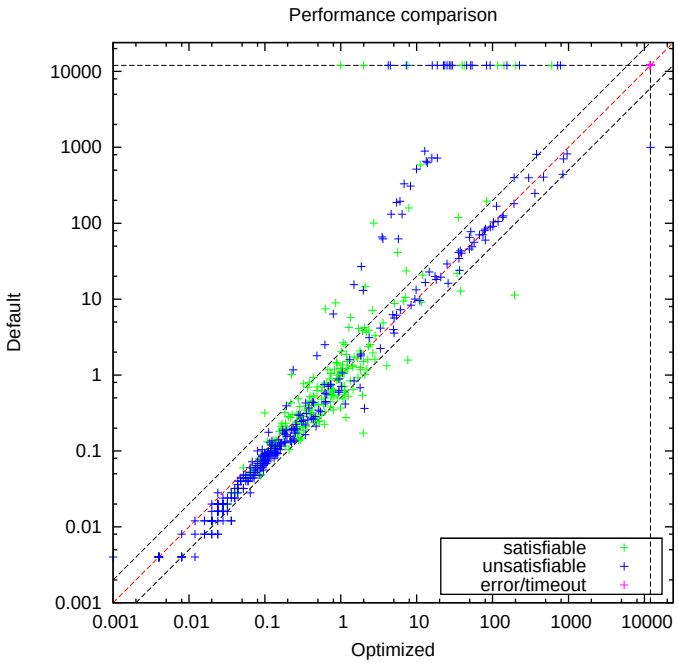
Tests solved (Optimized/Default)	**524/496**
Mean runtime(not include TIMEOUT tests)	**20.547/29.114**
Optimized compared with Default	Result
Better runtime tests/The number of tests	155/543
Equal runtime tests/The number of tests	36/543
Worse runtime tests/The number of tests	352/543

(b) Random

Table 4.9: Experimental result of Basic ParamILS using DAE and RAE



(a) Deterministic



(b) Random

Figure 4.3: Performance comparison of Basic ParamILS using DAE and RAE

The experimental results in Table 4.9 and Figure 4.3 show that Basic ParamILS using RAE is better in this case, although MathSAT is a deterministic algorithm. The main reason is that Basic ParamILS evaluates every configuration only once with the same number of MathSAT runs, and the MathSAT run time of the same instance on general purpose operating systems can vary in different runs. Therefore, RAE is more robust than DAE in the case of Basic ParamILS because RAE evaluates a single ⟨configuration, instance⟩ pair many times, each with a different seed (this helps to obtain a more representative picture of the algorithm's expected performance on that instance).

4.5 Focused ParamILS with DAE and RAE

The purpose of the experiments in this section is to check which algorithm evaluation (DAE or RAE) Focused ParamILS uses is better for MathSAT.

4.5.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Training MathSAT timeout	10s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration 1	Focused, deterministic=1, N=543 run_obj=runtime, overall_obj=mean
ParamILS Configuration 2	Focused, deterministic=0, N=543 run_obj=runtime, overall_obj=mean

Table 4.10: The experimental setup of Focused ParamILS with deterministic and random

4.5.2 Experimental result

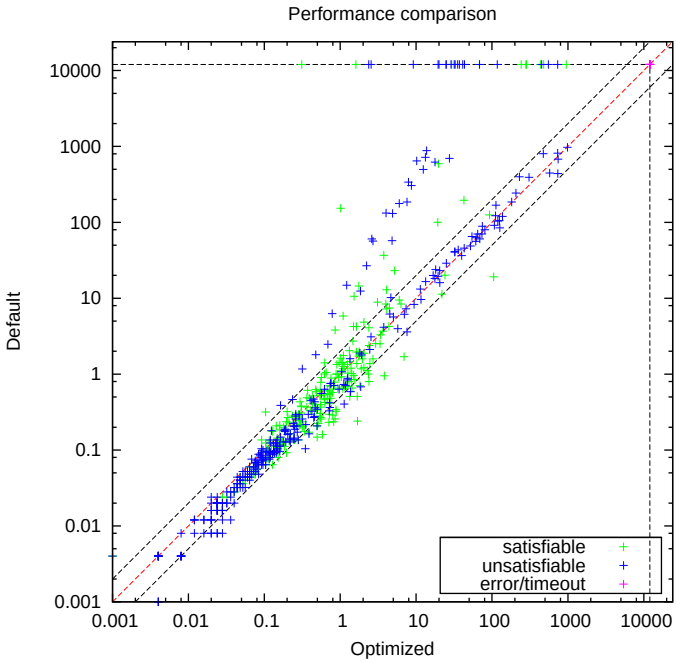
Tests solved (Optimized/Default)	**525/496**
Mean runtime(not include TIMEOUT tests)	**24.367/28.903**
Optimized compared with Default	Result
Better runtime tests/The number of tests	148/543
Equal runtime tests/The number of tests	40/543
Worse runtime tests/The number of tests	355/543

(a) Deterministic

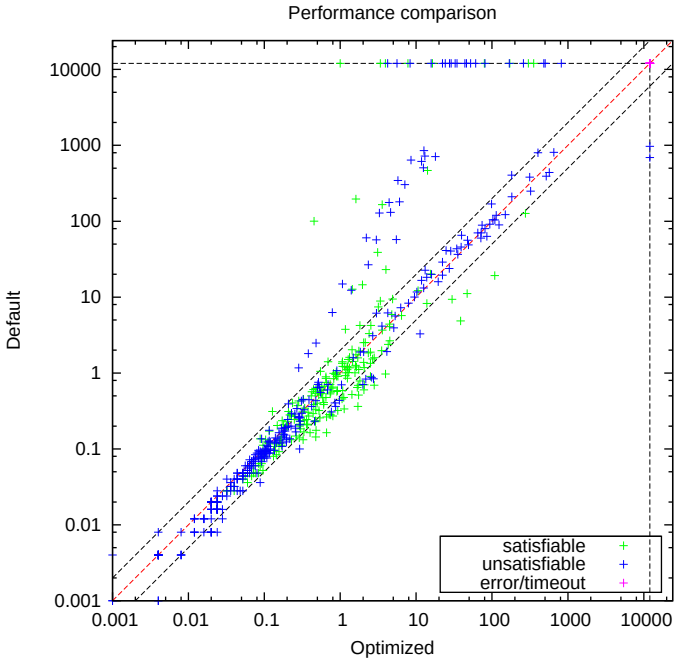
Tests solved (Optimized/Default)	**523/496**
Mean runtime(not include TIMEOUT test)	**18.150/28.593**
Optimized compared with Default	Result
Better runtime tests/The number of tests	165/543
Equal runtime tests/The number of tests	56/543
Worse runtime tests/The number of tests	322/543

(b) Random

Table 4.11: Experimental result of Focused ParamILS using DAE and RAE



(a) Deterministic



(b) Random

Figure 4.4: Performance comparison of Focused ParamILS using DAE and RAE

The experimental results in Table 4.11 and Figure 4.4 show that Focused ParamILS using DAE is better. This is because although using DAE, a MathSAT run is evaluated only once, Focused ParamILS evaluates each configuration many times, each with a different number of MathSAT runs. In other words, evaluating a MathSAT run once still guarantees the configuration evaluation is precise by evaluating that configuration many times. Besides, DAE saves a lot of time when evaluating a MathSAT run compared with RAE, and this makes Focused ParamILS evaluate a configuration more precisely by testing that configuration on a large number of instances.

4.6 Summary of Basic and Focused ParamILS using DAE and RAE

From the results in Table 4.12, it can be seen that Focused ParamILS is more stable than Basic ParamILS, and Focused ParamILS using DAE provides us with the best performance. Therefore, Focused ParamILS using DAE is chosen for running on other experiments in this thesis.

	Default	Basic DAE	Basic RAE	Focused DAE	Focused RAE
Tests solved	496	501	524	525	523

Table 4.12: The number of tests solved (on 543 tests) of configurations found by Basic and Focused ParamILS using DAE and RAE

4.7 RAE Basic ParamILS with different MathSAT timeouts

In order to choose the suitable MathSAT timeout (cutoff_time), we base on the percentage of tests solved by the default (or smt-comp) configuration using that MathSAT timeout. The experiments in this section and the next section are used to check whether ParamILS configured with the MathSAT timeouts in which the default configuration solves similar percentage of tests will have similar results.

4.7.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration	Basic, deterministic=0, N=100 run_obj=runtime, overall_obj=mean
Training MathSAT timeout	5s, 10s, 20s, 40s, 60s

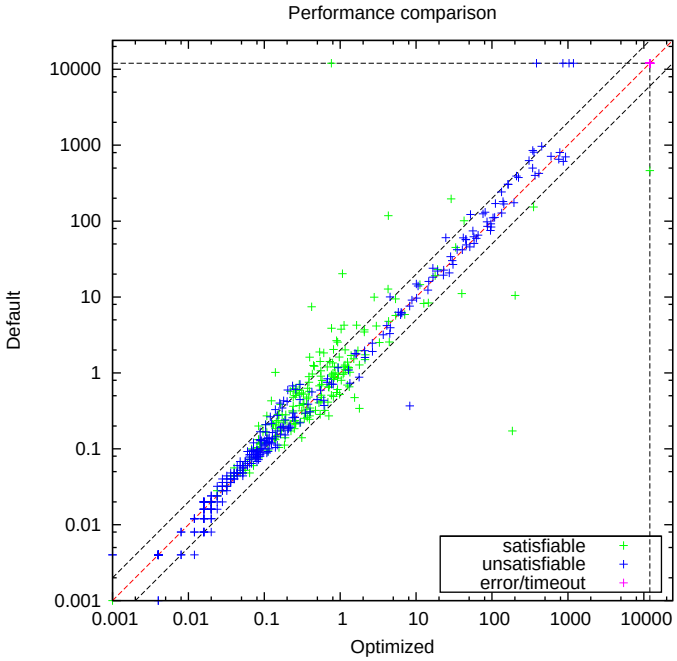
Table 4.13: The experimental setup of Basic ParamILS with different MathSAT timeouts

MathSAT timeout	5s	10s	20s	40s	60s
Percentage of tests solved by the default configuration	79.7%	82.5%	85.6%	88.8%	90.4%

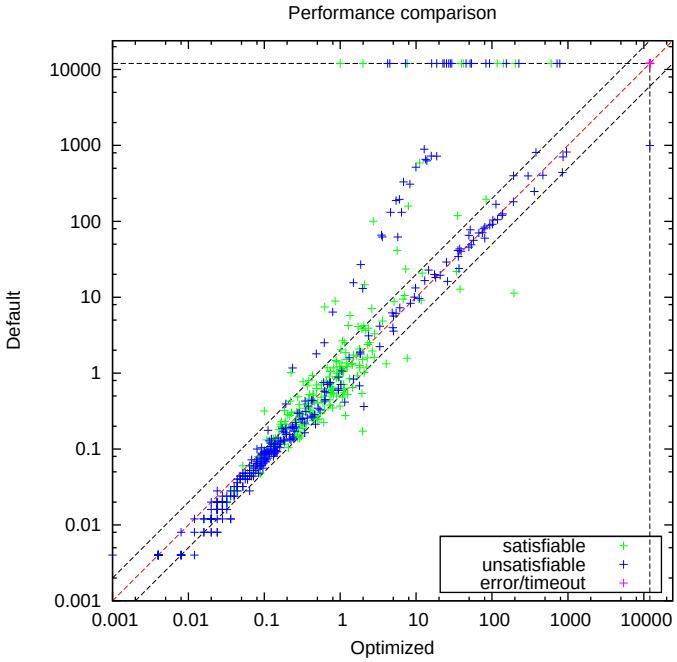
Table 4.14: The percentage of tests solved when running MathSAT with different timeouts on the training set

4.7.2 Experimental result

From the results in Table 4.15 and Figure 4.5, except for timeout=5s, other MathSAT timeouts have similar performance because the target algorithm MathSAT (using the default configuration) configured with these training timeouts solve approximately the same percentage of instances in the training set.

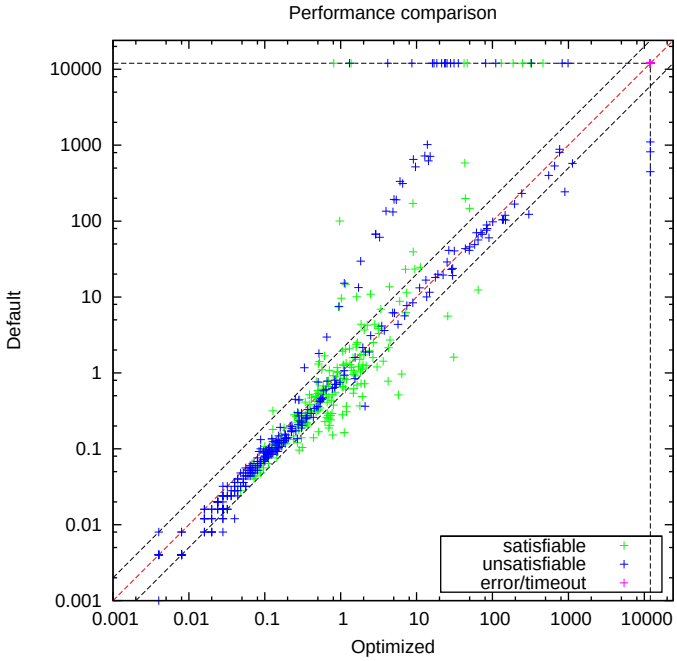


(a) training MathSAT timeout=5s

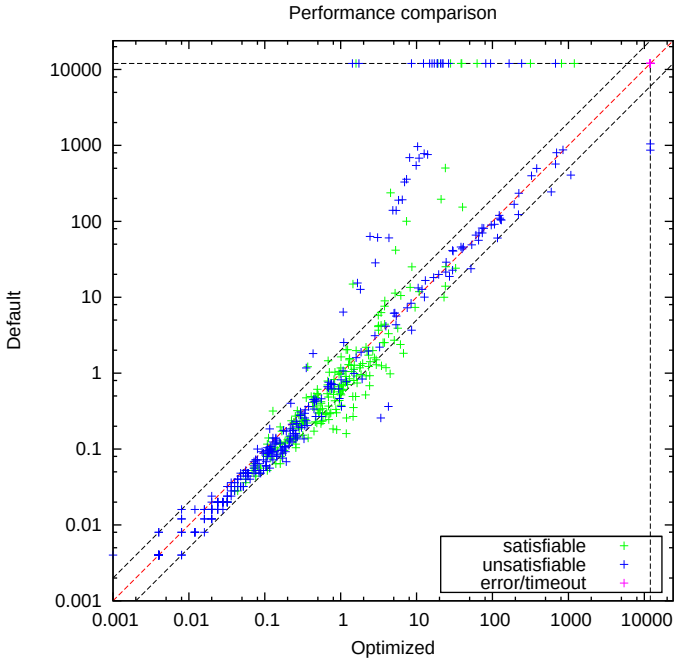


(b) training MathSAT timeout=10s

Figure 4.5: Performance comparison of Basic ParamILS with different training MathSAT timeouts

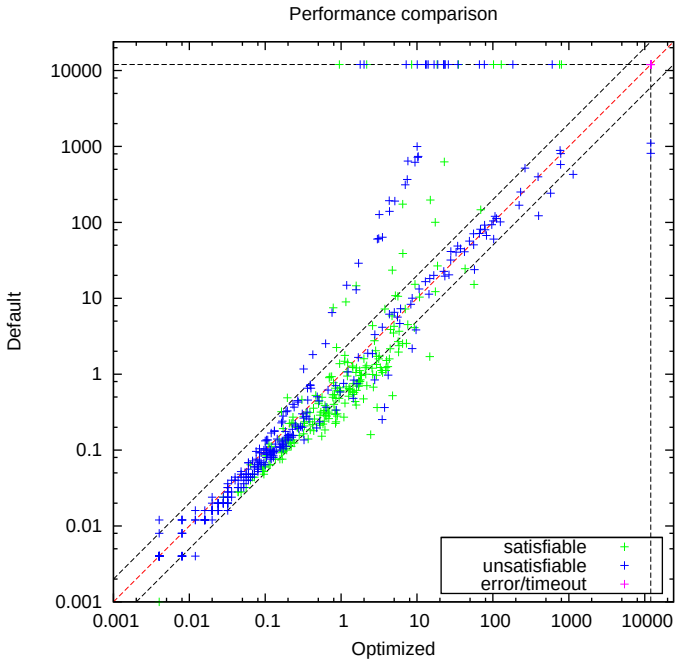


(c) training MathSAT timeout=20s



(d) training MathSAT timeout=40s

Figure 4.5: Performance comparison of Basic ParamILS with different training MathSAT timeouts



(e) training MathSAT timeout=60s

Figure 4.5: Performance comparison of Basic ParamILS with different training MathSAT timeouts

Tests solved (Optimized/Default)	**500/496**
Mean runtime(not include TIMEOUT tests)	**29.194/28.274**
Optimized compared with Default	Result
Better runtime tests/The number of tests	280/543
Equal runtime tests/The number of tests	97/543
Worse runtime tests/The number of tests	166/543

(a) training MathSAT timeout=5s

Tests solved (Optimized/Default)	**524/496**
Mean runtime(not include TIMEOUT tests)	**20.547/29.211**
Optimized compared with Default	Result
Better runtime tests/The number of tests	155/543
Equal runtime tests/The number of tests	36/543
Worse runtime tests/The number of tests	352/543

(b) training MathSAT timeout=10s

Tests solved (Optimized/Default)	**520/496**
Mean runtime(not include TIMEOUT tests)	**23.137/30.939**
Optimized compared with Default	Result
Better runtime tests/The number of tests	123/543
Equal runtime tests/The number of tests	34/543
Worse runtime tests/The number of tests	386/543

(c) training MathSAT timeout=20s

Tests solved (Optimized/Default)	**521/496**
Mean runtime(not include TIMEOUT tests)	**22.308/31.188**
Optimized compared with Default	Result
Better runtime tests/The number of tests	120/543
Equal runtime tests/The number of tests	49/543
Worse runtime tests/The number of tests	374/543

(d) training MathSAT timeout=40s

Tests solved (Optimized/Default)	**521/495**
Mean runtime(not include TIMEOUT tests)	**21.072/30.121**
Optimized compared with Default	Result
Better runtime tests/The number of tests	132/543
Equal runtime tests/The number of tests	47/543
Worse runtime tests/The number of tests	364/543

(e) training MathSAT timeout=60s

Table 4.15: Experimental result of RAE Basic ParamILS with different training MathSAT timeouts

4.8 DAE Focused ParamILS with different training MathSAT timeouts

In order to choose the suitable MathSAT timeout (cuttoff_time), we base on the percentage of tests solved by the default (or smt-comp) configuration using that MathSAT timeout. The experiments in this section and the previous section are used to check whether ParamILS configured with the MathSAT timeouts in which the default configuration solves similar percentage of tests will have similar results.

4.8.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training time	48 hours
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=543 run_obj=runtime, overall_obj=mean
Training MathSAT timeout	5s, 10s, 20s, 40s, 60s

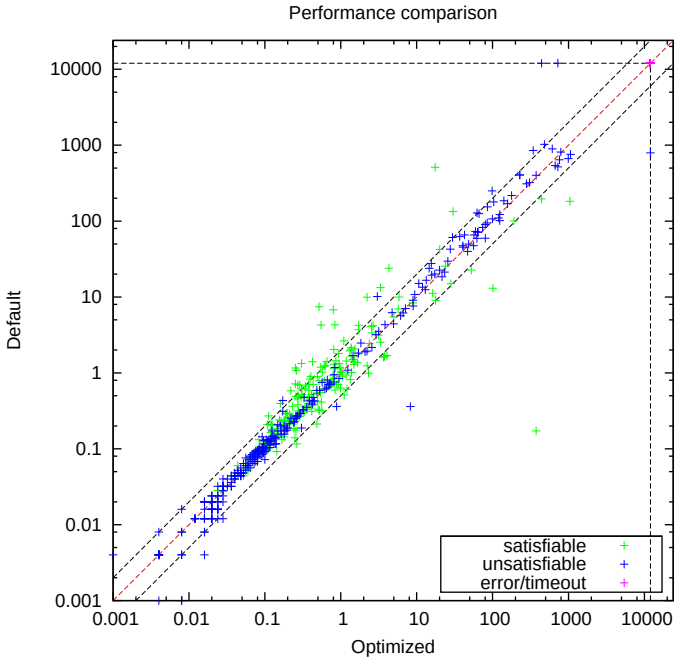
Table 4.16: The experimental setup of DAE Focused ParamILS with different timeouts

MathSAT timeout	5s	10s	20s	40s	60s
Percentage os tests solved by the default configuration	79.7%	82.5%	85.6%	88.8%	90.4%

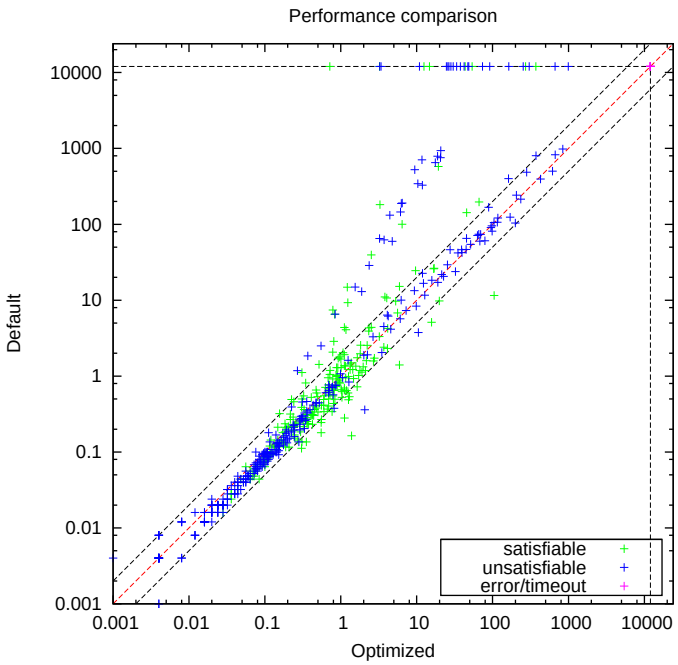
Table 4.17: The percentage of tests solved and mean runtime when running MathSAT with different MathSAT timeouts on the training set

4.8.2 Experimental result

From the results in Table 4.18 and Figure 4.6, except for training MathSAT timeout=5s, other training MathSAT timeouts have similar performance because the target algorithm MathSAT (using the default configuration) configured with these training MathSAT timeouts solve approximately the same percentage of instances in the training set.

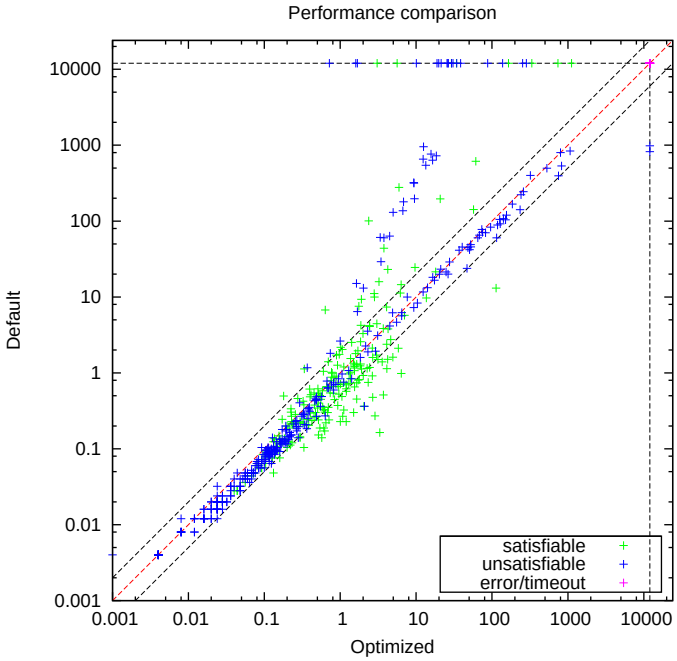


(a) timeout=5s

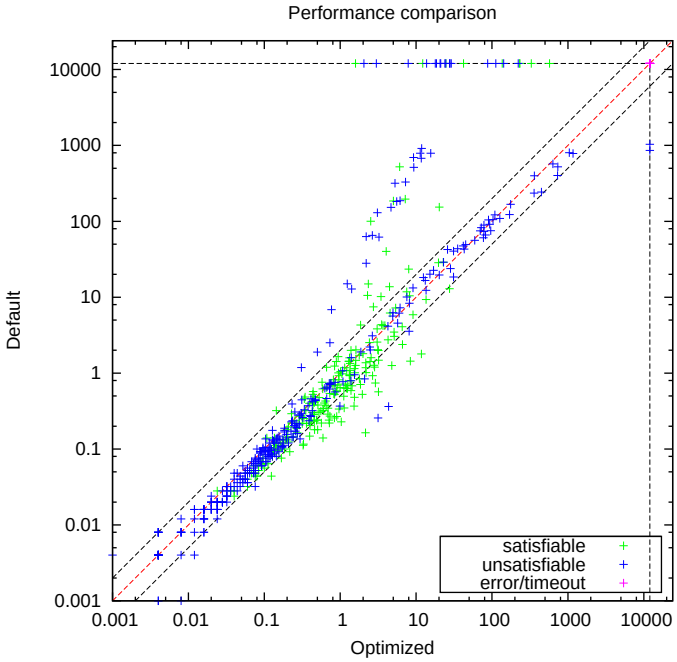


(b) timeout=10s

Figure 4.6: Performance comparison of DAE Focused ParamILS with different MathSAT timeouts

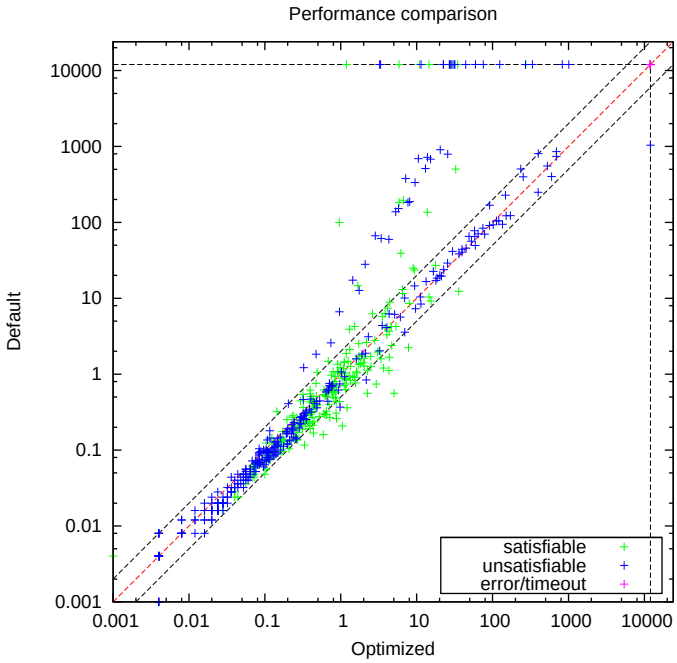


(c) timeout=20s



(d) timeout=40s

Figure 4.6: Performance comparison of DAE Focused ParamILS with different MathSAT timeouts



(e) timeout=60s

Figure 4.6: Performance comparison of DAE Focused ParamILS with different MathSAT timeouts

Tests solved (Optimized/Default)	**496/495**
Mean runtime(not include TIMEOUT tests)	**28.651s/28.633s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	275/543
Equal runtime tests/The number of tests	102/543
Worse runtime tests/The number of tests	166/543

(a) timeout=5s

Tests solved (Optimized/Default)	**523/495**
Mean runtime(not include TIMEOUT tests)	**19.407s/28.953s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	182/543
Equal runtime tests/The number of tests	50/543
Worse runtime tests/The number of tests	311/543

(b) timeout=10s

Tests solved (Optimized/Default)	**519/495**
Mean runtime(not include TIMEOUT tests)	**21.957s/30.743s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	129/543
Equal runtime tests/The number of tests	56/543
Worse runtime tests/The number of tests	358/543

(c) timeout=20s

Tests solved (Optimized/Default)	**520/496**
Mean runtime(not include TIMEOUT tests)	**19.954s/30.789s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	155/543
Equal runtime tests/The number of tests	59/543
Worse runtime tests/The number of tests	329/543

(d) timeout=40s

Tests solved (Optimized/Default)	**522/496**
Mean runtime(not include TIMEOUT tests)	**18.814s/30.496s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	166/543
Equal runtime tests/The number of tests	50/543
Worse runtime tests/The number of tests	327/543

(e) timeout=60s

Table 4.18: Performance comparison of DAE Focused ParamILS with different training MathSAT timeouts

4.9 DAE Focused ParamILS with different training times

ParamILS configured with the MathSAT timeout of 60s converges within 48 hours (by considering the ParamILS log file). The experiments in this section are used to check whether we can get the over-training situation by expanding the training time to 72 hours.

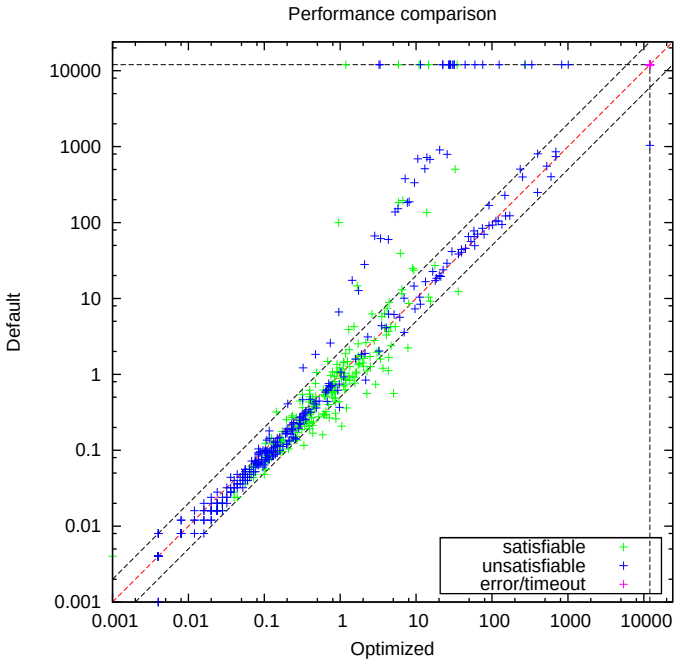
4.9.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Training MathSAT timeout	60s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=543 run_obj=runtime, overall_obj=mean
Training time 1	48 hours
Training time 2	72 hours

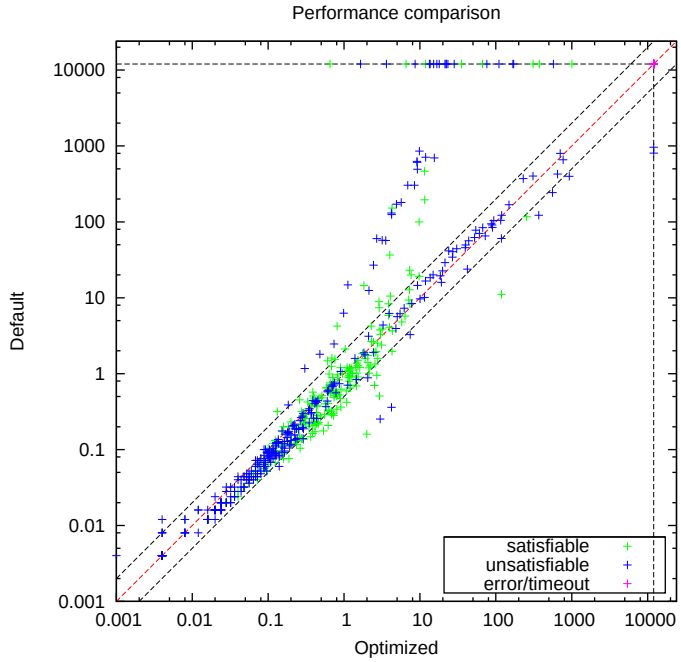
Table 4.19: The experimental setup of DAE Focused ParamILS with different training times

4.9.2 Experimental result

The results in Table 4.20 and Figure 4.7 show that we can get the over-training situation. Therefore, if the MathSAT timeout is less than or equal to 60s, we should not train ParamILS more than 48 hours.



(a) Training time = 48hours



(b) Training time = 72hours

Figure 4.7: Performance comparison of Focused ParamILS, deterministic=1 with different training times

Tests solved (Optimized/Default)	**522/496**
Mean runtime(not include TIMEOUT tests)	**18.814s/30.496s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	166/543
Equal runtime tests/The number of tests	50/543
Worse runtime tests/The number of tests	327/543

(a) Training time = 48hours

Tests solved (Optimized/Default)	**519/495**
Mean runtime(not include TIMEOUT tests)	**19.442s/27.850s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	167/543
Equal runtime tests/The number of tests	51/543
Worse runtime tests/The number of tests	325/543

(b) Training time = 72 hours

Table 4.20: Performance comparison of DAE Focused ParamILS with different training times

4.10 DAE Focused ParamILS with different numRuns

Because the ParamILS random behaviour depends on a random seed, *numRun*, therefore we need to check whether ParamILS performances with different random seeds are completely different or similar to each other. The experiments in this section are used to verify this property.

4.10.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.5-itp (Jun 24 2009 13:15:40, gmp 4.2.2, gcc 3.4.6)
Tunning time	48 hours
Training MathSAT timeout	10s
Testing MathSAT timeout	1200s
Training set	the QF_LRA of SMT LIB
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=543 run_obj=runtime, overall_obj=mean
numRun	1, 2, 3

Table 4.21: Experimental setup of DAE Focused ParamILS with different numRuns

4.10.2 Experimental result

The results in Table 4.22 and Figure 4.8 show that ParamILS has nearly the same performance on different numRuns. Hence, we fix the numRun parameter for other experiments in this thesis.

Tests solved (Optimized/Default)	**523/496**
Mean runtime(not include TIMEOUT tests)	**22.373s/30.867s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	180/543
Equal runtime tests/The number of tests	48/543
Worse runtime tests/The number of tests	315/543

(a) numRun 1

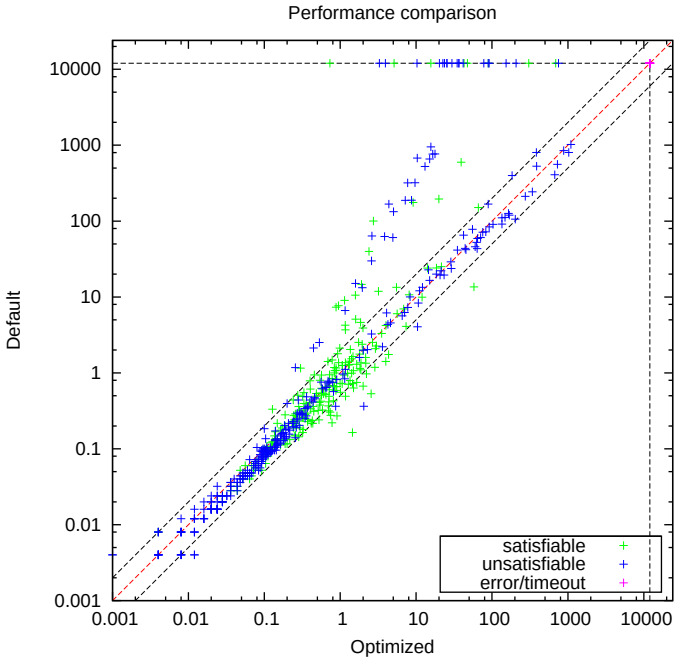
Tests solved (Optimized/Default)	**523/496**
Mean runtime(not include TIMEOUT tests)	**18.165s/28.200s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	186/543
Equal runtime tests/The number of tests	52/543
Worse runtime tests/The number of tests	305/543

(b) numRun 2

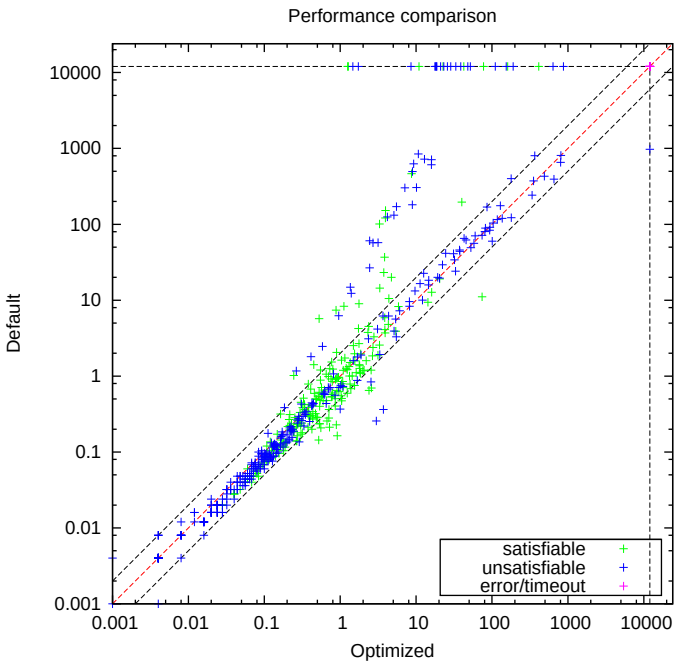
Tests solved (Optimized/Default)	**523/495**
Mean runtime(not include TIMEOUT tests)	**19.407s/28.953s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	182/543
Equal runtime tests/The number of tests	50/543
Worse runtime tests/The number of tests	311/543

(c) numRun 3

Table 4.22: Performance comparison of DAE Focused ParamILS with different numRuns

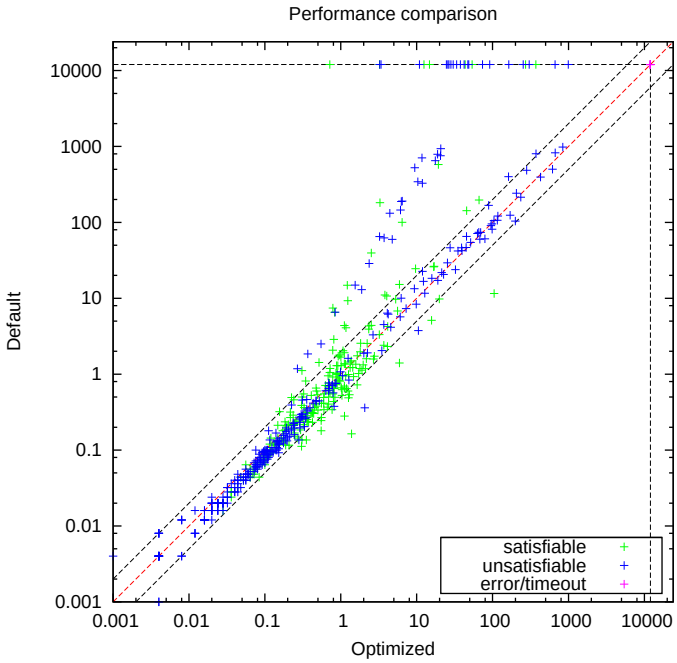


(a) numRun 1



(b) numRun 2

Figure 4.8: Performance comparison of DAE Focused ParamILS with different numRuns



(c) numRun 3

Figure 4.8: Performance comparison of DAE Focused ParamILS with different numRuns

4.11 Summary

From the experimental results in this section, we conclude that *Focused ParamILS using DAE* is the most suitable configuration of ParamILS for MathSAT. As for the training time, *48 hours* is enough for the ParamILS convergence because we experimented on the *QF_LRA*, one of the most difficult theories in the set of theories supported by MathSAT, and observed that ParamILS converged within *48 training hours* with the MathSAT timeouts less than 60s. In order to choose suitable MathSAT timeouts when training, we run MathSAT with the default (or smtcomp) configuration on the training dataset in advance and choose the three MathSAT timeouts (all timeouts have value from 0 to 1200 with step of 5) which are less than 60 seconds and solve similar percentage of tests.

Chapter 5

Configuring MathSAT on Five Theories on the SMT-COMP Benchmark

This section shows the results of using ParamILS to find the best possible configurations for twelve theories implemented by MathSAT on the SMT-COMP 2009 benchmark. The experiments in this section use the same dataset of SMT-COMP 2009 for training and testing because we want to get the best possible MathSAT configurations for the SMT-COMP 2009. In addition, MathSAT is under heavy development, therefore these experiments help a lot in revealing MathSAT bugs.

The experiment setup for all tests in this chapter is summarised as follows: Based on the experimental results in Chapter 4, Focused ParamILS using DAE is the best strategy for MathSAT. In addition, the training time of 48 hours is enough for MathSAT to converge with the MathSAT timeouts less than 60 seconds. As for choosing the training MathSAT timeouts, we run MathSAT with the smt-comp configuration on the training dataset in advance and choose the three MathSAT timeouts (all timeouts have value from 0 to 1200 with step 5) which are less than 60 seconds and solve similar percentage of tests.

Notice that the smt-comp configurations are *adaptive*, i.e. they can be changed according to different problem classes based on a statistics module in MathSAT.

Before moving to the next sections of experiments, we introduce briefly the five theories [17] :

- **QF_IDL**: Difference Logic over the integers. In essence, Boolean combinations of inequations of the form $(x - y < b)$ where x and y are integer variables and b is an integer constant.

- **QF_LIA**: Unquantified linear integer arithmetic. In essence, Boolean combinations of inequations between linear polynomials over integer variables.

These inequations are written in the form $(a_1.x_1 + ... + a_n.x_n \bowtie a_0)$, s.t. $\bowtie \in \{\leq, <, \neq, =, \geq, >\}$.

- **QF_LRA**: Unquantified linear real arithmetic. In essence, Boolean combinations of inequations between linear polynomials over real variables.

- **QF_UFIDL**: Difference Logic over the integers (in essence) but with uninterpreted sort and function symbols (QF_UF is described in chapter 6). For example:

$$UF : (f(x_1) = f(x_2)) \wedge \neg(f(x_2) = f(x_3))$$
$$IDL : (x_1 - x_3 < 1) \wedge (x_3 - x_1 < 1)$$

- **QF_UFLRA**: Unquantified linear real arithmetic with uninterpreted sort and function symbols.

5.1 QF_IDL with Focused ParamILS using DAE

5.1.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	35s (the smt-comp configuration solved 75.49% of tests)
Training MathSAT timeout 2	40s (the smt-comp configuration solved 76.47% of tests)
Training MathSAT timeout 3	45s (the smt-comp configuration solved 76.47% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_IDL of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=102 run_obj=runtime, overall_obj=mean

Table 5.1: The experimental setup of QF_IDL with different training MathSAT timeouts

5.1.2 Training Result

Table 5.2 shows the default configuration and the optimized configurations found by running ParamILS with different MathSAT timeouts. The **bold font** is used to show the difference between configurations. It can be seen that for QF_IDL, *ghost_filter* and *incr_solvers* should be enabled, while *mixed_cs* and *split_eq* should be disabled. Besides, the *deduction* level should be 2 instead of 0 as default and the *static learning* (sl) level should be 1 instead of 2 as default.

5.1.3 Testing Result

Table 5.3 and Figure 5.1 show the performance comparison between the optimized, default and adaptive smt-comp configurations. The optimized configurations solve *14 tests*, 12 tests, and 13 tests more compared with the adaptive smt-comp configuration. Moreover, the winner on this theory in the SMT competition 2009 can solve only *86 tests* but the best optimized configuration can solve *96 tests* (despite the fact that we are using the older version of MathSAT in this experiment, not the one in the SMT competition 2009). The result is a significant improvement because the smt-comp configuration is an ***adaptive*** configuration

Configuration Parameter	Default	35s	40s	45s
aep	yes	yes	yes	yes
deduction	**2**	**0**	**0**	**0**
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	yes	no	no
frequent_reduce_db	no	no	no	no
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	yes	yes	yes	yes
impl_expl_threshold	0	1	1	0
incr_tsolvers	**no**	**yes**	**yes**	**yes**
mixed_cs	**yes**	**no**	**no**	**no**
permanent_theory_lemmas	yes	yes	no	yes
pure_literal_filter	no	no	no	no
random_decisions	no	no	no	no
restarts	normal	adaptive	normal	normal
sl	**2**	**1**	**1**	**1**
split_eq	**yes**	**no**	**no**	**no**
tcomb	off	off	off	off
toplevelprop	1	2	1	1
tsolver	dl	dl	dl	dl

Table 5.2: QF_IDL configurations found by different training MathSAT timeouts

which is the combination of many a prior chosen configurations and changed according to different problem classes while the optimized configuration is ***fixed*** for all tests.

Tests solved (Optimized/Default/SMTCOMP)	**96/81/82**
Mean runtime (not include TIMEOUT tests)	**14.755s/25.069s/40.349s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	63/102
Equal runtime tests/The number of tests	21/102
Worse runtime tests/The number of tests	18/102

(a) training MathSAT timeout = 35s

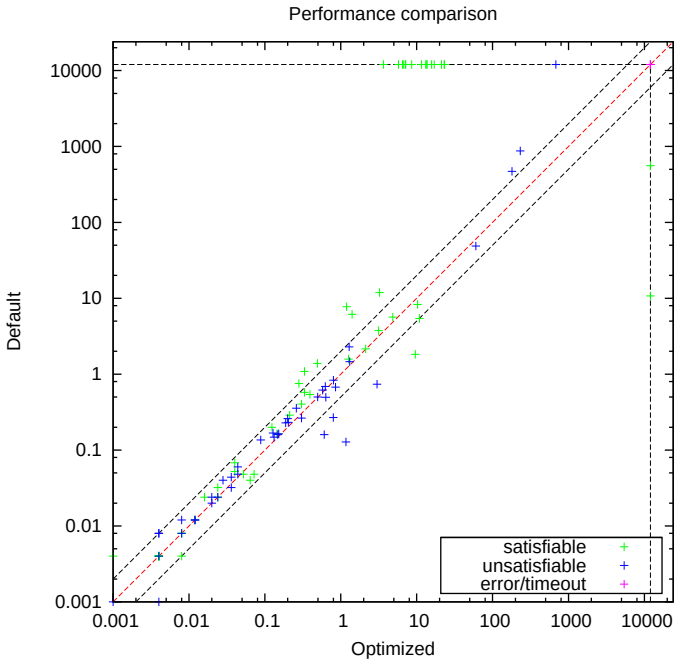
Tests solved (Optimized/Default/SMTCOMP)	**94/81/82**
Mean runtime(not include TIMEOUT tests)	**22.805s/25.069s/40.349s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	54/102
Equal runtime tests/The number of tests	30/102
Worse runtime tests/The number of tests	18/102

(b) training MathSAT timeout = 40s

Tests solved (Optimized/Default/SMTCOMP)	**95/81/82**
Mean runtime(not include TIMEOUT tests)	**13.165s/25.069s/40.349s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	58/102
Equal runtime tests/The number of tests	28/102
Worse runtime tests/The number of tests	16/102

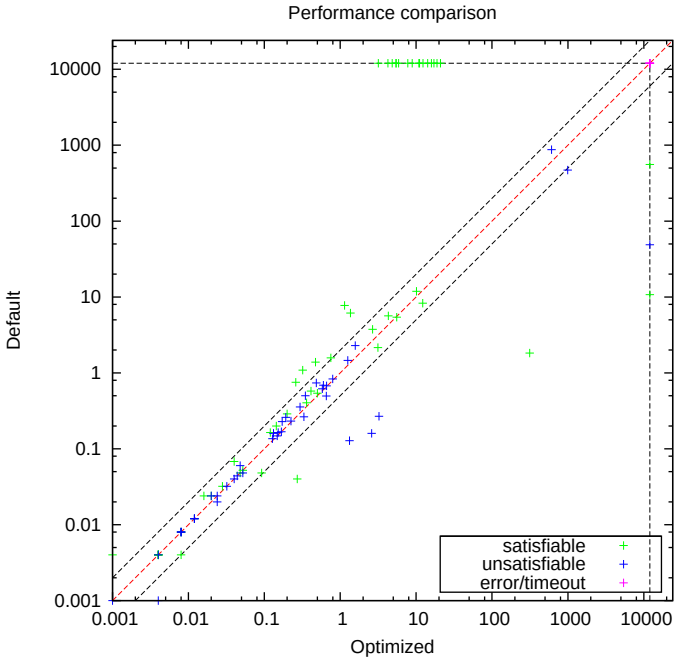
(c) training MathSAT timeout = 45s

Table 5.3: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts

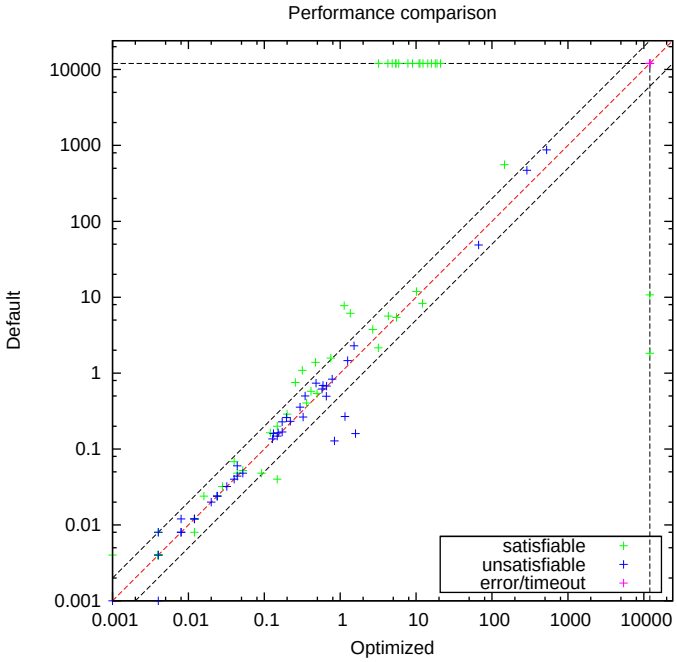


(a) training MathSAT timeout = 35s

Figure 5.1: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 40s



(c) training MathSAT timeout = 45s

Figure 5.1: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts (cont.)

5.2 QF_LIA with Focused ParamILS using DAE

5.2.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 83.41% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 83.90% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 85.58% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_LIA of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=205 run_obj=runtime, overall_obj=mean

Table 5.4: The experimental setup of QF_LIA with different training MathSAT timeouts

5.2.2 Training Result

Table 5.5 shows the default configuration and the optimized configurations found by ParamILS. For this theory, ParamILS suggests that *aep*, *pure_literal_filter*, and *split_eq* should be disabled while *ghost_filter*, *ibliwi*, and *random_decisions* should be enabled. In addition, the *static learning* level (sl) should be 0 instead of 2 as default and using one theory solver *la* is better to solve this theory than using two theories *euf, and la* as default.

5.2.3 Testing Result

Table 5.6 and Figure 5.2 show the performance comparison between the optimized, default and adaptive smt-comp configuration. In three optimized configurations, the optimized configuration found by the MathSAT timeout of 60 seconds has the mean runtime *reduced by half* compared with the mean runtime of the *adaptive* smt-comp configuration.

Configuration Parameter	Default	50s	55s	60s
aep	**yes**	**no**	**no**	**no**
deduction	2	2	2	2
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	no	no	yes
frequent_reduce_db	no	no	yes	yes
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	**no**	**yes**	**yes**	**yes**
impl_expl_threshold	0	0	0	0
incr_tsolvers	**no**	**yes**	**yes**	**yes**
mixed_cs	yes	yes	no	no
permanent_theory_lemmas	yes	yes	yes	no
pure_literal_filter	**yes**	**no**	**no**	**no**
random_decisions	**no**	**yes**	**yes**	**yes**
restarts	normal	normal	quick	quick
sl	**2**	**0**	**0**	**0**
split_eq	**yes**	**no**	**no**	**no**
tcomb	off	off	off	off
toplevelprop	1	0	0	1
tsolver	**euf laz**	**laz**	**laz**	**laz**

Table 5.5: QF_LIA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**201/201/202**
Mean runtime (not include TIMEOUT tests)	**11.892s/15.495s/26.322s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	165/205
Equal runtime tests/The number of tests	12/205
Worse runtime tests/The number of tests	28/205

(a) training MathSAT timeout = 50s

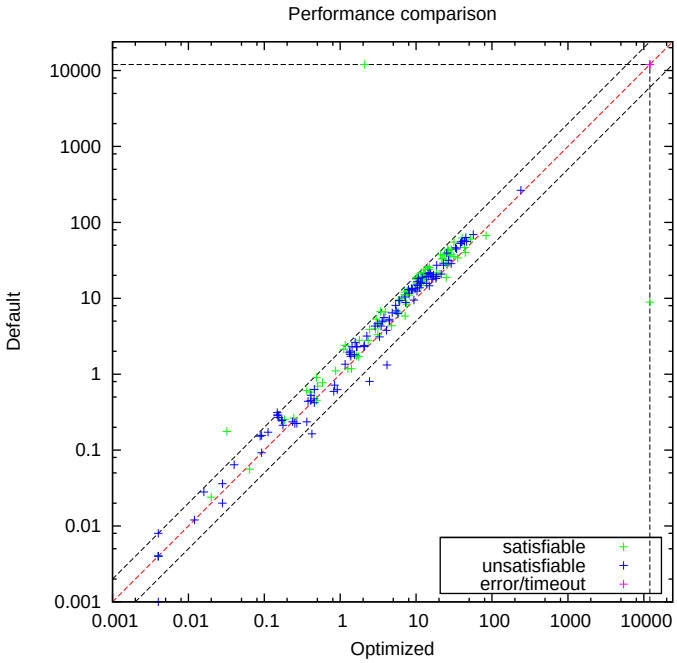
Tests solved (Optimized/Default/SMTCOMP)	**201/201/202**
Mean runtime(not include TIMEOUT tests)	**13.377s/15.495s/26.322s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	165/205
Equal runtime tests/The number of tests	8/205
Worse runtime tests/The number of tests	32/205

(b) training MathSAT timeout = 55s

Tests solved (Optimized/Default/SMTCOMP)	**202/201/202**
Mean runtime(not include TIMEOUT tests)	**13.850s/15.495s/26.322s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	164/202
Equal runtime tests/The number of tests	9/202
Worse runtime tests/The number of tests	32/202

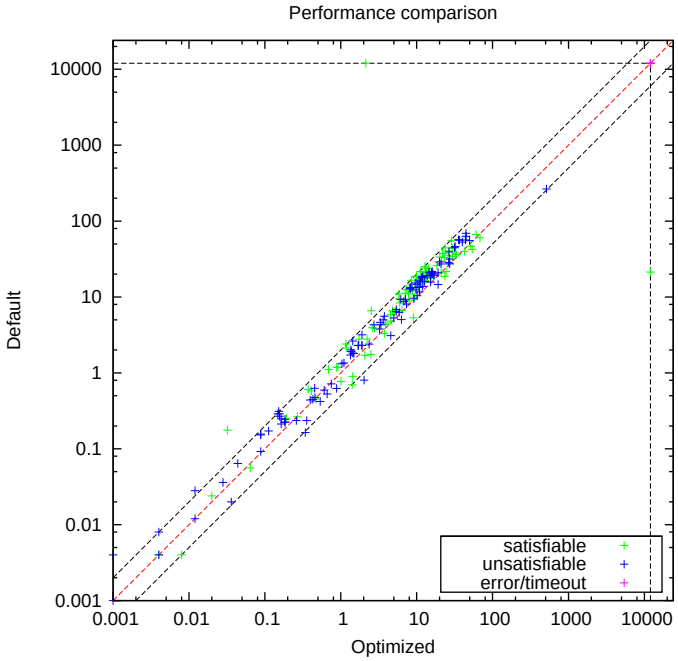
(c) training MathSAT timeout = 60s

Table 5.6: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts

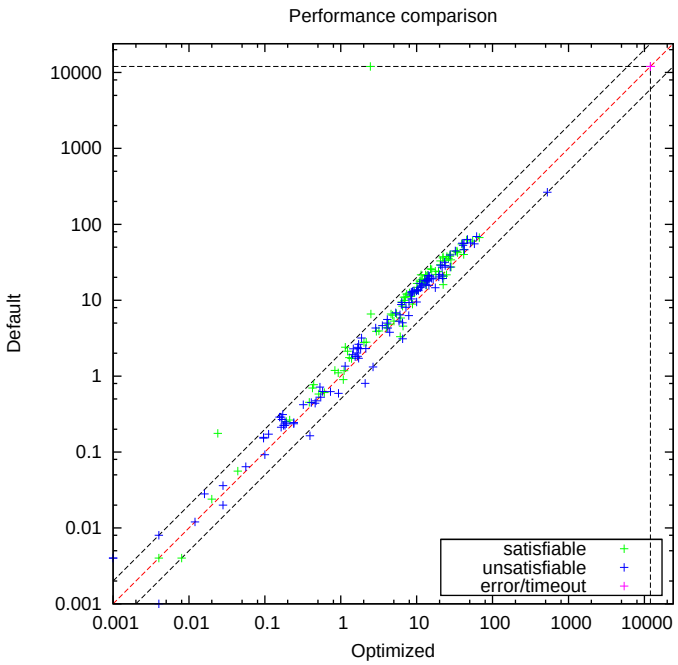


(a) training MathSAT timeout = 50s

Figure 5.2: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 55s



(c) training MathSAT timeout = 60s

Figure 5.2: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts (cont.)

5.3 QF_LRA with Focused ParamILS using DAE

5.3.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	40s (the smt-comp configuration solved 85.64 % of tests)
Training MathSAT timeout 2	45s (the smt-comp configuration solved 86.13% of tests)
Training MathSAT timeout 3	50s (the smt-comp configuration solved 86.63% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_LRA of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=202 run_obj=runtime, overall_obj=mean

Table 5.7: The experimental setup of QF_LRA with different training MathSAT timeouts

5.3.2 Training Result

Table 5.8 shows the default configuration and the optimized configurations found by ParamILS. The main difference between the optimized configurations and the default configuration is that *pure_literal_filter* is disabled (instead of being enabled as default) and *split_eq* is enabled (instead of being disabled by default), and only one theory *la* is used in the optimized configurations while the default configuration uses two theories solver *euf, and la*.

5.3.3 Testing Result

Table 5.9 and Figure 5.3 show the performance comparison between the optimized, default and adaptive smt-comp configurations. It can be seen that the first optimized configuration solve *1 test* more compared with the *adaptive smt-comp* configuration and the other two optimized configurations has the mean runtime *reduced by a factor of around 1.5* compared with the *adaptive smt-comp* configuration. If compared with the default configuration, the three optimized configurations solve *7, or 6 tests more*.

Configuration Parameter	Default	40s	45s	50s
aep	yes	yes	no	no
deduction	2	2	2	2
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	yes	yes	no
frequent_reduce_db	no	yes	no	no
ghost_filter	no	no	no	yes
ibliwi	no	yes	no	no
impl_expl_threshold	0	0	0	0
incr_tsolvers	no	no	no	no
mixed_cs	yes	yes	yes	yes
permanent_theory_lemmas	yes	yes	yes	no
pure_literal_filter	**yes**	**no**	**no**	**no**
random_decisions	no	yes	no	no
restarts	normal	quick	adaptive	normal
sl	2	2	2	1
split_eq	**no**	**yes**	**yes**	**yes**
tcomb	off	off	off	off
toplevelprop	1	1	1	1
tsolver	euf la	la	la	la

Table 5.8: QF_LRA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**184/177/183**
Mean runtime(not include TIMEOUT tests)	**27.641s/35.057s/19.177s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	68/202
Equal runtime tests/The number of tests	37/202
Worse runtime tests/The number of tests	97/202

(a) training MathSAT timeout = 40s

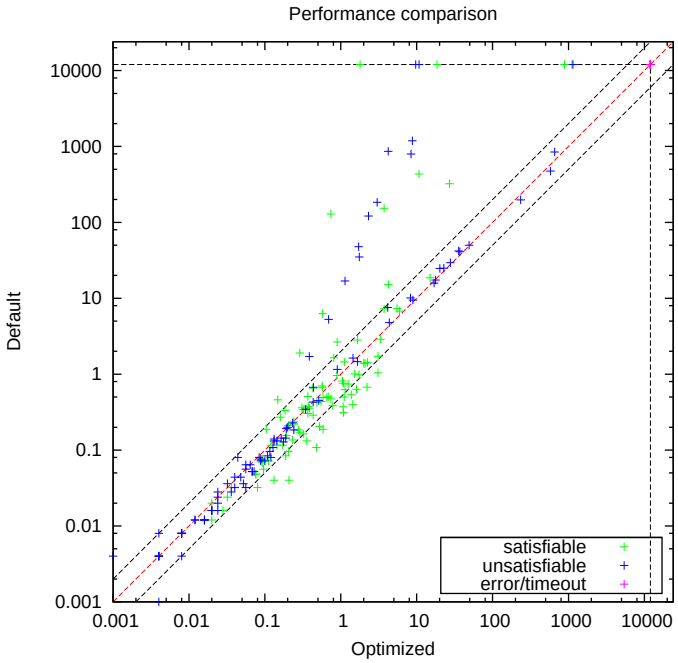
Tests solved (Optimized/Default/SMTCOMP)	**183/177/183**
Mean runtime(not include TIMEOUT tests)	**12.988s/35.057s/19.177s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	76/202
Equal runtime tests/The number of tests	33/202
Worse runtime tests/The number of tests	93/202

(b) training MathSAT timeout = 45s

Tests solved (Optimized/Default/SMTCOMP)	**183/177/183**
Mean runtime(not include TIMEOUT tests)	**13.155s/35.057s/19.177s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	78/202
Equal runtime tests/The number of tests	32/202
Worse runtime tests/The number of tests	92/202

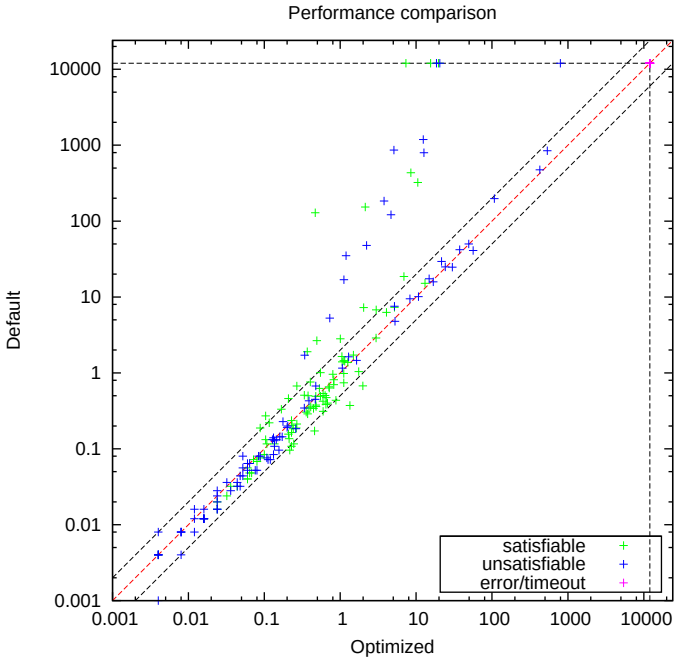
(c) training MathSAT timeout = 50s

Table 5.9: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts

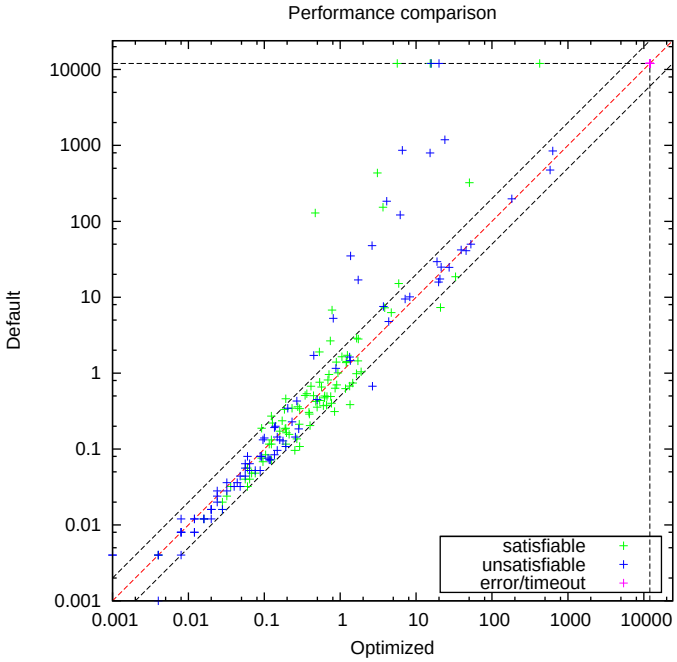


(a) training MathSAT timeout = 40s

Figure 5.3: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 45s



(c) training MathSAT timeout = 50s

Figure 5.3: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts (cont.)

5.4 QF_UFIDL with Focused ParamILS using DAE

5.4.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 82.67% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 83.66% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 84.65% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UFIDL of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=202 run_obj=runtime, overall_obj=mean

Table 5.10: The experimental setup of QF_UFIDL with different training MathSAT timeouts

5.4.2 Training Result

Table 5.11 shows the default configuration and the optimized configurations found by ParamILS. For QF_UFIDL, ParamILS suggests using *deduction* of 2 and *impl_expl_threshold* of 0, enabling *dyn_ack* and *ghost_filter*, disabling *expensive_ccmin*.

5.4.3 Testing Result

Table 5.12 and Figure 5.4 show the performance comparison of the optimized, default and *adaptive smt-comp* configuration. Similar to the case of QF_IDL, the optimized configurations solve *3 tests more* compared with the *adaptive smt-comp* configuration although the optimized configurations are ***fixed*** for all tests and the smt-comp configuration can be ***changed*** according to different problem classes.

Configuration Parameter	Default	50s	55s	60s
aep	yes	yes	yes	yes
deduction	**2**	**1**	**1**	**1**
dual_rail	off	off	off	off
dyn_ack	**no**	**yes**	**yes**	**yes**
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	**yes**	**no**	**no**	**no**
frequent_reduce_db	no	no	yes	yes
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	yes	yes	yes	yes
impl_expl_threshold	**0**	**1**	**1**	**1**
incr_tsolvers	no	no	no	no
mixed_cs	yes	yes	yes	yes
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	no	no	no	no
random_decisions	no	no	yes	yes
restarts	normal	normal	normal	normal
sl	2	2	2	2
split_eq	yes	yes	yes	yes
tcomb	dtc	dtc	dtc	dtc
toplevelprop	1	0	1	1
tsolver	dl euf euf	dl euf	dl euf	dl euf

Table 5.11: QF_UFIDL configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**192/182/189**
Mean runtime(not include TIMEOUT tests)	**24.645s/39.063s/33.415s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	150/202
Equal runtime tests/The number of tests	25/202
Worse runtime tests/The number of tests	27/202

(a) training MathSAT timeout = 50s

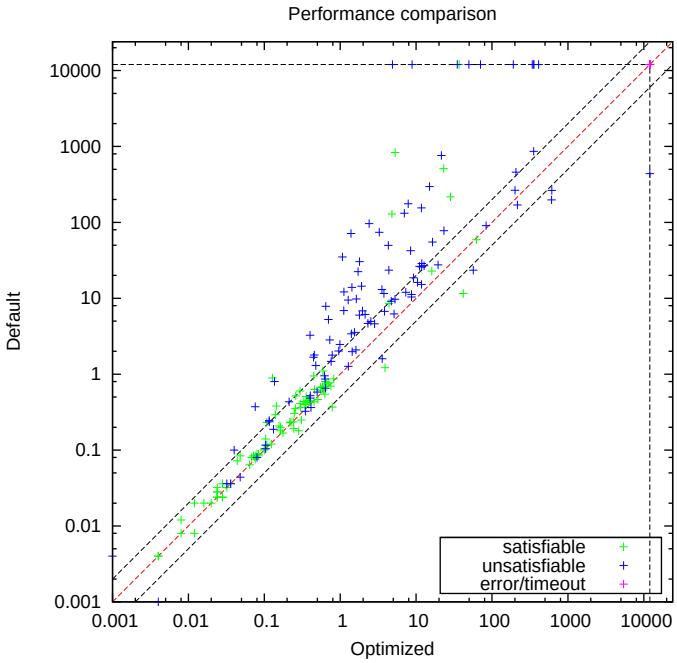
Tests solved (Optimized/Default/SMTCOMP)	**192/182/189**
Mean runtime(not include TIMEOUT tests)	**28.460s/39.063s/33.415s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	140/202
Equal runtime tests/The number of tests	21/202
Worse runtime tests/The number of tests	41/202

(b) training MathSAT timeout = 55s

Tests solved (Optimized/Default/SMTCOMP)	**192/182/189**
Mean runtime(not include TIMEOUT tests)	**28.502s/39.063s/33.415s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	138/202
Equal runtime tests/The number of tests	22/202
Worse runtime tests/The number of tests	42/202

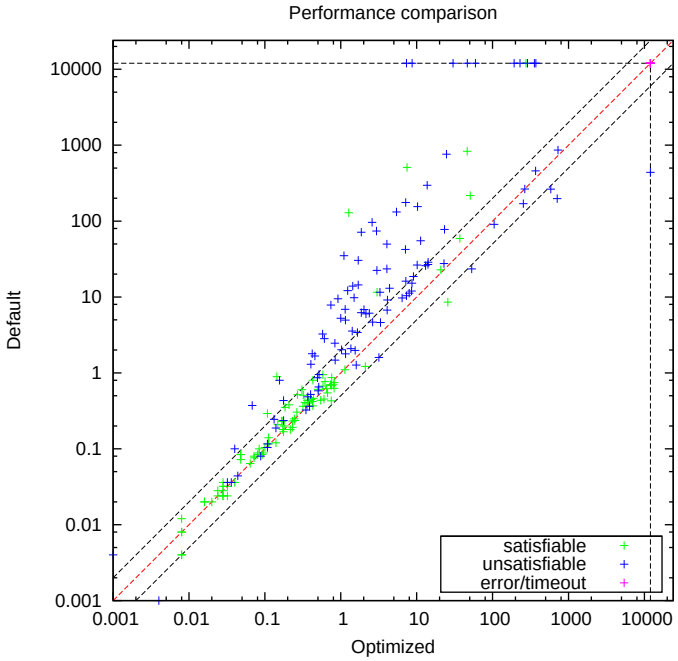
(c) training MathSAT timeout = 60s

Table 5.12: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts

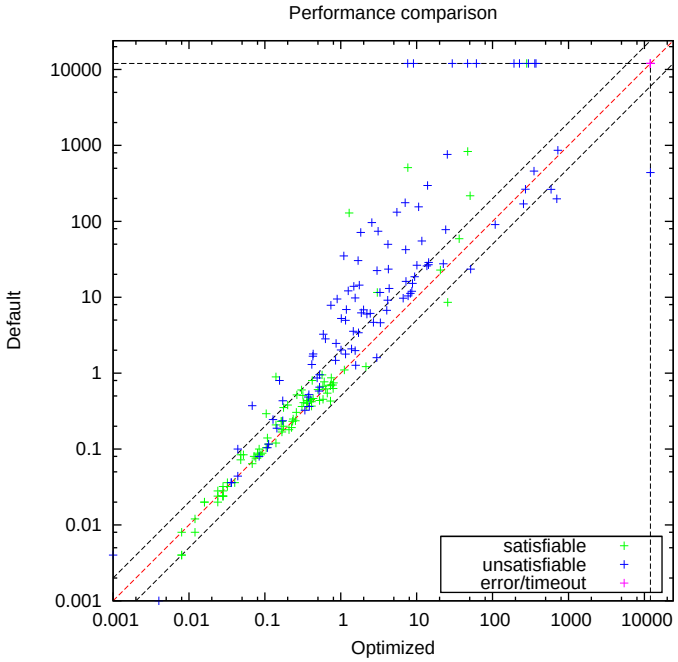


(a) training MathSAT timeout = 50s

Figure 5.4: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 55s



(c) training MathSAT timeout = 60s

Figure 5.4: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts (cont.)

5.5 QF_UFLRA with Focused ParamILS using DAE

5.5.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	5s (the smt-comp configuration solved 100% of tests)
Training MathSAT timeout 2	10s (the smt-comp configuration solved 100% of tests)
Training MathSAT timeout 3	15s (the smt-comp configuration solved 100% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UFLRA of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=200 run_obj=runtime, overall_obj=mean

Table 5.13: The experimental setup of QF_UFLRA with different training MathSAT timeouts

5.5.2 Training Result

Table 5.14 shows the default configuration and the optimized configurations found by ParamILS. For this theory, ParamILS suggest using *deduction* of 1, *sl* of 2, and *toplevelprop* of 1. ParamILS also recommend users to enable *dyn_ack*, *incr_tsolvers*, and disable *expensive_ccmin*, *pure_literal_filter*.

5.5.3 Testing Result

Table 5.15 and Figure 5.5 show the performance comparison of the optimized, default and *adaptive smt-comp* configuration. In all cases, the mean runtime of the optimized configurations are reduced approximately by *a factor of 8* and *a factor of 4* compared with the mean runtime of the *adaptive smt-comp* and default configuration, respectively.

Configuration Parameter	Default	5s	10s	15s
aep	yes	yes	yes	yes
deduction	**2**	**1**	**1**	**1**
dual_rail	off	off	off	off
dyn_ack	**no**	**yes**	**yes**	**yes**
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	50	1	50
expensive_ccmin	**yes**	**no**	**no**	**no**
frequent_reduce_db	no	no	no	yes
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	no	no	yes	yes
impl_expl_threshold	0	0	0	0
incr_tsolvers	**no**	**yes**	**yes**	**yes**
mixed_cs	yes	yes	yes	yes
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	**yes**	**no**	**no**	**no**
random_decisions	no	no	no	no
restarts	normal	normal	normal	normal
sl	**2**	**0**	**0**	**0**
split_eq	no	no	no	no
tcomb	dtc	dtc	dtc	dtc
toplevelprop	**1**	**0**	**0**	**0**
tsolver	euf la	euf la	euf la	euf la

Table 5.14: QF_UFLRA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.056s/0.204s/0.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	190/200
Equal runtime tests/The number of tests	5/200
Worse runtime tests/The number of tests	5/200

(a) training MathSAT timeout = 5s

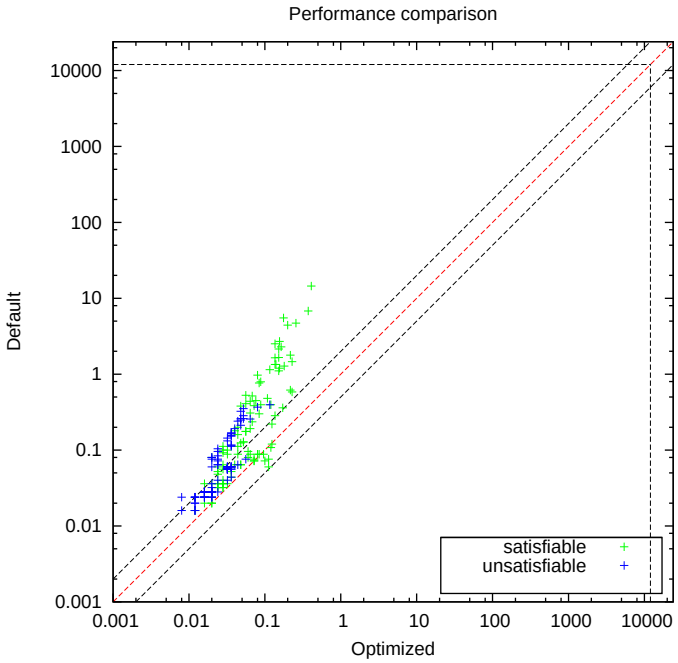
Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.059s/0.204s/0.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	186/200
Equal runtime tests/The number of tests	5/200
Worse runtime tests/The number of tests	9/200

(b) training MathSAT timeout = 10s

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.062s/0.204s/0.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	185/200
Equal runtime tests/The number of tests	11/200
Worse runtime tests/The number of tests	4/200

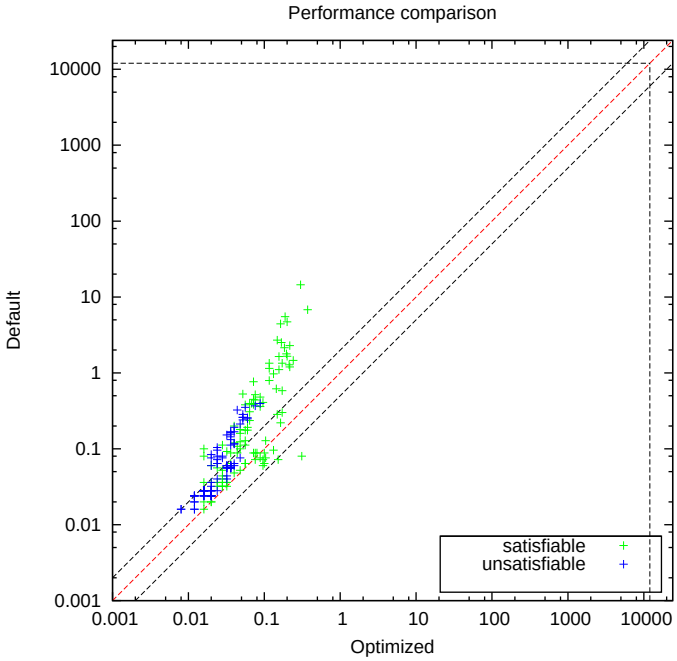
(c) training MathSAT timeout = 15s

Table 5.15: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts

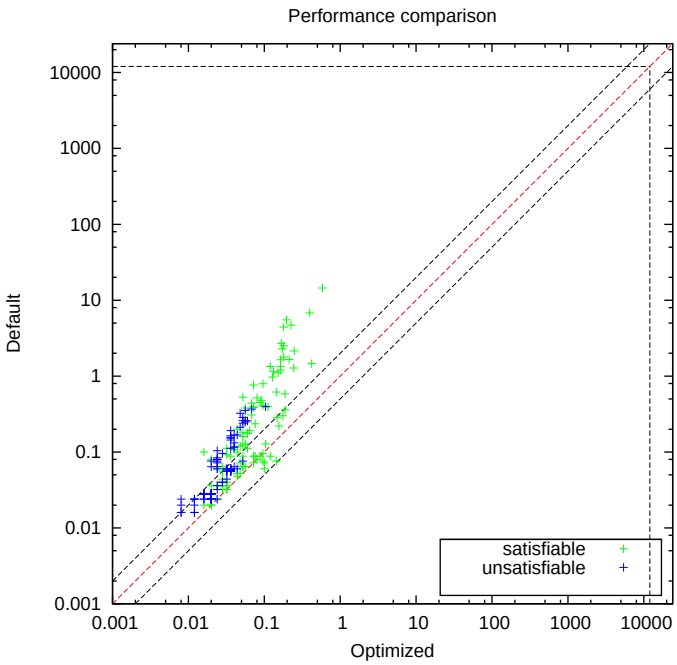


(a) training MathSAT timeout = 5s

Figure 5.5: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 10s



(c) training MathSAT timeout = 15s

Figure 5.5: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts (cont.)

5.6 Summary

Table 5.16 summaries the performance comparison of the five theories on the SMT-COMP benchmark.

In three theories of QF_IDL, QF_LRA, and QF_UFIDL, the number of solved tests is improved significantly. For example, in the case of QF_IDL, the optimized configuration found by ParamILS solves *14 tests more* compared with the *adaptive* smt-comp configuration (this configuration can be changed according to different problem classes) and *10 tests more* compared with the *winner* of SMT-COMP 2009 on this theory (although we are using the older version of MathSAT, not the new one in the competition).

In two other cases of QF_LIA, and QF_UFLRA, the number of tests solved by the optimized configuration is equal to the number of tests solved by the adaptive smt-comp configuration. However, the mean runtime is reduced approximately by *half* in case of QF_LIA and by *a factor of eight* in case of QF_UFLRA.

Tests solved (Optimized/Default/SMTCOMP)	**96/81/82**
Mean runtime (not include TIMEOUT tests)	**14.755s/25.069s/40.349s**
The number of tests	102

(a) QF_IDL

Tests solved (Optimized/Default/SMTCOMP)	**202/201/202**
Mean runtime(not include TIMEOUT tests)	**13.850s/15.495s/26.322s**
The number of tests	205

(b) QF_LIA

Tests solved (Optimized/Default/SMTCOMP)	**184/177/183**
Mean runtime(not include TIMEOUT tests)	**27.641s/35.057s/19.177s**
The number of tests	202

(c) QF_LRA

Tests solved (Optimized/Default/SMTCOMP)	**192/182/189**
Mean runtime(not include TIMEOUT tests)	**24.645s/39.063s/33.415s**
The number of tests	202

(d) QF_UFIDL

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.056s/0.204s/0.410s**
The number of tests	200

(e) QF_UFLRA

Table 5.16: Performance Comparison of the five theories on the SMT-COMP benchmark

Chapter 6

Configuring on Other Theories on the SMT-COMP Benchmark

This chapter summaries the experimental results on the three other theories QF_UFLIA, QF_UF, QF_RDL. In these theories, we found bugs in the testing phase this is due to the fact that in the training phase, we use not only the tested parameter but also the ***internal parameters*** which were used by developers and not tested carefully.

Before moving to the next sections of experiments, we introduce briefly the three theories experimented in this chapter:

- **QF_UF**: Unquantified formulas built over a signature of uninterpreted (i.e., free) sort and function symbols. For example: $\{(f(x_1) = f(x_2)), \neg(f(x_2) = f(x_3))\}$.
- **QF_RDL** Difference Logic over the reals. In essence, Boolean combinations of inequations of the form $(x - y < b)$ where x and y are real variables and b is a rational constant.
- **QF_UFLIA**: Unquantified linear integer arithmetic with uninterpreted sort and function symbols.

6.1 QF_UFLIA with Focused ParamILS using DAE

6.1.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 98.01% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 98.01% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 98.51% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UFLIA of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=202 run_obj=runtime, overall_obj=mean

Table 6.1: The experimental setup of QF_UFLIA with different training MathSAT timeouts

6.1.2 Experimental result

Testing bugs:

In testing phase, we found many inconsistent results. For example:

bin/mathsat -input=smt -solve -logic=QF_UFLIA data/QF_UFLIA/wisas/xs_16_36.smt returns UNSAT.

bin/mathsat -input=smt -solve -deduction=2 -impl_expl_threshold=0 -no_aep -dual_rail=off -no_pure_literal_filter -random_decisions -split_eq -no_incr_tsolvers -tsolver=dl -tsolver=laz -dyn_ack -dyn_ack_limit=0 -dyn_ack_threshold=10 -no_expensive_ccmin -no_frequent_reduce_db -no_ghost_filter -no_ibliwi -mixed_cs -permanent_theory_lemmas -restarts=adaptive -sl=0 -tcomb=dtc -toplevelprop=2 data/QF_UFLIA/wisas/xs_16_36.smt returns SAT.

6.2 QF_UF with Focused ParamILS using DAE

6.2.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	45s (the smt-comp configuration solved 79.00% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 80.00% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 81.00% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UF of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=200 run_obj=runtime, overall_obj=mean

Table 6.2: The experimental setup of QF_UF with different training MathSAT timeouts

6.2.2 Experimental result

Testing bugs:

In testing phase, we found many inconsistent results. For example:

bin/mathsat -input=smt -solve -logic=QF_UF data/QF_UF/QG-classification/qg6/iso_icl_nogen
returns UNSAT.

bin/mathsat -input=smt -solve -deduction=1 -impl_expl_threshold=0 -no_aep -dual_rail=circuit -no_pure_literal_filter -random_decisions -no_split_eq -tsolver=euf -no_dyn_ack -expensive_ccmin -frequent_reduce_db -no_ghost_filter -ibliwi -mixed_cs -permanent_theory_lemmas -restarts=normal -sl=2 -toplevelprop=0 data/QF_UF/QG-classification/qg6/iso_icl_nogen013.smt
returns SAT.

6.3 QF_RDL with Focused ParamILS using DAE

6.3.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 86.85% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 88.00% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 89.14% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_RDL of SMT-COMP 2009
Testing set	Training set
ParamILS Configuration	Focused, deterministic=1, N=175 run_obj=runtime, overall_obj=mean

Table 6.3: The experimental setup of QF_RDL with different training MathSAT timeouts

6.3.2 Experimental result

Testing bugs:

In testing phase, we found many inconsistent results of MathSAT on the same test. For example:

bin/mathsat -input=smt -solve -logic=QF_RDL data/QF_RDL/sal/fischer6-mutex-20.smt
returns UNSAT.

bin/mathsat -input=smt -solve -deduction=0 -impl_expl_threshold=0 -aep -random_decisions -no_split_eq -tsolver=euf -tsolver=dl -expensive_ccmin -frequent_reduce_db -ghost_filter -dual_rail=off -no_pure_literal_filter -ibliwi -no_mixed_cs -permanent_theory_lemmas -restarts=normal -sl=0 -toplevelprop=2 data/QF_RDL/sal/fischer9-mutex-20.smt
returns SAT.

Chapter 7

More Results of the Five Theories on the SMT-LIB Benchmark

This section shows the results of using ParamILS to find the best possible configurations for the five theories having no bugs (in Chapter 5) on the SMT-LIB benchmark. The experiments in this section use *different datasets* of SMT-LIB for training and testing because we want to get the best possible MathSAT configurations for *general cases*. For each theory, a training dataset is extracted from SMT-LIB by using the benchmark selection tool of SMT-COMP 2009. Then, we remove that training dataset from SMT-LIB, and extract another dataset for testing.

The experiment setup for all tests in this chapter is summarised as follows: Based on the experimental results in Chapter 4, Focused ParamILS using DAE is the best strategy for MathSAT. In addition, the training time of 48 hours is enough for MathSAT to converge with the MathSAT timeouts less than 60 seconds. As for choosing the training MathSAT timeouts, we run MathSAT with the smtcomp configuration on the training dataset in advance and choose the three MathSAT timeouts (all timeouts have value from 0 to 1200 with step 5) which are less than 60 seconds and solve similar percentage of tests.

Notice that the smt-comp configurations are *adaptive*, i.e. they can be changed according to different problem classes based on a statistics module in MathSAT.

7.1 QF_IDL with Focused ParamILS using DAE

7.1.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	35s (the smt-comp configuration solved 75.49% of tests)
Training MathSAT timeout 2	40s (the smt-comp configuration solved 76.47% of tests)
Training MathSAT timeout 3	45s (the smt-comp configuration solved 76.47% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_IDL of SMT-LIB
Testing set	the QF_IDL of SMT-LIB (removed training set)
ParamILS Configuration	Focused, deterministic=1, N=102 run_obj=runtime, overall_obj=mean

Table 7.1: The experimental setup of QF_IDL with different training MathSAT timeouts

7.1.2 Training Result

Table 7.2 shows the default configuration and the optimized configurations found by ParamILS. For this theory, ParamILS suggest enabling *random_decisions*, *restarting adaptively* and using *toplevelprop* of 0 instead of 2 as default.

7.1.3 Testing Result

Table 7.3 and Figure 7.1 show the performance comparison of the optimized, default and *adaptive smt-comp* configurations. The second optimized configuration solves *1 test more* compared with the default configuration. If compared with the *adaptive smt-comp* configuration, only the second optimized configuration solves the same number of tests and the mean runtime of this optimized configuration is reduced slightly by *a factor of 1.275*.

Configuration Parameter	Default	35s	40s	45s
aep	yes	yes	yes	yes
deduction	2	2	2	1
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	yes	yes	yes
frequent_reduce_db	no	no	no	no
ghost_filter	no	no	no	no
ibliwi	yes	no	yes	no
impl_expl_threshold	0	0	0	0
incr_tsolvers	no	no	no	no
mixed_cs	yes	yes	yes	yes
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	no	no	no	no
random_decisions	**no**	**no**	**yes**	**yes**
restarts	**normal**	**adaptive**	**adaptive**	**quick**
sl	2	2	2	2
split_eq	yes	yes	yes	yes
tcomb	off	off	off	off
toplevelprop	1	2	0	0
tsolver	dl	dl	dl	dl

Table 7.2: QF_IDL configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**94/94/95**
Mean runtime (not include TIMEOUT tests)	**1.848s/2.744s/6.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	60/100
Equal runtime tests/The number of tests	29/100
Worse runtime tests/The number of tests	11/100

(a) training MathSAT timeout = 35s

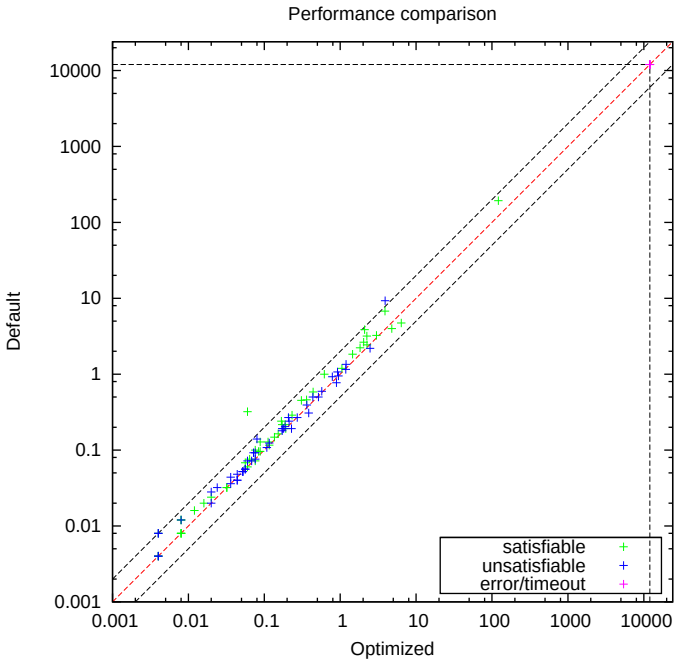
Tests solved (Optimized/Default/SMTCOMP)	**95/94/95**
Mean runtime(not include TIMEOUT tests)	**5.207s/2.744s/6.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	50/100
Equal runtime tests/The number of tests	12/100
Worse runtime tests/The number of tests	38/100

(b) training MathSAT timeout = 40s

Tests solved (Optimized/Default/SMTCOMP)	**94/94/95**
Mean runtime(not include TIMEOUT tests)	**1.352s/2.744s/6.410s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	42/100
Equal runtime tests/The number of tests	14/100
Worse runtime tests/The number of tests	44/100

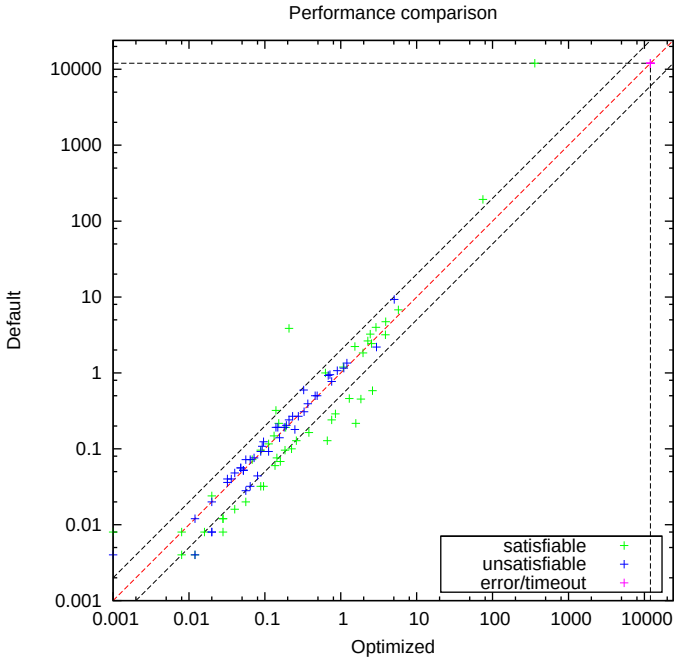
(c) training MathSAT timeout = 45s

Table 7.3: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts

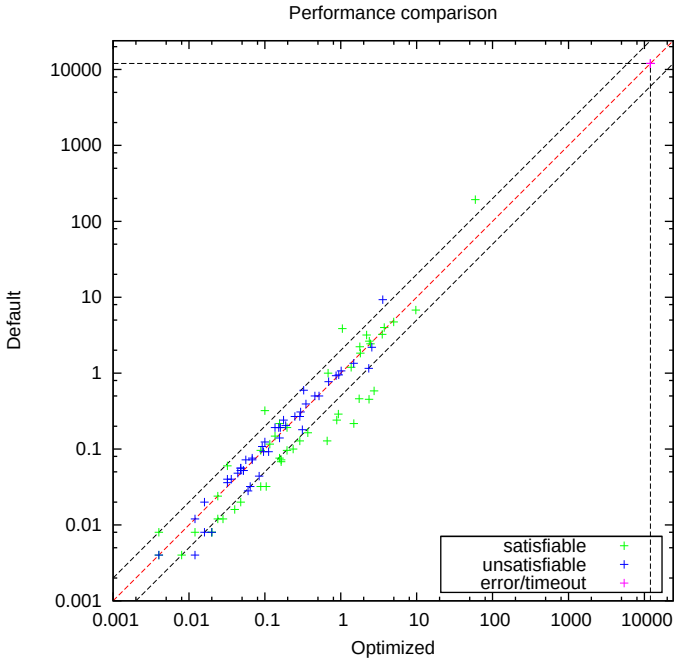


(a) training MathSAT timeout = 35s

Figure 7.1: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 40s



(c) training MathSAT timeout = 45s

Figure 7.1: Performance comparison of QF_IDL configurations found by different training MathSAT timeouts (cont.)

7.2 QF_LIA with Focused ParamILS using DAE

7.2.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 83.41% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 83.90% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 85.58% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_LIA of SMT-LIB
Testing set	the QF_LIA of SMT-LIB (remove training set)
ParamILS Configuration	Focused, deterministic=1, N=205 run_obj=runtime, overall_obj=mean

Table 7.4: The experimental setup of QF_LIA with different training MathSAT timeouts

7.2.2 Training Result

Table 7.5 shows the default configuration and the optimized configuration found by ParamILS. For QF_LIA, ParamILS suggest using *impl_expl_threshold* of 1 instead of 0, *sl* of 0 instead of 2, *toplevelprop* of 1, enabling *incr_tsolvers* and disabling *split_eq*.

7.2.3 Testing Result

Table 7.6 and Figure 7.2 of the optimized, default and *adaptive smt-comp* configurations. Although only in two of three cases, the optimized configurations solve the same number of tests compared with the default and the adaptive smt-comp configurations, the mean runtime of the optimized configurations are reduced approximately by *a factor of 1.35, 3.56* compared with the mean runtime of the default and smt-comp configuration, respectively.

Configuration Parameter	Default	50s	55s	60s
aep	yes	yes	no	yes
deduction	2	2	1	2
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	no	no	yes
frequent_reduce_db	no	no	no	no
ghost_filter	no	no	yes	no
ibliwi	no	yes	no	no
impl_expl_threshold	**0**	**1**	**0**	**1**
incr_tsolvers	**no**	**yes**	**yes**	**yes**
mixed_cs	**yes**	**no**	**yes**	**no**
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	yes	yes	no	yes
random_decisions	no	no	yes	no
restarts	normal	normal	normal	normal
sl	**2**	**0**	**0**	**0**
split_eq	**yes**	**no**	**no**	**no**
tcomb	off	off	off	off
toplevelprop	**1**	**1**	**0**	**1**
tsolver	euf laz	laz	euf laz	euf laz

Table 7.5: QF_LIA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**199/199/199**
Mean runtime (not include TIMEOUT tests)	**12.456s/17.369s/44.378s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	181/200
Equal runtime tests/The number of tests	5/200
Worse runtime tests/The number of tests	14/200

(a) training MathSAT timeout = 50s

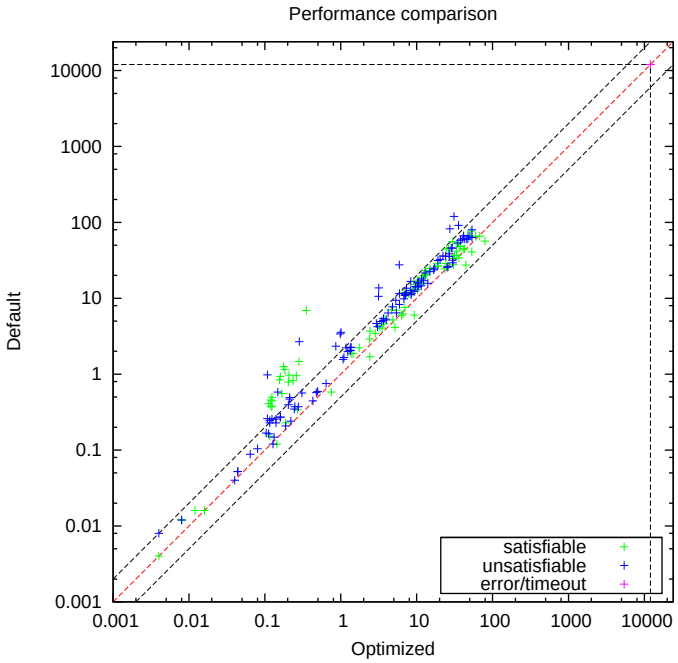
Tests solved (Optimized/Default/SMTCOMP)	**198/199/199**
Mean runtime(not include TIMEOUT tests)	**12.791s/17.369s/44.378s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	148/200
Equal runtime tests/The number of tests	8/200
Worse runtime tests/The number of tests	44/200

(b) training MathSAT timeout = 55s

Tests solved (Optimized/Default/SMTCOMP)	**199/199/199**
Mean runtime(not include TIMEOUT tests)	**12.069s/17.369s/44.378s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	183/200
Equal runtime tests/The number of tests	8/200
Worse runtime tests/The number of tests	9/200

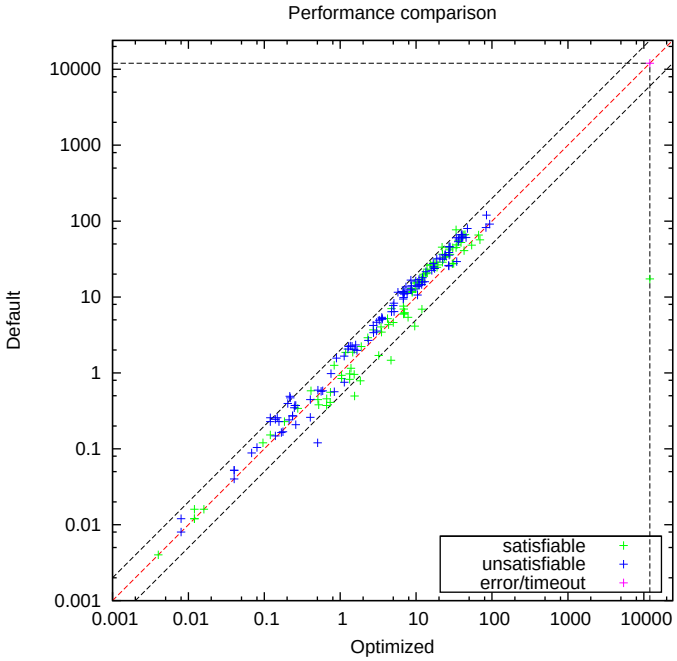
(c) training MathSAT timeout = 60s

Table 7.6: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts

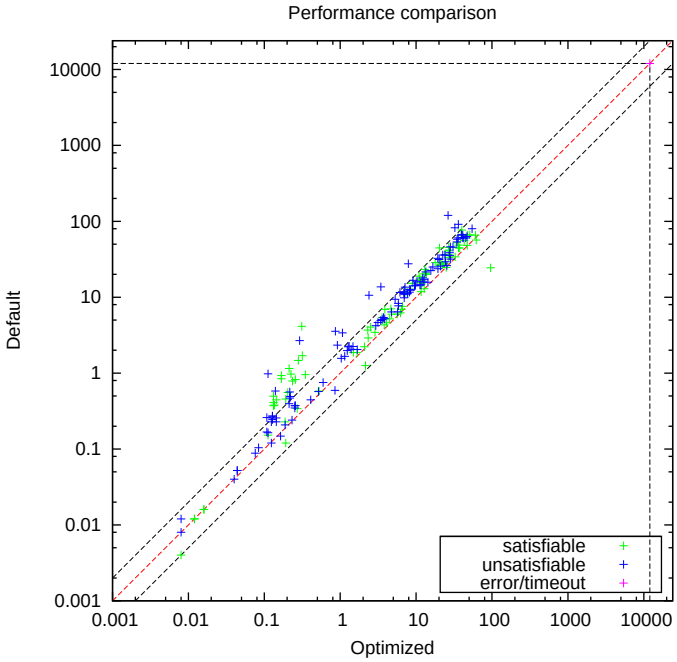


(a) training MathSAT timeout = 50s

Figure 7.2: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 55s



(c) training MathSAT timeout = 60s

Figure 7.2: Performance comparison of QF_LIA configurations found by different training MathSAT timeouts (cont.)

7.3 QF_LRA with Focused ParamILS using DAE

7.3.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	40s (the smt-comp configuration solved 85.64 % of tests)
Training MathSAT timeout 2	45s (the smt-comp configuration solved 86.13% of tests)
Training MathSAT timeout 3	50s (the smt-comp configuration solved 86.63% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_LRA of SMT-LIB
Testing set	the QF_LRA of SMT-LIB (removed training set)
ParamILS Configuration	Focused, deterministic=1, N=202 run_obj=runtime, overall_obj=mean

Table 7.7: The experimental setup of QF_LRA with different training MathSAT timeouts

7.3.2 Training Result

Table 7.8 shows the default configuration and the optimized configurations found by ParamILS. For QF_LRA, ParamILS suggests disabling *aep, pure_literal_filter*, enabling *split_eq* and using only one theory solver *la*.

7.3.3 Testing Result

Table 7.9 and Figure 7.3 show the performance comparison of the optimized, deaf-ault and *adaptive smt-comp* configuration. Only the second optimized configuration has the same number of solved tests as the smt-comp configuration. However, if compared with the default configuration, the optimized configurations solve *6, or 7 tests more*.

Configuration Parameter	Default	40s	45s	50s
aep	**yes**	**no**	**no**	**no**
deduction	2	2	2	2
dual_rail	off	off	off	off
dyn_ack	no	no	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	yes	yes	yes
frequent_reduce_db	no	no	no	yes
ghost_filter	no	no	yes	no
ibliwi	no	no	no	no
impl_expl_threshold	0	0	0	1
incr_tsolvers	no	no	no	no
mixed_cs	yes	no	no	no
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	**yes**	**no**	**no**	**yes**
random_decisions	no	no	no	no
restarts	normal	normal	normal	normal
sl	2	2	2	2
split_eq	**no**	**yes**	**yes**	**yes**
tcomb	off	off	off	off
toplevelprop	1	1	1	1
tsolver	**euf la**	**la**	**la**	**la**

Table 7.8: QF_LRA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**199/193/200**
Mean runtime(not include TIMEOUT tests)	**11.099s/31.269s/9.033s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	88/200
Equal runtime tests/The number of tests	23/200
Worse runtime tests/The number of tests	89/200

(a) training MathSAT timeout = 40s

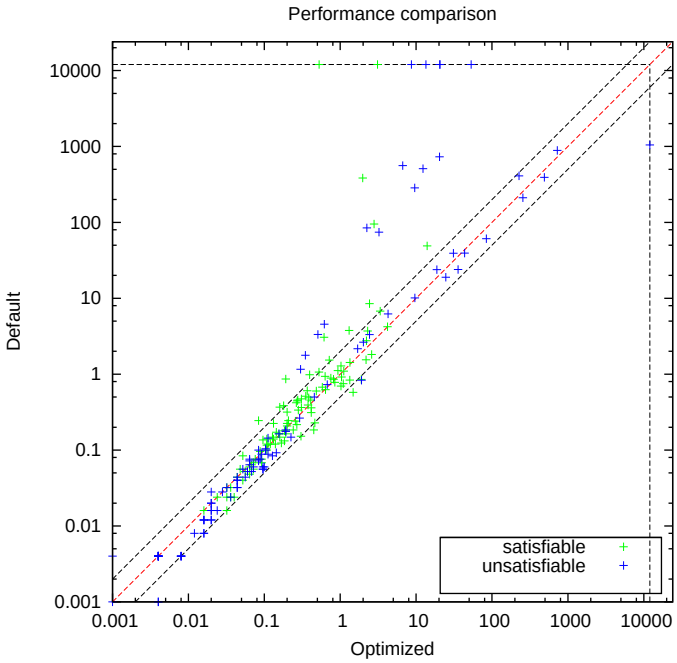
Tests solved (Optimized/Default/SMTCOMP)	**200/193/200**
Mean runtime(not include TIMEOUT tests)	**19.759s/31.269s/9.033s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	90/200
Equal runtime tests/The number of tests	18/200
Worse runtime tests/The number of tests	92/200

(b) training MathSAT timeout = 45s

Tests solved (Optimized/Default/SMTCOMP)	**199/193/200**
Mean runtime(not include TIMEOUT tests)	**13.799s/31.269s/9.033s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	65/200
Equal runtime tests/The number of tests	17/200
Worse runtime tests/The number of tests	118/200

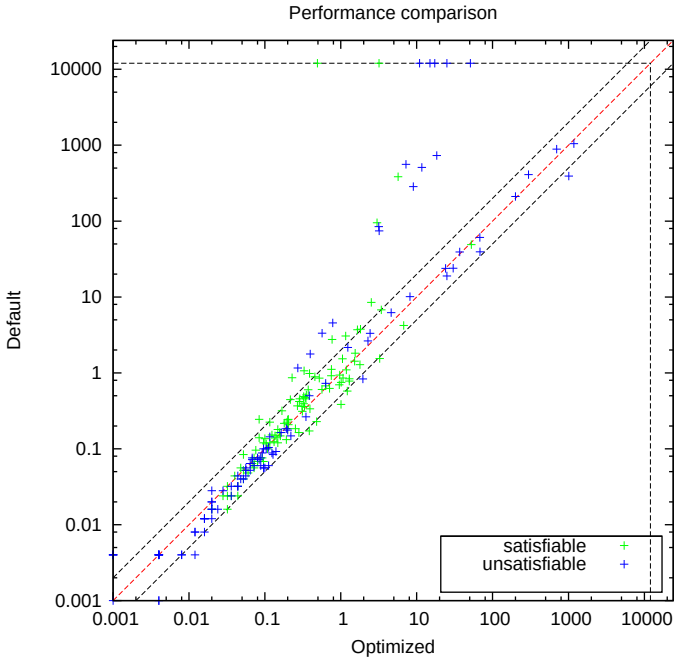
(c) training MathSAT timeout = 50s

Table 7.9: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts

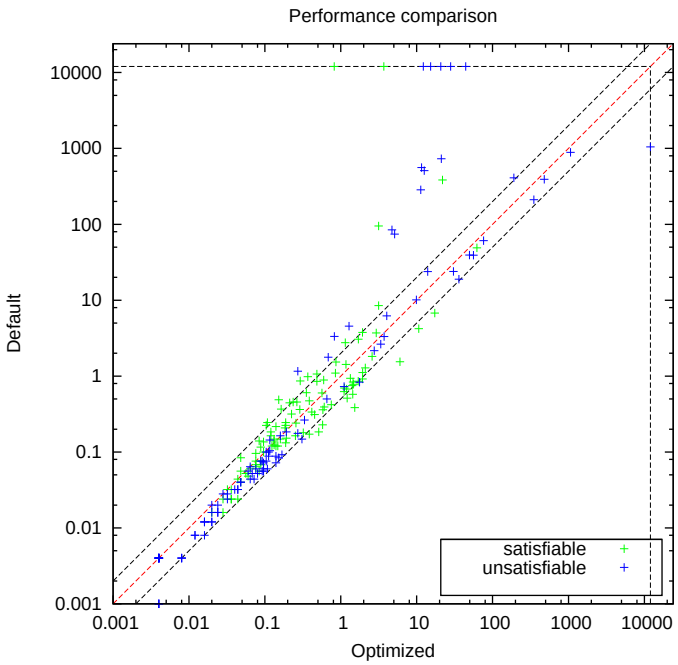


(a) training MathSAT timeout = 40s

Figure 7.3: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 45s



(c) training MathSAT timeout = 50s

Figure 7.3: Performance comparison of QF_LRA configurations found by different training MathSAT timeouts (cont.)

7.4 QF_UFIDL with Focused ParamILS using DAE

7.4.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	50s (the smt-comp configuration solved 82.67% of tests)
Training MathSAT timeout 2	55s (the smt-comp configuration solved 83.66% of tests)
Training MathSAT timeout 3	60s (the smt-comp configuration solved 84.65% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UFIDL of SMT-LIB
Testing set	the QF_UFIDL of SMT-LIB (removed training)
ParamILS Configuration	Focused, deterministic=1, N=202 run_obj=runtime, overall_obj=mean

Table 7.10: The experimental setup of QF_UFIDL with different training MathSAT timeouts

7.4.2 Training Result

Table 7.11 shows the default configuration and the optimized configurations found by ParamILS. The main difference between the optimized configurations and the default configuration is that using *deduction* of 2 instead of 1, and enabling *ghost_filter* instead of disabling as default.

7.4.3 Testing Result

Table 7.12 and Figure 7.4 show the performance comparison of the optimized, default and *adaptive smt-comp* configuration. There is no optimized configuration having the number of solved tests greater than the number of tests solved by the smt-comp configuration. However, if compared with the default configuration, the optimized configurations solve *11, 10, 6 tests more*.

Configuration Parameter	Default	50s	55s	60s
aep	yes	yes	no	no
deduction	**2**	**1**	**1**	**1**
dual_rail	off	off	off	off
dyn_ack	no	yes	no	no
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	1	1	1	1
expensive_ccmin	yes	yes	yes	no
frequent_reduce_db	no	no	no	yes
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	yes	yes	yes	yes
impl_expl_threshold	0	1	0	1
incr_tsolvers	no	no	no	no
mixed_cs	yes	yes	no	no
permanent_theory_lemmas	yes	yes	yes	yes
pure_literal_filter	no	no	no	no
random_decisions	no	no	yes	yes
restarts	normal	quick	normal	adaptive
sl	2	2	0	1
split_eq	yes	yes	yes	yes
tcomb	dtc	dtc	decide	decide
toplevelprop	1	1	1	0
tsolver	euf dl	euf dl	euf dl	euf dl

Table 7.11: QF_UFIDL configurations found by different training MathSAT time-outs

Tests solved (Optimized/Default/SMTCOMP)	**198/187/200**
Mean runtime(not include TIMEOUT tests)	**23.804s/39.339s/30.264s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	162/200
Equal runtime tests/The number of tests	11/200
Worse runtime tests/The number of tests	27/200

(a) training MathSAT timeout = 50s

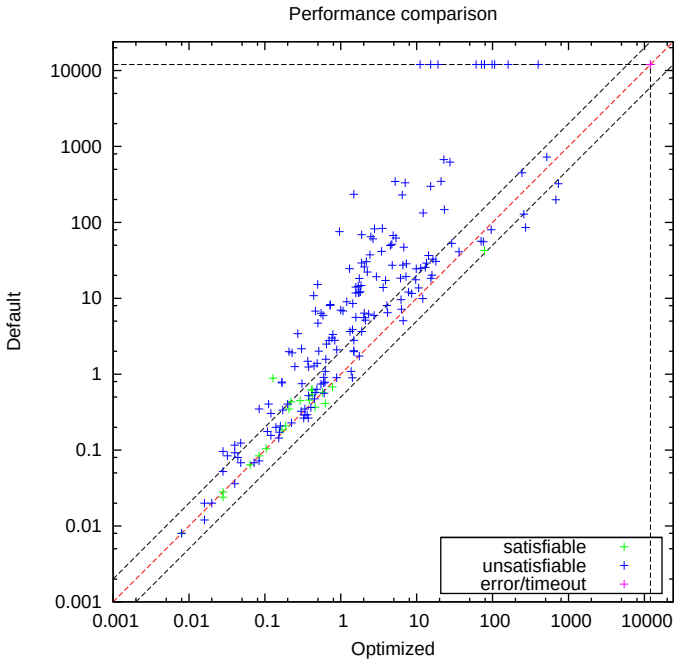
Tests solved (Optimized/Default/SMTCOMP)	**197/187/200**
Mean runtime(not include TIMEOUT tests)	**26.924s/39.339s/30.264s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	139/200
Equal runtime tests/The number of tests	7/200
Worse runtime tests/The number of tests	54/200

(b) training MathSAT timeout = 55s

Tests solved (Optimized/Default/SMTCOMP)	**193/187/200**
Mean runtime(not include TIMEOUT tests)	**20.543s/39.339s/30.264s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	132/200
Equal runtime tests/The number of tests	10/200
Worse runtime tests/The number of tests	58/200

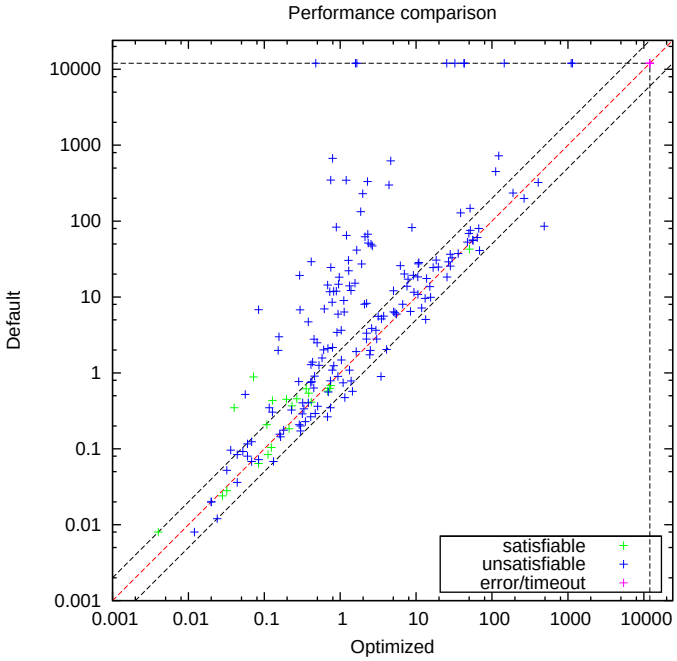
(c) training MathSAT timeout = 60s

Table 7.12: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts

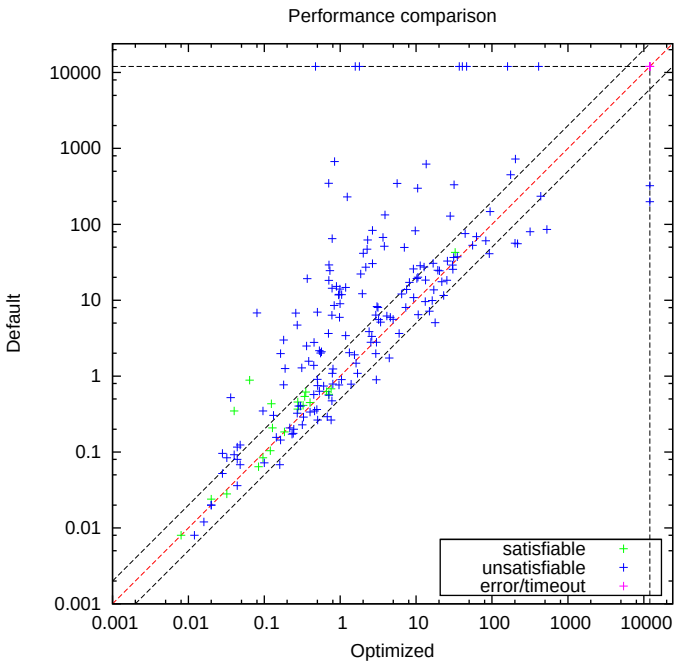


(a) training MathSAT timeout = 50s

Figure 7.4: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 55s



(c) training MathSAT timeout = 60s

Figure 7.4: Performance comparison of QF_UFIDL configurations found by different training MathSAT timeouts (cont.)

7.5 QF_UFLRA with Focused ParamILS using DAE

7.5.1 Experimental setup

CPU	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Operating system	Debian 5.0.4
MathSAT version	v4.2.10-itp (May 12 2010 08:54:51, gmp 4.3.2, gcc 4.4.3)
Training MathSAT timeout 1	5s (the smt-comp configuration solved 100% of tests)
Training MathSAT timeout 2	10s (the smt-comp configuration solved 100% of tests)
Training MathSAT timeout 3	15s (the smt-comp configuration solved 100% of tests)
Testing MathSAT timeout	1200s
Training time	48 hours
Training set	the QF_UFLRA of SMT-LIB
Testing set	the QF_UFLRA of SMT-LIB (removed training)
ParamILS Configuration	Focused, deterministic=1, N=200 run_obj=runtime, overall_obj=mean

Table 7.13: The experimental setup of QF_UFLRA with different training MathSAT timeouts

7.5.2 Training Result

Table 7.14 shows the default configuration and the optimized configurations found ParamILS. For QF_UFLRA, ParamILS recommends users to enable *dyn_ack*, *frequent_reduce_db*, *ghost_filter*, *incr_tsolvers* and disable *pure_literal_filter*. ParamILS also suggests using *deduction* of 1, *dyn_ack_threshold* of 50, or 10, *sl* of 0, *toplevel-prop* of 0.

7.5.3 Testing Result

Table 7.15 and Figure 7.5 show the performance comparison of the optimized, default and *adaptive smt-comp* configurations. It can be seen that in call cases, the mean runtime of the optimized configurations are reduced approximately by *a factor of 6.678 and 3.375* compared with the mean runtime of the default, and smt-comp configuration.

Configuration Parameter	Default	5s	10s	15s
aep	yes	yes	yes	yes
deduction	**2**	**1**	**1**	**1**
dual_rail	off	off	off	off
dyn_ack	**no**	**yes**	**yes**	**yes**
dyn_ack_limit	0	0	0	0
dyn_ack_threshold	**1**	**50**	**50**	**10**
expensive_ccmin	yes	yes	no	yes
frequent_reduce_db	**no**	**yes**	**yes**	**yes**
ghost_filter	**no**	**yes**	**yes**	**yes**
ibliwi	no	no	no	no
impl_expl_threshold	0	0	0	0
incr_tsolvers	**no**	**yes**	**yes**	**yes**
mixed_cs	yes	yes	yes	yes
permanent_theory_lemmas	yes	yes	no	yes
pure_literal_filter	**yes**	**no**	**no**	**no**
random_decisions	no	no	no	no
restarts	normal	normal	normal	normal
sl	**2**	**0**	**0**	**0**
split_eq	no	no	no	no
tcomb	dtc	dtc	dtc	dtc
toplevelprop	**1**	**0**	**0**	**0**
tsolver	euf la	euf la	euf la	euf la

Table 7.14: QF_UFLRA configurations found by different training MathSAT timeouts

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.057s/0.374s/0.189s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	193/200
Equal runtime tests/The number of tests	4/200
Worse runtime tests/The number of tests	3/200

(a) training MathSAT timeout = 5s

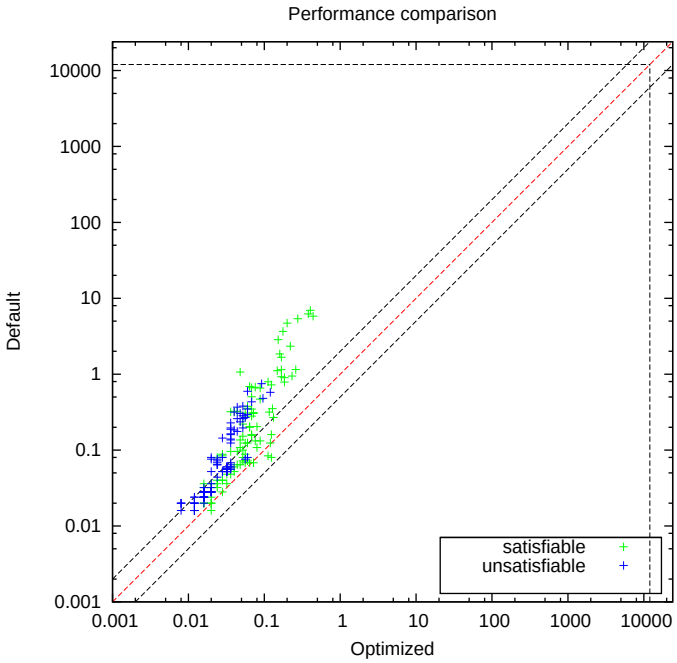
Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.059s/0.374s/0.189s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	189/200
Equal runtime tests/The number of tests	4/200
Worse runtime tests/The number of tests	7/200

(b) training MathSAT timeout = 10s

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.056s/0.374s/0.189s**
Optimized compared with Default	Result
Better runtime tests/The number of tests	192/200
Equal runtime tests/The number of tests	5/200
Worse runtime tests/The number of tests	3/200

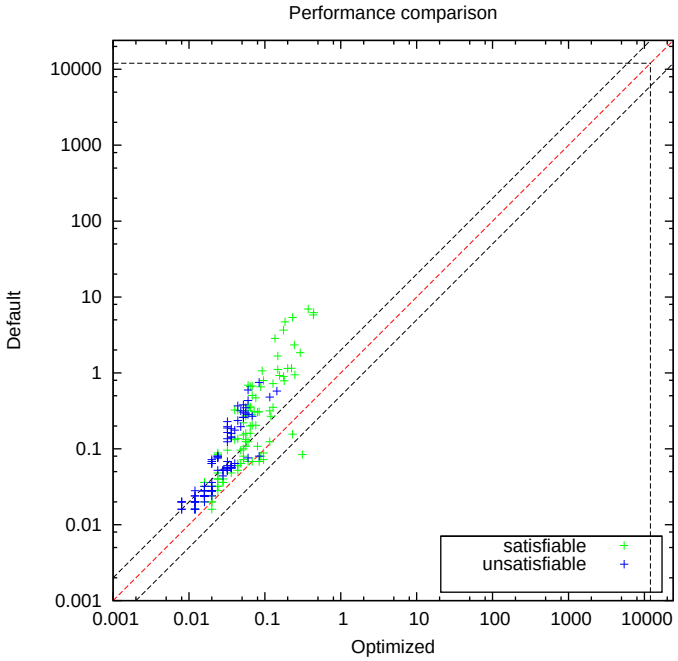
(c) training MathSAT timeout = 15s

Table 7.15: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts

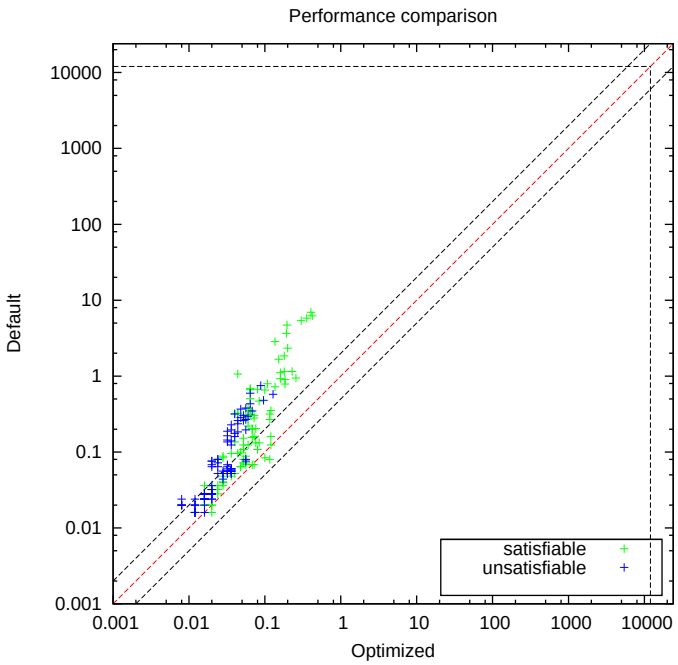


(a) training MathSAT timeout = 5s

Figure 7.5: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts



(b) training MathSAT timeout = 10s



(c) training MathSAT timeout = 15s

Figure 7.5: Performance comparison of QF_UFLRA configurations found by different training MathSAT timeouts (cont.)

7.6 Summary

Table 7.16 summaries the performance comparison of the five theories on the SMT-LIB benchmark.

In four theories of QF_IDL, QF_LIA, QF_LRA, and QF_UFLRA, the number of tests solved by the optimized configuration is equal to the number of tests solved by the *adaptive* smt-comp configuration (this configuration can be changed according to different problem classes) and is larger than the number of tests solved by the default configurations. In addition, the mean runtime is reduced by a factor of *1.231* in case of QF_IDL, by a factor of *3.677* in case of QF_LIA, and by a factor of *3.375* in case of QF_UFLRA. Only in the case of QF_LRA, the mean runtime of the optimized configuration is increased by a factor of *2.187* compared with the mean runtime of the adaptive smt-comp configuration. But if compared with the default configuration, the optimized configuration solves *7 tests more*.

In the case of QF_UFIDL, the optimized configuration solves *2 tests less* compared with the adaptive smt-comp configuration. However, if compared with the default configuration, the optimized configuration solves *11 tests more*.

Tests solved (Optimized/Default/SMTCOMP)	**95/94/95**
Mean runtime(not include TIMEOUT tests)	**5.207s/2.744s/6.410s**
The number of tests	100

(a) QF_IDL

Tests solved (Optimized/Default/SMTCOMP)	**199/199/199**
Mean runtime(not include TIMEOUT tests)	**12.069s/17.369s/44.378s**
The number of tests	200

(b) QF_LIA

Tests solved (Optimized/Default/SMTCOMP)	**200/193/200**
Mean runtime(not include TIMEOUT tests)	**19.759s/31.269s/9.033s**
The number of tests	200

(c) QF_LRA

Tests solved (Optimized/Default/SMTCOMP)	**198/187/200**
Mean runtime(not include TIMEOUT tests)	**23.804s/39.339s/30.264s**
The number of tests	200

(d) QF_UFIDL

Tests solved (Optimized/Default/SMTCOMP)	**200/200/200**
Mean runtime(not include TIMEOUT tests)	**0.056s/0.374s/0.189s**
The number of tests	200

(e) QF_UFLRA

Table 7.16: Performance Comparison of the five theories on the SMT-LIB benchmark

Chapter 8

Conclusion

The main contribution of this thesis is a comprehensive study of the most effective SMT techniques. This includes an empirical analysis approach to study the characteristics of the MathSAT configuration scenario, two experimental groups on eight and five theories to determine the best possible configurations for MathSAT on the SMT-COMP 2009 and SMT-LIB benchmark. Here, we describe these in more detail:

- We have done many experiments on a theory to determine the most suitable scenario for MathSAT before starting experiments on a set of theories. The main parameters in a scenario are the ParamILS strategy (Basic or Focused), the timeout of ParamILS (tunnerTimeout), the timeout of each MathSAT run(cutoff_time), the effect of ParamILS random seeds, the determinism of MathSAT. From the experimental results, we have concluded that *Focused ParamILS using DAE* is the most suitable ParamILS for MathSAT. As for the training time, *48 hours* is enough for the ParamILS convergence because we have experimented on the *QF_LRA*, one of the most difficult theories in the set of theories supported by MathSAT, and observed that ParamILS converged within *48 training hours* with the MathSAT timeouts less than 60s. In order to choose suitable MathSAT timeouts when training, we have run MathSAT with the default (or smt-comp) configuration on the training dataset in advance and chosen the three MathSAT timeouts (all timeouts have value from 0 to 1200 with step 5) which are less than 60 seconds and solve similar percentage of tests.

- Then, we have started ParamILS on eight theories using the SMT-COMP 2009 benchmark. In these experiments, we have used the same dataset of SMT-COMP 2009 for training and testing phases in order to check whether we could have better configurations than the default configurations and the configurations used in SMT-COMP 2009 (smt-comp configurations, these

configurations could be ***changed*** according to different problem classes based on a statistics module in MathSAT). In three theories of QF_IDL, QF_LRA, and QF_UFIDL, the number of solved tests is improved significantly. For example, in the case of QF_IDL, the optimized configuration found by ParamILS solves *14 tests more* compared with the adaptive smt-comp configuration and *10 tests more* compared with the *winner* of SMT-COMP 2009 on this theory (despite the fact that we are using the older version of MathSAT, not the new one in the competition). In two other cases of QF_LIA, and QF_UFLRA, the number of tests solved by the optimized configuration is equal to the number of tests solved by the adaptive smt-comp configuration. However, the mean runtime is reduced approximately by *half* in case of QF_LIA and by *a factor of eight* in case of QF_UFLRA. For other three theories, we have encountered some bugs when testing, and report them to the MathSAT team because we used not only the tested parameters but also *internal* parameters which are used for developers and not tested carefully.

- Next, we have used the benchmark selection tool of SMT-COMP 2009 to extract from the SMT-LIB benchmark different training and testing datasets for five successfully tested theories (no errors found in training and testing phases for these theories in Chapter 5) to find general optimized MathSAT configurations. In four theories of QF_IDL, QF_LIA, QF_LRA, and QF_UFLRA, the number of tests solved by the optimized configuration is equal to the number of tests solved by the *adaptive* smt-comp configuration and is larger than the number of tests solved by the default configurations. In addition, the mean runtime is reduced by a factor of *1.231* in case of QF_IDL, by a factor of *3.677* in case of QF_LIA, and by a factor of *3.375* in case of QF_UFLRA. Only in the case of QF_LRA, the mean runtime of the optimized configuration is increased by a factor of *2.187* compared with the mean runtime of the adaptive smt-comp configuration. But if compared with the default configuration, the optimized configuration solves *7 tests more*. In the case of QF_UFIDL, the optimized configuration solves *2 tests less* compared with the adaptive smt-comp configuration. However, if compared with the default configuration, the optimized configuration solves *11 tests more*.

Chapter 9

List of Acronyms

- **RAE** Random Algorithm Evaluation (used in ParamILS).
- **DAE** Deterministic Algorithm Evaluation (used in ParamILS).

Bibliography

[1] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In AAAI, pages 11521157. AAAI Press, 2007. 26

[2] Diplomarbeit In Englischer Sprache, Von Frank Hutter, and Betreuer Dr. Thomas Stützle. Stochastic local search for solving the most probable explanation problem in bayesian networks, 2004. 26, 33

[3] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An Automatic Algorithm Conguration Framework. In Journal of Artificial Intelligence Research 36 (2009) 267-306, 2009. 21

[4] Roberto Sebastiani. Lazy Satisability Modulo Theories. JSAT, Vol. 3, Number 3-4, pp. 141-224, 2007. 7, 8, 12, 13, 15, 17, 19

[5] Alberto Griggio. An effective SMT engine for Formal Verification. Ph.D. Thesis, DISI - University of Trento. 13

[6] Anders Franzn. Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT. Ph.D. Thesis, DISI - University of Trento. 14

[7] Jinbo Huang. The Effect of Restarts on the Efficiency of Clause Learning. IJCAI, pp. 2318-2323, 2007. 18

[8] Armin Biere. Adaptive Restart Strategies for Conflict Driven SAT Solvers. SAT, Lecture Notes in Computer Science, Vol. 4996, pp. 28-33, Springer, 2008. 19

[9] Alessandro Cimatti, Alberto Griggio and Roberto Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. ACM Transaction on Computational Logics, TOCL. To appear. Available from TOCL accepted papers web page. 17, 20

[10] Dutertre, B., de Moura, L.: The Yices SMT Solver. http://yices.csl.sri.com/tool-paper.pdf (2006) 15

[11] Niklas Sorensson, Niklas Een. Minisat 1.13 system description, http://www.minisat.se/downloads/MiniSat_v1.13_short.pdf. 15

[12] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, Cesare Tinelli. Satisfiability Modulo Theories. Part II, Chapter 26, The Handbook of Satisfiability. 2009. IOS press. 8

[13] Jan-Willem Roorda and Koen Claessen. SAT-Based assistance in abstraction refinement for symbolic trajectory evaluation. In Computer Aided Verification, pages 175-189, Springer-Verlag, 2006. 13

[14] Bozzano, M., Bruttomsesso, R., Cimatti, A., Junttila, T., Van Rossum, P., Ranise, S., And Sebastiani, R.. Efficient Theory Combination via Boolean Search. Information and Computation 204, 10(October), 1411-1596, 2006. 20

[15] Lourenco, H. R., Martin, O., & Stützle, T. Iterated local search. In F. Glover & G. Kochenberger (Eds.), Handbook of Metaheuristics (pp. 321-353). Kluwer Academic Publishers, Norwell, MA, USA, 2002. 22

[16] Carsten Sinz and Markus Iser. Problem-Sensitive Restart Heuristics for the DPLL Procedure. SAT, Lecture Notes in Computer Science, Vol. 5584, pp. 356-362, Springer, 2009. 18

[17] http://goedel.cs.uiowa.edu/smtlib/ 57

[18] W. Ackermann. Solvable Cases of the Decision Problem. North Holland Pub. Co., Amsterdam, 1954. 14

[19] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In Proc. European Conference on Planning, CP-99, 1999. 7, 9, 10, 13, 19

[20] A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. In Proc. SAT04, 2004. 7

[21] G. Audemard, P. Bertoli, A. Cimatti, A. Korni lowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In Proc. CADE'2002., volume 2392 of LNAI. Springer, July 2002. 9, 10, 13, 17, 18

[22] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In Proc. BMC04, volume 89/4 of ENTCS. Elsevier, 2003. 7

[23] G. Audemard, A. Cimatti, A. Korni lowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In Proc. FORTE02., volume 2529 of LNCS. Springer, November 2002. 7, 19

[24] T. Ball, B. Cook, S. K. Lahiri, and L. Zhang. Zapato: Automatic Theorem Proving for Predicate Abstraction Refinement. In Proc. CAV04, volume 3114 of LNCS. Springer, 2004. 7

[25] C. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In Proceedings of the 16th International Conference on Computer Aided Verification (CAV 04), volume 3114 of LNCS. Springer, 2004. 7

[26] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In Proc. CAV05, volume 3576 of LNCS. Springer, 2005. 9

[27] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In 14th International Conference on Computer-Aided Verification, 2002. 9

[28] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostaks method for combining decision procedures. In Frontiers of Combining Systems (FROCOS), LNAI. Springer, April 2002. S. Margherita Ligure, Italy. 8

[29] C. W. Barrett, Y. Fang, B. Goldberg, Y. Hu, A. Pnueli, and L. D. Zuck. TVOC: A Translation Validator for Optimizing Compilers. In Proc. CAV05, volume 3576 of LNCS. Springer, 2005. 7

[30] R. J. Bayardo and R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT instances. In Proc. AAAI97, pages 203208. AAAI Press, 1997. 8, 9, 16

[31] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In Proc. CAV99, 1999. 8

[32] A. Boralv. A Fully Automated Approach for Proving Safety Properties in Inter- locking Software Using Automatic Theorem-Proving. In Proceedings of the Second International ERCIM Workshop on Formal Methods for Industrial Critical Systems, 1997. 8

[33] M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzen, Z. Hanna, Z. Khasidashvili, A Palti, and R. Sebastiani. Encoding RTL Constructs for MathSAT: a Preliminary Report. In Proc. PDPAR05, volume 144 of ENTCS. Elsevier, 2006. 7, 20

[34] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In Proc. TACAS05, volume 3440 of LNCS. Springer, 2005. 7, 9

[35] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. MathSAT: A Tight Integration of SAT and Mathematical Decision Procedure. Journal of Automated Reasoning, 35(1-3), October 2005. 9, 17, 19

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

[37] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In Proc. ASP-DAC 2002, pages 741-746. IEEE, 2002. 20

[38] R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and Verifying Systems Using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions. In Proc. CAV02, volume 2404 of LNCS. Springer, 2002. 9

[39] J. R. Burch and D. L. Dill. Automatic Verification of Pipelined Microprocessor Control. In Proc. CAV 94, volume 818 of LNCS. Springer, 1994. 7

[40] C. Castellini, E. Giunchiglia, and A. Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. Artificial Intelligence, 147(12):85-117, 2003. 8

[41] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. Journal of the ACM, 5(7), 1962. 9

[42] M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of the ACM, 7:201-215, 1960. 9

[43] L. de Moura and N. Bjrner. System Description: Z3 0.1. In 3rd Int. Competition of Satisfiability Modulo Theories tools, 2007. http://research.microsoft.com/projects/z3/smtcomp07.pdf. 7

[44] L. de Moura and H. Ruess. An Experimental Evaluation of Ground Decision Procedures. In Proc. CAV04, volume 3114 of LNCS. Springer, 2004. 9

[45] L. de Moura, H. Rue, and M. Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In Proc. of the 18th International Conference on Automated Deduction, volume 2392 of LNCS, pages 438-455. Springer, July 2002. 7, 9

[46] D. Detlefs, G. Nelson, and J. Saxe. Simplify: a theorem prover for program checking. Journal of the ACM, 52(3):365-473, 2005. 7

[47] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In Proc. CAV06, volume 4144 of LNCS. Springer, 2006. 7

[48] N. Een and A. Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In proc. SAT05, volume 3569 of LNCS. Springer, 2005. 8, 9

[49] N. Een and N. Sorensson. An extensible SAT-solver. In Theory and Applications of Satisfiability Testing (SAT 2003), volume 2919 of LNCS, pages 502-518. Springer, 2004. 8, 9, 11

[50] J.C: Filliatre, S. Owre, H. Rue, and N. Shankar. ICS: Integrated Canonizer and Solver. Proc. CAV2001, 2001. 8

[51] C. Flanagan, R. Joshi, X. Ou, and J. B. Saxe. Theorem Proving Using Lazy Proof Explication. In Proc. CAV 2003, LNCS. Springer, 2003. 7, 9, 10

[52] Anders Franzen. Using Satisfiability Modulo Theories for Inductive Verification of Lustre Programs. In Proc. BMC05, volume 144 of ENTCS. Elsevier, 2006. 7

[53] M. K. Ganai, M. Talupur, and A. Gupta. SDSAT: Tight integration of small domain encoding and lazy approaches in a separation logic solver. In Proc. TACAS06, volume 3920 of LNCS. Springer, 2006. 7

[54] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast Decision Procedures. In Proc. CAV04, volume 3114 of LNCS. Springer, 2004. 9, 10, 13

[55] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT Based Decision Procedures for Classical Modal Logics. Journal of Automated Reasoning. Special Issue: Satisfiability at the start of the year 2000, 2001. 17

[56] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In Proc. AAAI98, pages 948-953, 1998. 11

[57] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In CADE-13, LNAI, New Brunswick, NJ, USA, August 1996. Springer Verlag. 9, 10

[58] E. Goldberg and Y. Novikov. BerkMin: A Fast and Robust SAT-Solver. In Proc. DATE 02, page 142, Washington, DC, USA, 2002. IEEE Computer Society. 8, 9, 11

[59] J. Hoffmann and R. I. Brafman. Contingent Planning via Heuristic Forward Search witn Implicit Belief States. In Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), pages 71-80. AAAI, 2005. 8

[60] John N. Hooker and V. Vinay. Branching Rules for Satisfiability. Journal of Auto- mated Reasoning, 15(3):359-383, 1995. 11

[61] I. Horrocks and P. F. Patel-Schneider. Optimising propositional modal satisfiability for description logic subsumption. In Proc. AISC98, volume 1476 of LNAI. Springer, 1998. 9

[62] R.G. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. Annals of Mathematics and Artificial Intelligence, 1(1-4):167-187, 1990. 11

[63] H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In Proc. KR96, 1996. 8

[64] H. Kim and F. Somenzi. Finite Instantiations for Integer Difference Logic. In proc FMCAD06. ACM Press, 2006. 7

[65] S. K. Lahiri and S. A. Seshia. The UCLID Decision Procedure. In Proc. CAV04, volume 3114 of LNCS, 2004. 7

[66] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 366-371, 1997. 11

[67] M. Mahfoudh, P. Niebert, E. Asarin, and O. Maler. A Satisfibaility Checker for Difference Logic. In Proceedings of SAT-02, pages 222-230, 2002. 7, 9

[68] K. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In Proc. CAV 02, volume 2404 of LNCS. Springer, 2002. 8

[69] M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In Design Automation Conference, 2001. 8, 9, 11

[70] C. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. TOPLAS, 1(2):245-257, 1979. 8

[71] G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. Journal of the ACM, 27(2):356-364, 1980. 8

[72] R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In Proc. CAV05, volume 3576 of LNCS. Springer, 2005. 7, 9, 13

[73] D. C. Oppen. Complexity, Convexity and Combinations of Theories. Theoretical Computer Science, 12:291-302, 1980. 8

[74] D.C. Oppen. Reasoning about Recursively Defined Data Structures. Journal of the ACM, 27(3):403-411, 1980. 8

[75] G. Parthasarathy, M. K. Iyer, K.-T. Cheng, and L.-C. Wang. An efficient finite-domain constraint solver for circuits. In Proc. DAC04. ACM Press, 2004. 7

[76] S. Ranise and D. Deharbe. Light-Weight Theorem Proving for Debugging and Verifying Units of Code-. In Proc. of the International Conference on Software Engineering and Formal Methods SEFM03. IEEE Computer Society Press, 2003. 7

[77] S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). http://www.SMT-LIB.org, 2006. 8

[78] S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, Department of Computer Science, The University of Iowa, 2006. Available at http://www.SMT-LIB.org. 8

[79] S. A. Seshia, S. K. Lahiri, and R. E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In Proc. DAC03, 2003. 9, 19

[80] N. Shankar and Harald Rue. Combining shostak theories. Invited paper for Floc02/RTA02, 2002. 8

[81] H. M. Sheini and K. A. Sakallah. A Scalable Method for Solving Satisfiability of Integer Linear Arithmetic Logic. In Proc. SAT05, volume 3569 of LNCS. Springer, 2005. 7

[82] H. M. Sheini and K. A. Sakallah. From Propositional Satisfiability to Satisfiability Modulo Theories. Invited lecture. In Proc. SAT06, volume 4121 of LNCS. Springer, 2006. 9

[83] R. Shostak. A Pratical Decision Procedure for Arithmetic with Function Symbols. Journal of the ACM, 26(2):351-360, 1979. 8

[84] R.E. Shostak. Deciding Combinations of Theories. Journal of the ACM, 31:1-12, 1984. 8

[85] J. P. M. Silva and K. A. Sakallah. GRASP - A new Search Algorithm for Satisfiability. In Proc. ICCAD96, 1996. 8, 9

[86] SMT-COMP05: 1st Satisfiability Modulo Theories Competition, 2005. http://www.csl.sri.com/users/demoura/smt-comp/2005/. 9

[87] SMT-COMP06: 2nd Satisfiability Modulo Theories Competition, 2006. http://www.csl.sri.com/users/demoura/smt-comp/. 9

[88] G. Stalmarck and M. Saflund. Modelling and Verifying Systems and Software in Propositional Logic. Ifac SAFECOMP90, 1990. 8

[89] P. Stephan, R. Brayton, , and A. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 15:1167-1176, 1996. 8

[90] O. Strichman. Tuning SAT checkers for Bounded Model Checking. In Proc. CAV00, volume 1855 of LNCS, pages 480-494. Springer, 2000. 11

[91] O. Strichman. On Solving Presburger and Linear Arithmetic with SAT. In Proc. of Formal Methods in Computer-Aided Design (FMCAD 2002), LNCS. Springer, 2002. 9

[92] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In Proc. of Computer Aided Verification, (CAV02), LNCS. Springer, 2002. 9, 19

[93] C. Tinelli. A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In Proc. JELIA-02, volume 2424 of LNAI, pages 308-319. Springer, 2002. 9, 13

[94] M. N. Velev and R. E. Bryant. Exploiting Positive Equality and Partial Non-Consistency in the Formal Verification of Pipelined Microprocessors. In Design Automation Conference, pages 397-401, 1999. 9

[95] S. Wolfman and D. Weld. The LPSAT Engine and its Application to Resource Planning. In Proc. IJCAI, 1999. 7, 9, 10, 17

[96] Y. Yu and S. Malik. Lemma Learning in SMT on Linear Constraints. In Proc. SAT06, volume 4121 of LNCS. Springer, 2006. 19