CONFLICT RESOLUTION

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2010

By
Nestan Tsiskaridze
School of Computer Science

Contents

Abstract 11

Declaration 13

Copyright 15

Acknowledgements 17

1 Introduction 19

Motivation . . . 19

Main Contribution . . . 20

A guide for Readers . . . 21

2 Background and Related Work 23

2.1 Basic Notations . . . 23

2.2 Statement of the Problem . . . 25

2.3 Historical Background and Applications . . . 26

2.4 Related Work . . . 28

2.4.1 The Fourier-Motzkin Elimination Method . . . 29

2.4.2 Modifications of the Fourier-Motzkin Method . . . 31

2.4.3 The Simplex Method . . . 32

2.4.4 The Relaxation Method . . . 33

3 Conflict Resolution 35

3.1 Algorithm Description . . . 35

3.2 Correctness and Termination . . . 41

3.3 Properties of the CRA Algorithm . . . 42

3.4 Extensions of the CRA Algorithm . . . 45

3.4.1 Extensions With $\{>, =\}$ . . . 45

3.4.2 Conflict Resolution and Linear Programming 47
3.4.3 Conflict Resolution and the Fourier-Motzkin Method . . . 47
3.4.4 Conflict Resolution and Satisfiability Modulo Theories . . 49

4 Implementation 51
4.1 Data Structure 51
4.1.1 Parser 52
4.1.2 Levels 54
4.1.3 Order 55
4.1.4 Processing Input Constraints 58
4.2 Implementation 59
4.2.1 Phase I - Early Stage Implementation 60
4.2.2 Phase II - Implementation with Various Heuristics 61
4.2.2.1 Strategies for Selecting Conflicts 61
4.2.2.2 Strategies for Selecting Assignment Values 62
4.2.2.3 Other Heuristics 63
4.2.3 Phase III - Simple Preprocessing with Highly-Tuned Heuristics 64
4.2.3.1 Strategies for Setting the Order 65
4.2.3.2 Preprocessing to Avoid Half-Bounding Levels . . 65
4.2.3.3 Preprocessing to Avoid Almost Half-Bounding Levels 66

5 Experiments 69
5.1 Benchmarks with Randomly Generated Problems 70
5.2 Benchmarks Extracted From SMT-LIB With Hard Reality Tool . 71
5.3 Experimental Results 73
5.3.1 Phase I - Early Stage Implementation 73
5.3.1.1 Randomly Generated Problems 73
5.3.1.2 Real-Life Problems 75
5.3.2 Phase II – Implementation with Various Heuristics 76
5.3.2.1 Randomly generated benchmarks 79
5.3.2.2 Real-Life Benchmarks 85
5.3.3 Phase III – Simple Preprocessing with Highly-Tuned Heuristics 93

6 Conclusions 97
6.1 Thesis Achievements 97
6.2 Significance of the Research 98
6.3 Future Work 99

Bibliography 103

List of appendices 109

Appendix A 111

Appendix B 115

Appendix C 117

Appendix D 119

Index 160

Word Count 34.393

List of Tables

5.1 Randomly Generated Problems 74
5.2 Hard Reality Problems 75
5.3 CRA vs Barcelogic on real-life HR benchmarks 76
5.4 List of the bundles of heuristics of the CRA algorithm 80
5.5 CRA vs CRA with simple preprocessing on the third set of real-life benchmarks 93
5.6 CRA with simple preprocessing vs Barcelogic, CVC3, Z3 on the first and the second sets of real-life benchmarks. 94

List of Figures

4.1 Implementation Schema 52

4.2 Data Structure Design 54

4.3 Data Structure Design with Levels 56

4.4 Sorting Input Constraints 59

5.1 A randomly generated problem 75

5.2 Implementations of the CRA for the major heuristics 'FC' on randomly generated set of benchmarks, 'number of variables versus number of solved problems'. 81

5.3 Implementations of the CRA for the major heuristics 'MAX' on randomly generated set of benchmarks, 'time versus number of problems solved' 82

5.4 Implementations of the CRA for the major heuristics 'RC' on randomly generated set of benchmarks, 'time versus number of problems solved' 83

5.5 Implementations of CRA with the best performance of all major heuristics for the randomly generated set of benchmarks, 'time versus number of problems solved' 84

5.6 A quadruple of implementations of CRA with 'MP' heuristics for the first set of benchmarks, 'time versus number of problems solved' 86

5.7 A quadruple of implementations of CRA with 'MP' heuristics for the second set of benchmarks, 'time versus number of problems solved' 86

5.8 A quadruple of implementations of CRA with 'FC' heuristics for the third set of benchmarks, 'time versus number of problems solved' 87

5.9 A quadruple of implementations of CRA with 'MIN' heuristics for the third set of benchmarks, 'time versus number of problems solved' 88

5.10 Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, 'number of variables versus number of solved problems' 88

5.11 Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, 'time versus number of problems solved' 89

5.12 Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, 'total run-time versus number of solved problems' 90

5.13 Implementations of CRA with the best performance of all major heuristics for the second set of benchmarks, 'number of variables versus number of solved problems' 90

5.14 Implementations of CRA with the best performance of all major heuristics for the third set of benchmarks, 'number of variables versus number of solved problems' 91

5.15 Implementations of CRA with the best performance of all major heuristics for the third set of benchmarks, 'total run-time versus number of solved problems' 92

5.16 Benchmarks where CRA with simple preprocessing and Z3 showed different performances, 'time versus number of problems solved' . 95

Abstract

This thesis proposes a new method for solving systems of linear constraints over the rational and real numbers (or, equivalently, linear programming) – the *conflict resolution method*. The method is a new approach to a classic problem in mathematics and computer science, that has been known since the 19th century. The problem has a wide range of real-life applications of increasing importance in both academic and industrial areas. Although, the problem has been a subject of intensive research for the past two centuries only a handful of methods had been developed for solving it. Consequently, new results in this field may be of a particular value, not mentioning the development of new approaches.

The motivation of our research did not arise solely from the field of linear programming, but rather was instantiated from problems of *Satisfiability Modulo Theories* (or shortly *SMT*). SMT is a new and rapidly developing branch of automated reasoning dedicated to reasoning in first-order logic with (combination) of various theories, such as, linear real and integer arithmetic, theory of arrays, equality and uninterpreted functions, and others.

The role of linear arithmetic in solving SMT problems is very significant, since a considerable part of SMT problems arising from real-life applications involve theories of linear real and integer arithmetic. Reasoning on such instances incorporates reasoning in linear arithmetic. Our research spanned the fields of SMT and linear programming.

We propose a method, that is not only used for solving linear programming problems, but also is well-suited to SMT framework. Namely, there are certain requirements imposed on theory reasoners when they are integrated in SMT solving. Our conflict resolution method possesses all the attributes necessary for integration into SMT. As the experimental evaluation of the method has shown, the method is very promising and competitive to the existing ones.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would like to thank my supervisor Andrei Voronkov, who changed my life by offering me this opportunity to continue my studies at Manchester University, for suggesting an extremely interesting topic as a research direction, and continuously helping, encouraging and understanding during the whole period of my studies. From him I have learnt a lot and could always rely in tough times through my PhD years. My most sincere gratitude to him for being the best ever supervisor for me.

I would like to thank Konstantin, who has been always ready to help, for his ideas, and constant attention, without whom the conflict resolution method might not have come to life. It has been a great pleasure to work teamed up with him and Andrei.

I am very grateful to my father, for encouraging me to continue study abroad, for invaluable support and being aside despite the distance. Also, I am grateful to Joseph and my siblings for their best support and encouragement. And along with them, I would like to thank my mum for being an irreplaceable part of my life, without whom I would never be where I am.

I would also like to thank the Department of Computer Science in Manchester University, for all the support I have been getting from them, for awarding me a best paper award, for everyday care and understanding, for granting me a scholarship and making it possible for me to do my PhD.

Special thanks to Birte for being such a beautiful precious friend to me from the very first days of my stay in the UK. Special thanks to Irakli for his constant encouragement and for believing that I can make it. Also, I thank Eva and Rina for being always caring and sharing beautiful times together. Great thanks to Simon for being there to support me, also, to Pavel, Krystof, Juan and my office mates. Special thanks to Veronica for her constant encouragement and for making the last year of my PhD so cosy and colourful by being by my side every day.

There are many more people, who I like to thank, my friends in Georgia, in Britain and around the globe, whom I am so happy to have in my life.

Chapter 1

Introduction

Motivation. This research has been motivated by solving problems of *Satisfiability Modulo Theories (SMT)* [3]. SMT is a new branch of automated reasoning dedicated to deciding satisfiability of formulae in first-order logic with respect to various theories. SMT is a rapidly developing area of research. It has large variety of real life applications, such as, software and hardware verification problems and constraint satisfaction [12, 49, 10, 32, 43, 48]. Typical instances of SMT problems may be presented as a system of thousands of constraints like:

$$p \vee \neg q \vee f(g(x_1)) \neq x_3 \vee g(x_1 - x_2) = f(x_3) \vee x_1 + x_2 \leq 3$$

This formula contains propositional atoms, and atoms over the combination of two theories: *linear real and integer arithmetic* and *equality with uninterpreted functions*. More examples of theories involved in SMT problems include theory of arrays and list structures, and bit-vectors and others.

Deciding satisfiability of systems of such formulae concerns finding a satisfying assignment to all variables occurring in the system. If such an assignment does not exist the SMT instance is unsatisfiable. Solving such systems directly involves operation on sets of constraints over the theories presented in the formulae. Namely, these operations concern deciding satisfiability of sets of constraints within these theories.

A significant part of SMT problems involve theories of linear real and integer arithmetic. These instances draw our interest and we took aim at finding new methods of integrating linear arithmetic reasoning in SMT solving. Reasoning in linear real (integer) arithmetic implies deciding satisfiability of systems of linear constraints, i.e. solving systems of linear constraints.

The problem of solving systems of linear constraints, itself, is a classic problem in mathematics and has been known for several centuries. However, there are only a handful of methods for solving it: the Fourier-Motzkin elimination method of the 19th century and the simplex and interior point methods of the 20th century.

An example of a simple system with linear constraints over the reals is given below.

$$\begin{array}{rcrcrcrcl} 3x_3 & + & 4x_2 & & & - & 1 & > & 0 \\ -x_3 & + & x_2 & & & + & 1 & \geq & 0 \\ & & 2x_2 & - & x_1 & & & = & 0 \end{array}$$

This system consists of 3 variables and 3 constraints and is satisfiable when the variables are assigned the values: $x_1 = 2, x_2 = 1, x_3 = 1$.

In real-life examples systems are much bigger, with hundreds and thousands of constraints and inequalities. Thus, real-life problems are more complex.

This problem has been proved to be equivalent to a so called *linear programming problem.* A linear programming problem is a problem of optimising a given linear function with respect to a system of linear constraints. The following is an example of a simple linear programming problem, extracted from [7]. The problem consists of 3 variables, 4 constraints and an objective function to be maximised.

Maximise: $5x_3 + 5x_2 + 3x_1$
subject to:

$$\begin{array}{rcrcrcrcl} -x_3 & - & 3x_2 & - & x_1 & + & 3 & \geq & 0 \\ x_3 & & & - & 3x_1 & + & 2 & \geq & 0 \\ -2x_3 & + & x_2 & - & 2x_1 & + & 4 & \geq & 0 \\ -2x_3 & - & 3x_2 & + & x_1 & + & 2 & \geq & 0 \\ & & & & x_3, & x_2, & x_1 & \geq & 0 \end{array}$$

This problem has an optimal solution equal to 10.

Eventually, the motivation of this research was to find new ways of operating with systems of linear constraints that would be friendly to, and integrate well with, SMT solving methods. The research resulted in developing a new method for solving systems of linear constraints over the rational and real numbers – the *conflict resolution method* [30].

Main Contribution. This thesis is devoted to our conflict resolution method. The method is new and has a number of properties crucial for integration into SMT solving, but may also have an independent value for linear optimisation and a number of other combinatorial problems.

The method works with a set of constraints and an assignment on variables. It iteratively refines the assignment trying to make it into a solution. If such a refinement is impossible, it is due to a pair of conflicting constraints. We resolve the conflict by deriving a new constraint and adding it to the system. Adding new constraints is done in such a way, that the expanded system has the same set of solutions as the initial system. The refinements are done successively until either the assignment becomes a solution of the system or the initial system is expanded to a system containing a contradiction. Generation of new constraints ensures that new constraints are non-redundant in some natural sense.

The use of the technique developed for resolving conflicts in our method makes the method similar to the Fourier-Motzkin variable elimination method.

We implemented a solver based on our method and compared it with versions of the Fourier-Motzkin elimination method and the simplex method incorporated in the existing SMT solvers, as well as with our implementations of the Fourier-Motzkin elimination method and its most prominent modification – the Chernikov algorithm. As the experiments show our solver is highly competitive with existing implementations of the linear arithmetic solvers integrated into the state-of-the-art SMT solvers, and in some cases even outperforms some of them.

In this chapter we mainly aimed to introduce readers to the motivation behind our research and explain on an intuitive level the essence of the proposed method. The thesis discusses in detail the method, its implementation and experiments, along with giving an overview of the historical background and related work. A thesis guide for readers is provided in the next section.

We presented the conflict resolution method in 2009 at the *Conference on Principles and Practice of Constraint Programming (CP 2009)*, where our paper was awarded *a Runner-Up as the Best Paper Prize.* The paper has also been granted *the Best PhD Paper Award 2009 at the School of Computer Science at the University of Manchester.*

A Guide for Readers. The thesis is structured as follows. Chapter 2 defines basic notions used throughout the thesis, formulates the problem, gives a brief historical overview of the background of the problem and its applications. It briefly introduces the related work, including the Fourier-Motzkin variable elimination method, its modification the Chernikov algorithm, the simplex method and the relaxation method. In Chapter 3 we describe the conflict resolution method, prove correctness and termination of the algorithm and discuss its properties along with some extensions and its usability in SMT. In Chapter 4 we discuss in detail our implementation of the conflict resolution algorithm. In Chapter 5 we present the results of our experimental evaluation of the algorithm. Finally, in Chapter 6 we summarise thesis achievements and discuss further research directions.

Chapter 2

Background and Related Work

In this chapter, we formulate the problem of solving systems of linear constraints, introduce basic notations used throughout the thesis. We also give a brief overview of the historical background, importance of the problem, its applications and related work.

2.1 Basic Notations

Throughout the thesis we consider constraints and variables over the field $\mathbb{Q}$ of rational numbers. This is the most important and frequently considered case, which allows conducting exact calculation without rounding errors. Nevertheless, the algorithmic part of the thesis remains valid for any sub-field $\mathbb{F}$ of a field of real numbers $\mathbb{R}$ (including $\mathbb{R}$). Let $\mathbb{Q}$ denote the set of rationals. Let n be a positive integer, we denote by X a finite set of variables $\{x_1, \ldots, x_n\}$.

Throughout the thesis we assume that all the variables have rational values, i.e. $x_i \in \mathbb{Q}$ for $1 \leq i \leq n$. An expression $a_n x_n + \ldots + a_1 x_1 + b \diamond 0$, where $a_i \in \mathbb{Q}$ for $1 \leq i \leq n$ and $\diamond$ denotes one of the binary relations $\{\geq, >, =, \neq\}$ is called a $\mathbb{Q}$-*linear constraint over* X. When all coefficients $a_1, a_2, \ldots a_n$ are equal to 0 the constraint becomes $b \diamond 0$, which does not depend on variables $x_1, x_2, \ldots, x_n$ and is either *true* or *false*. The constraint that is identically true we denote by $\top$, and the constraint that is identically false we denote by $\bot$. Such constraints are called *trivial*. For brevity, in the sequel we will call such rational linear constraints over X simply *linear constraints*. We call a *system of linear constraints* any finite set of linear constraints.

Let $\succ$ be a total order on X. To simplify the notations, without loss of generality we may assume that $x_n \succ x_{n-1} \succ \ldots \succ x_1$. We will say x_i is *higher* than x_j (or x_j is *lower* than x_i) if $x_i \succ x_j$. We call x the *highest variable* in a constraint if x is higher than any other variable present in this constraint.

A constraint is called *normalised* if its highest variable has a coefficient 1 or -1. Evidently, the constraints $\bot$, $\top$, that do not contain variables are normalised. If x is the highest variable in a constraint the normalised form of this constraint should be of one of the forms: $x + p \diamond 0$ or $-x + q \diamond 0$, where p and q contain only variables lower than x, and $\diamond$ is as defined above. If the highest variable x has a coefficient $a \neq 0$, than the constraint can be normalised by dividing all its coefficients by $|a|$. Evidently, every constraint can be effectively normalised, i.e. transformed into an equivalent normalised constraint. In the sequel, we assume that all constraints are normalised.

We define an *assignment* σ over the set of variables X as a mapping from X to $\mathbb{Q}$, i.e. $\sigma : X \to \mathbb{Q}$. Given an assignment σ and a variable $x \in X$ we call $v = \sigma(x)$ a *value* of the variable x under the assignment σ. Thus all variables $x_i \in X$ $(1 \leq i \leq n)$ can obtain respective values $v_i = \sigma(x_i)$ under the assignment σ.

In what follows the notation $x\sigma$ simply means $\sigma(x)$. Given an assignment σ, a variable $x \in X$ and an arbitrary number $v \in \mathbb{Q}$ we can *update* σ *at* x *by* v, denoted by σ_x^v, by changing the value of x in the assignment σ to v and leaving the values of σ for all the other variables unchanged.

For a linear form q over X, we denote by $q\sigma$ the value of q after replacing all variables $x \in X$ by their values under the assignment $\sigma(x)$. An assignment σ is called a *solution of a linear constraint* $q \diamond 0$ if $q\sigma \diamond 0$ is a true numeric relation; σ is a *solution of a system* S of linear constraints if it is a solution of every constraint in the system S. If σ is a solution of a linear constraint c (or a system S of such constraints), we also say that σ *satisfies* c (respectively, S), denoted by $\sigma \models c$ (respectively, $\sigma \models S$), otherwise we say that σ *violates* c (respectively, S). Thus, if σ violates a system of constraints S, it violates at least one constraint contained in S. A system of linear constraints is said to be *satisfiable* if there exists an assignment σ that satisfies S, otherwise it is said to be *unsatisfiable*.

To focus on the main ideas, all algorithms described in the thesis will be initially considered for solving systems of linear constraints of the form $q \geq 0$. The general case will be discussed later in Chapter 3.

2.2 Statement of the Problem

The problem of solving systems of linear constraints (essentially, equivalent to the linear programming problem) is a classic problem in mathematics as well as in computer science. It concerns deciding the satisfiability of a system of linear constraints. Satisfiability is decided by either finding an assignment to all variables that satisfies the system, or proving that such an assignment does not exist.

Geometrically solutions of a linear constraint $a_nx_n+\ldots+a_1x_1+b=0$ comprise an $(n-1)$-dimensional plane, so called *hyperplane* in $\mathbb{Q}^n$. Replacing equality by a non-strict inequality $a_nx_n+\ldots+a_1x_1+b \geq 0$ the set of solutions expands to a half-space bounded by this hyperplane. A system of linear constraints geometrically represents an intersection of the half-spaces defined by these constraints. A geometric object, shaped by the intersection of finitely many half-spaces is called a *convex polyhedron*. This polyhedron shapes the set of all solutions of the system. If the polyhedron is empty the system has no solutions.

The problem of solving a system of linear constraints is equivalent to a problem of optimisation of a given linear function subject to a system of linear constraints. Such an optimisation problem is called a *linear programming (LP)* problem. The linear function to be optimised is called an *objective function*. These two problems are equivalent in sense that any method for solving one of them directly yields the method for solving the other one. A succinct proof of this statement can be found in [44].

The geometric interpretation of a linear programming problem is also related to a notion of polyhedron. The system of linear constraints defines a convex polyhedron. The problem concerns finding a point in the polyhedron for which the value of the objective function is optimal. Obviously, the linear programming problem has no solution if the polyhedron is empty. It also has no solution if the polyhedron is unbounded towards the direction of the optimisation of the objective function.

2.3 Historical Background and Applications

The problem of solving systems of linear constraints dates back to the early 19th century when Fourier, a French mathematician and physicist, suggested the first method for solving it [17]. The method, however, had been forgotten for almost a century and was rediscovered by Danes [13] and Motzkin [38] in the early 20th century. The method is known as *Fourier-Motzkin elimination method.*

The Fourier-Motzkin method has number of properties that distinguish it from other methods developed, however its performance in practice is poor compared to other methods.

The traces of the idea of the linear programming were observed already in Fourier's works [16, 17]. But as a discipline linear programming was established in 1940's by the works of Dantzig, Kantorovich, Koopmans, and von Neumann.

In 1939 Kantorovich formulated certain practical problems for organising and planning production and transportation purposes in terms related to linear programming and proposed a method for solving them [24]. However, his works were not recognised until the establishment of linear programming as a new theory in works by Dantzig and others. Two decades later Koopmans, showed that the problems formulated by Kantorovich are equivalent to the general linear programming problem, and the method proposed by him was actually a method for general linear programming [29].

The work by Koopmans [28], as of 1947, was the second rediscovery of linear programming. He gave a linear programming formulation to the problem of optimum allocation of the resources in a transportation system and presented a method for solving this problem. His work illustrated the successful utilisation of the linear programming framework in classical economic theories.

In 1975 Kantorovich and Koopmans were jointly awarded the Nobel Prize in Economic Science 'for their contribution to the theory of optimum allocation of resources'.

However, the work by Dantzig [8], presented in 1947, was the one that decided the success of linear programming in practice, led to the recognition of its importance, and originated it as a new branch of applied mathematics. He introduced the *simplex method* for solving linear programming problems. The idea behind the simplex method originates from Fourier's works [16] and was put into algorithmic shape by Dantzig in 1951 [8].

Dantzig's work was followed by a rapid growth of real-life applications of linear programming in wast variety of areas that seemed of no relevance at first sight. Such an advance in applications influenced further development of linear programming as a new discipline itself.

One should also mention here von Neumann, who laid the mathematical fundamentals of linear programming in 1947, and first introduced the important notion of Duality in linear programming [50].

Simplex is very efficient (almost linear) in practice and has a polynomial behaviour in probability. However, its worst case complexity is exponential and there is no version of the simplex method that shows polynomial time complexity in all instances.

For a long time it was an open question whether there exists a polynomial time algorithm for linear programming, or for the equivalent problem of solving systems of linear constraints. This question was answered in 1979 by Khachiyan [26],

He showed that the *ellipsoid method*, developed by Shor [46, 45, 47] and Yudin and Nemirovskii [51, 52] for nonlinear programming, can be successfully adopted for solving linear programming problems and systems of linear constraints in polynomial time.

Evidently, Khachiyan's method was a significant theoretical breakthrough, however it turned out to perform poorly in practice.

The success of Khachiyan was followed by a larger breakthrough of both theoretical and practical value. In 1984 Karmarkar introduced his *interior point method* for solving linear programming problems [25].

Since Karmarkar's first interior point method, many interior point methods have been developed and analysed in the last three decades. Success of the early implementations was based on the affine scaling variants of Karmarkar's method. The most common among them are the *barrier function* method by Nocedal and Wright in 1999 [41] and the *path-following method* by Mehrotra in 1989 [34].

Even though both Khachiyan's and Karmarkar's methods are polynomial time, the number of iterations, i.e. the number of arithmetic operations performed, depends on the size of numbers in the input data. It remains an open question, whether there exists an algorithm for linear programming that is *strongly polynomial* – i.e. makes a polynomial number of elementary arithmetic operations, and this number depends only on the dimension of the problem (number of variables and number of constraints). Elementary operations here are addition, subtraction, multiplication, division, comparison.

A thorough survey of the history of linear programming can be found in Dantzing's monograph on linear programming of 1963 [9] and in a book by Schrijver on theory of linear and integer programming [44].

Despite of rapid development of the simplex method and interior point methods and despite their success, the Fourier-Motzkin elimination method remained significant because of some specific features. See below.

Applications. The problem of solving systems of linear constraints (equivalently linear programming) has a great many applications in both academic and industrial areas. It is used most extensively in the fields of business and economics, in a variety of commercial and non-commercial applications, and in engineering problems. It is used in a large number of combinatorial and optimisation problems, Satisfiability Modulo Theories, and hybrid systems. It is used for solving problems of planning, scheduling, assignment, optimal resource allocation, and design. Industrial applications include: transportation, telecommunication, energy.

More applications of linear programming can be found in Riley and Gass [42], and Gass [19].

2.4 Related Work

In this section we introduce related work and briefly discuss some of the existing methods.

The problem of solving systems of linear constraints has been studied over several centuries and there are only a handful of methods for solving it: the Fourier-Motzkin elimination method of the 19th century and simplex and interior point methods of the 20th century.

Nowadays, due to the computational cost of the Fourier-Motzkin elimination method, the majority of linear programming solvers implement various modifications and versions of the simplex and interior point methods. Indeed, the best performance has been retained by the simplex algorithm.

Nevertheless, the Fourier-Motzkin elimination method has a certain advantage over them in the following respects: it yields an explicit representation of a solution set of a problem, provides a symbolic solution, and allows parallelisation. The Fourier-Motzkin elimination methods is also of particular value in quantifier elimination for linear real arithmetic (see, e.g., [35, 37, 20, 15]). For this reason, the method is still very convenient, sometimes even irreplaceable, in certain situations.

The following sections briefly discusses existing methods related to our research: the Fourier-Motzkin elimination method, its most prominent modification – the Chernikov algorithm, the simplex and interior point methods, and the relaxation method.

2.4.1 The Fourier-Motzkin Elimination Method

The first method for solving systems of linear constraints was proposed by Fourier in 1826. His approach in its basis, was an analogue to the existing method of variable elimination for linear equations, Gaussian elimination.

Let us consider a system S of linear constraints with variables $x_1, \ldots, x_n$. The method either finds at least one solution, or determines that S has no solutions. The method is based on an iterative algorithm changing S by eliminating a variable at each step. We assume that the variables are eliminated according to the order $\succ$ (defined earlier in Section 2.1), so that, the highest variable x_n is eliminated first. At each step, if the highest variable in the current system of linear constraints is x_k, we denote the current system by S_k, thus $S_n = S$. When the algorithm terminates, we obtain a system containing only trivial constraints, we denote this system by S_0.

Let $k > 0$. The system S_{k-1} is obtained from S_k by (i) adding new linear constraints as follows: for every pair of linear constraints $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S_k we add to S_{k-1} a new constraint $p + q \geq 0$ and (ii) removing all linear constraints containing x_k.

One can show that the original system S is unsatisfiable if and only if S_0 contains $\bot$. If S_0 does not contain $\bot$, we can build a solution σ to S using the following observation: An assignment σ satisfies S_k if and only if σ satisfies S_{k-1} and

$$x_k\sigma \in [\max\{-p\sigma \mid (x_k + p \geq 0) \in S_k\}, \min\{q\sigma \mid (-x_k + q \geq 0) \in S_k\}] \,. \qquad (2.1)$$

As usual, we assume that the minimum of the empty set is $+\infty$ and the maximum of it is $-\infty$. Condition (2.1) essentially says that the value of x_k lies in a certain interval determined by the values of variables $x_1, \ldots, x_{k-1}$. One can prove that this interval is non-empty whenever σ satisfies S_{k-1}. Thus, we can change any solution σ of S_{k-1} into a solution of S_k by updating σ at x_k by an arbitrary value in this interval. In this way we can build a solution to $S = S_n$ as follows. We start with an arbitrary assignment σ (which obviously satisfies S_0) and update it at $x_1, \ldots, x_n$ as described above. In fact, all solutions of the initial system can be derived this way.

As we know, geometrically S defines a convex polyhedron in n-dimensional space (see Section 2.2). When eliminating the variable x_n this polyhedron is projected along the axis x_n into the $n-1$-dimensional space determined by the variables $x_{n-1}, \ldots, x_1$. The projection is represented by the system S_{n-1} of linear constraints. The Fourier-Motzkin elimination method generates a finite number of constraints at each iteration. Therefore, S_{n-1} also forms a polyhedron, this time in $n-1$-dimensional space. The geometric idea behind the Fourier-Motzkin elimination method is in succesive projections of the initial polyhedron into the smaller-dimensional spaces along each of the axes.

Performance of the Method. Note that the Fourier-Motzkin algorithm applied to a set of linear constraints always terminates and generates only a finite number of linear constraints. However, the algorithm is in general exponential.[1] In general, the number of linear constraints in S_{k-1} is in the worst case quadratic in the number of constraints in S_k.

The poor performance of the Fourier-Motzkin elimination method is mainly due to two main reasons:

1. It derives a large number of new constraints, a great many of which are redundant - i.e. are a consequence of other constraints. Checking redundancy is usually burdensome and expensive. Therefore, the method often results in exponential blow up in the number of constraints.

2. It is a refutation method and is not suited for model searching, in contrast to simplex and interior point methods. Thus, even if there is a trivial model of the problem, the Fourier-Motzkin elimination method will go on deducing constraints until all possibilities are exhausted.

[1] Some papers claim it is double-exponential but we could not find any paper proving this. Schrijver [44] defines a sequence of systems of size $O(n^3)$ on which the method generates $O(2^n)$ constraints. Some papers refer to [5] as giving an example of double-exponential behaviour but [5] only repeats the example from [44] verbatim.

The performance of the Fourier-Motzkin method may be improved by reducing the number of redundant inequalities and thus decreasing the number of inequalities generated at each iteration. Several modifications of the Fourier-Motzkin algorithm have been implemented, to which the next section is dedicated.

2.4.2 Modifications of the Fourier-Motzkin Method

During the past several decades, various modifications of the Fourier-Motzkin method have been developed. All of them mainly aim to reduce the number of derived constraints and improve the efficiency of the method this way. Namely, they aim to identify redundancy among derived constraints by providing some easy-to-check sufficient criteria for redundancy. One of the most prominent modifications of the Fourier-Motzkin elimination method is the Chernikov algorithm [6]. Chernikov suggested associating with each constraint some bookkeeping information on how this constraint was derived. Under certain conditions a newly derived constraint can be shown to be redundant based on this information. There are a number of extensions and modifications of this and other ideas developed over the past decades (e.g., [14, 27, 22, 23]).

The idea by Chernikov is to attach *an index*, a set of natural numbers, to each constraint. The initial constraints in S are indexed with a natural number corresponding to their ordinal number in the system. An index of each derived constraint is defined as a union of indices of the predecessors of this constraint (a pair of constraints it was derived from).

Chernikov's redundancy criteria for constraints derived in the Fourier-Motzkin elimination method are defined as follows:

A constraint derived on the k-th iteration is redundant if any of these criteria holds:

1. The cardinality of the index of this constraint is greater than $k+1$;

2. The index of the constraint contains a complete index of another constraint derived on the same iteration.

Informally, the Chernikov algorithm extends the Fourier-Motzkin method with these two restrictions on added linear constraints. The Chernikov algorithm modifies the Fourier-Motzkin method in the following way. Let $S = S_n$ be a system of linear constraints. Define the index set of an initial constraint $c \in S_n$

to be $\{c\}$. Let $k > 0$. The system S_{k-1} is obtained from S_k by removing all linear constraints containing x_k and adding new linear constraints as follows. For every pair of linear constraints $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S_k, with index sets I, J respectively, we add to S_{k-1} a new constraint $p + q \geq 0$ with the index set $I \cup J$ if it is non-redundant within Chernikov's definition, i.e. if neither of the above stated Chernikov criteria hold. Based on these criteria it is possible to essentially decrease the number of generated constraints when eliminating variables. It is shown in [6] that the original system S is unsatisfiable if and only if S_0 contains $\perp$.

The method of Fourier-Motzkin with filtering of redundant inequalities based on Chernikov's method or its variations has been used in several software implementations (see, e.g., one of the recent works [33]).

2.4.3 The Simplex Method

The simplex method is the most prominent method for linear programming. It was designed by Dantzig in 1947 and has, at the moment, the best performance in practice among the existing methods for linear programming. In the current section we briefly describe the simplex method; one can find a more detailed introduction to the method in [8, 44, 7].

Simplex deals with linear programming problems, but can be efficiently adopted for solving systems of linear constraints. Recall that a linear programming problem concerns finding an optimal value of a linear objective function subject to a system of linear constraints. If the system has no solutions the linear programming problem has no solution either.

Geometrically the system of linear constraints defines a convex polyhedron. The essence of the problem is to find such a point in the polyhedron that makes the objective function optimal. The convexity of the polyhedron guarantees that if the problem has a solution than the optimal value of the objective function can be achieved at a vertex of the polyhedron. Such a vertex is called *an optimal* vertex.

The idea behind simplex is to walk along the edges of the polyhedron from vertex to vertex until an optimal value of the objective function is obtained. Simplex initiates the walk by finding one of the vertices of the polyhedron, i.e., constructing a solution of the system in the vertex of the polyhedron. Then, it follows a path along the edges of the polyhedron towards the more or equally optimised vertices, until the optimal vertex is reached.

If the polyhedron is empty, i.e., the system has no solutions, simplex terminates, indicating that the system is infeasible. If the polyhedron is unbounded towards the direction of the optimisation, the optimal value is either $+\infty$ in case of maximisation of the objective function or $-\infty$ in case of minimisation.

The method is very fast in practice. Even though there are some instances which take simplex exponential time to solve, experiments suggest the number of vertices on the path towards the optimal vertex is almost linear in the dimensions of problems. Also theoretically, in certain probabilistic models the simplex method shows a linear-time (in the size of the problem) behaviour in average.

Obviously, the performance of simplex is directly related to the path chosen when searching for the optimal vertex. So far none of the existing heuristics for choosing the path results in polynomial-time performance of the simplex method for each LP instance.

The good performance of simplex in practice is also caused by the constant space used by simplex (it runs on a fixed *simplex tableau*).

2.4.4 The Relaxation Method

The methods described earlier in this chapter are *direct methods*; they provide a solution to a problem in finite number of steps. In this section we introduce an *iterative method* for solving systems of linear constraints – *the relaxation method.* With certain prerequisites on a system of constraints the method constructs a sequence of assignments to the variables that either converges to a solution or terminates with a solution.

The relaxation method was first introduced by Agmon [1], Motzkin and Schoenberg in 1954 [39]. We give a brief description of the method suited to our needs, more extensive descriptions can be found in [44], [21].

The method of relaxation is a distinguished geometric approach to the problem.

Let S be a set of linear constraints over the variables $x_0, x_1, \ldots, x_n$. Choose an arbitrary initial assignment to the variables σ_0. The method constructs a sequence of assignments $\sigma_0, \sigma_1, \sigma_2, \ldots$ iteratively as follows. If at any iteration a satisfying assignment is constructed, the method stops. Otherwise, the method goes on constructing new assignments. Suppose, an assignment σ_k for some $k > 0$ is not a solution to S. Then there exists a violated constraints in S. Geometrically the fact that a constraint is violated under the assignment σ_k means that the point M_k corresponding to σ_k lies outside the multidimensional half-spaces defined by this constraint. Choose one of the violated constraints $q \geq 0$ in S. Construct the assignment σ_{k+1} by going along the projection of the point M_k into the hyperplane defined by $q = 0$ and choosing a new point on the projection. The points should be chosen with one and the same predefined linear proportion at each iteration. The choices for a violated constraint and a proportion value are subject to any desired heuristics. However, the convergence and termination properties of the method directly depend on some prerequisites on the system and the choices for these parameters.

It has been proved, e.g. see [44], that if the system has a feasible solution such a sequence of assignments either converges to a solution or terminates with a solution provided the following:

- Always choose the "most violated" constraint, i.e. the one with the hyperplane farthest from the point corresponding to the current assignment.
- Always choose a new assignment within the interval defined by the point corresponding to the current assignment and its reflection into the hyperplane of the chosen violated constraint.

The termination is guaranteed if the polyhedron is full-dimensional and the assignments are constructed by reflecting the point corresponding to the current assignment into the hyperplane of the "most violated" constraint. In other words, by iteratively reflecting the point corresponding to the initial assignment into the hyperplanes of the "most violated" constraints, the method eventually finds a solution.

In case when the polyhedron is not full-dimensional the relaxation method requires some modifications to be made to terminate and yield to the exact solution. However, these modifications are very complex and computationally expensive. For this reason, the relaxation method is not utilised much in practise.

In Chapter 4 we describe how we used the criteria posed in the relaxation method for fine-tuning the conflict resolution method.

Chapter 3

Conflict Resolution

In this chapter we introduce our *conflict resolution algorithm (CRA)* for solving systems of linear constraints over the rationals and reals. We present properties of the CRA algorithm, prove its correctness and termination, and discuss some CRA extensions.

3.1 Algorithm Description

As previously, we consider a system S of linear constraints and suppose that we have the same order on variables $x_n \succ x_{n-1} \succ \ldots \succ x_1$ (as introduced in Chapter 2).

First, we introduce some auxiliary notions useful for describing our algorithm. Let c be a linear constraint. If the highest variable in c is x_k, then we say that k is the *level* of c. Thus, each constraint is at a certain level. If c contains no variables, then we define the level of c to be 0. In the description of the conflict resolution algorithm we assume that all constraints are normalised. By the definition of a normalised constraint, see Section 2.1, we assume that constraints written in the form $x_k + p \geq 0$ or $-x_k + q \geq 0$ have the highest variable x_k, and p and q do not contain x_k, thus these constraints are at level k. The notion of level induces a partial order on linear constraints, which we will denote also by $\succ$, as follows. For two linear constraints c_1 and c_2, we have $c_1 \succ c_2$ if and only if the level of c_1 is strictly greater than the level of c_2.

For every set S of linear constraints and a positive integer k, we denote by $S_{=k}$ (respectively, $S_{<k}$) the subset of S consisting of all constraints at level k (respectively, of all levels strictly less than k).

We consider constraints of level k split into two groups according to the sign of the coefficient of the highest variable x_k. This allows us to identify lower and upper bounds on the value of the variable x_k if an assignment to the smaller variables is given.

For any system S of linear constraints, a non-negative integer k and an assignment σ we denote

$$\begin{aligned} L(S,\sigma,k) &\overset{\text{def}}{=} \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}; \\ U(S,\sigma,k) &\overset{\text{def}}{=} \min\{q\sigma \mid (-x_k + q \geq 0) \in S\}; \\ I(S,\sigma,k) &\overset{\text{def}}{=} [L(S,\sigma,k), U(S,\sigma,k)]. \end{aligned}$$

Here, $L(S,\sigma,k)$ and $U(S,\sigma,k)$ represent, respectively, a lower and an upper bound on the value of the variable x_k, as the values of the smaller variables are chosen from the assignment σ. These bounds essentially form an interval $I(S,\sigma,k)$ bounding the value of the variable x_k. If the interval is empty, it means that there is at least one pair of constraints that are conflict under the current assignment σ, i.e. there is a k-conflict.

We say a level k is *half-bounding* if all constraints at level k have the same sign of the coefficient of the highest variable x_k; i.e. level k contains either only constraints of the form $-x_k + p \geq 0$ and $-x_k + p > 0$ bounding the maximal value of the variable x_k, or only constraints of the form $x_k + q \geq 0$ and $x_k + q > 0$ bounding the minimal value of x_k. Obviously, such levels define *half-bounded intervals* for x_k.

We call a *state* of a system a pair (S,σ), where S is a system of linear constraints and σ an assignment (as defined in the Chapter 2).

Let $\mathbb{S} = (S,\sigma)$ be a state and k a positive integer. We define a *k-conflict* in the state $\mathbb{S}$ as a pair of constraints $(x_k + p \geq 0, -x_k + q \geq 0)$ that satisfies the conditions (i) both $x_k + p \geq 0$ and $-x_k + q \geq 0$ are linear constraints in S and (ii) $p\sigma + q\sigma < 0$. Instead of "k-conflict" we will sometimes simply say "conflict". Note that if σ is a solution of S, then $\mathbb{S}$ contains no conflicts.

We will now formulate our method. Given a system S of linear constraints, it starts with an initial state (S,σ), where σ is an arbitrary assignment and repeatedly transforms the current state either by updating S – adding a new linear constraint to S, or updating the assignment σ. These transformations are applied in accordance to the two rules which we formulate below as *transformation rules* on states $\mathbb{S} \Rightarrow \mathbb{S}'$, meaning that $\mathbb{S}$ can be transformed into $\mathbb{S}'$.

Let k be an integer such that $1 \leq k \leq n$. The first transformation rule is called *the conflict resolution rule (CR)*, when applied it updates the current system of constraints S by adding a new constraint to it. This rule is applicable at level k with a condition that a k-conflict is present there. In terms of the interval $I(S, \sigma, k)$, this means that $I(S, \sigma, k)$ is empty. The CR rule resolves the k-conflict by deriving a new constraint from the conflicting pair of constraints and adding it to the system.

The conflict resolution rule (CR) (at level k) is the following rule:

$$(S, \sigma) \Rightarrow (S \cup \{p + q \geq 0\}, \sigma),$$

where (S, σ) contains a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$.

The second rule is *the assignment refinement rule (AR)*, as its name suggests it refines the assignment on variables. The AR rule is applicable at level k when (i) there are no k-conflicts; (ii) the current assignment satisfies all constraints at levels below k, but violates at least one constraint at level k, i.e. $\sigma \models S_{<k}$ and $\sigma \not\models S_{=k}$. This, in terms of the interval $I(S, \sigma, k)$, means that the $I(S, \sigma, k)$ is non-empty, but the value v of the variable x_k does not belong to it $v \notin I(S, \sigma, k)$. In such case the assignment can be refined. The AR rule updates σ at the variable x_k by a new value v' so that now $\sigma_{x_k}^{v'} \models S_{=k}$.

The assignment refinement rule (AR) (at level k) is the following rule:

$$(S, \sigma) \Rightarrow (S, \sigma_{x_k}^{v}),$$

where

(1) σ satisfies all constraints in S at levels $0, \ldots, k - 1$.

(2) σ violates at least one constraint in S at level k.

(3) $\sigma_{x_k}^{v}$ satisfies all constraints in S at level k.

We will call any application of an inference rule an *inference*. Thus, our algorithm will perform CR-inferences and AR-inferences.

Note, that the AR and CR rules exclude each-other: if one of them is applicable, the other is not. This is formulated in the following lemma that is a key lemma for our method.

Lemma 3.1.1 *(Proved in Section 3.3) Let (S, σ) be a state and $1 \leq k \leq n$. Let σ satisfy all constraints in S at levels $0, \ldots, k-1$ and violate at least one constraint at level k. If $I(S, \sigma, k)$ is empty, then the conflict resolution rule at level k is applicable and the assignment refinement rule at this level is not applicable. If $I(S, \sigma, k)$ is non-empty, then the assignment refinement rule at the level k is applicable and the conflict resolution rule at this level is not applicable.*

Note that the conflict resolution rule always derives a linear constraint that is violated by current assignment σ:

Lemma 3.1.2 *Let (S, σ) contain a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$. Then $\sigma \not\models p + q \geq 0$.* ❑

In the next lemma we give formal and more detailed explanation of the assignment refinement rule. The lemma illustrates the conditions (1) – (3) of the AR rule in terms of the interval $I(S, \sigma, k)$.

Lemma 3.1.3 *(i) Condition (2) of the assignment refinement rule implies $x_k\sigma \notin I(S, \sigma, k)$. (ii) Condition (3) of the assignment refinement rule is equivalent to $v \in I(S, \sigma, k)$. (iii) The interval $I(S, \sigma, k)$ is non-empty if and only if $\mathbb{S}$ contains no k-conflicts.*

Proof. (i) We assume that $x_k\sigma \in I(S, \sigma, k)$ and prove that σ satisfies $S_{=k}$. Take any constraint in $S_{=k}$. Without loss of generality assume that it has the form $x_k + p \geq 0$. Since $x_k\sigma \in I(S, \sigma, k)$, we have $x_k\sigma \geq L(S, \sigma, k)$, that is, $x_k\sigma \geq \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}$. This implies $x_k\sigma \geq -p\sigma$, hence σ is a solution of $x_k + p \geq 0$.

(ii) In one direction, assume $v \in I(S, \sigma, k)$. Note that $x_k\sigma^v_{x_k} = v$, so $x_k\sigma^v_{x_k} \in I(S, \sigma, k)$. Using the same arguments as in (i) but with σ replaced by $\sigma^v_{x_k}$ we can prove $\sigma^v_{x_k} \models S_{=k}$. In the other direction, assume $\sigma^v_{x_k} \models S_{=k}$. We have to prove $v \in I(S, \sigma, k)$, that is, $v \geq L(S, \sigma, k)$ and $v \leq U(S, \sigma, k)$. We will only prove the former condition, the latter one is similar. The former condition means $v \geq \max\{-p\sigma \mid (x_k + p \geq 0) \in S\}$. To prove it, we have to show that for all

Algorithm 1 The Conflict Resolution Algorithm CRA

Input: A set S of linear constraints.

Output: A solution of S or "unsatisfiable".

```
 1: if $\perp \in S$ then return "unsatisfiable"
 2: $\sigma :=$ arbitrary assignment;
 3: $k := 1$
 4: while $k \leq n$ do
 5:     if $\sigma \not\models S_{=k}$ then
 6:         while $(S, \sigma)$ contains a $k$-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$ do
 7:             $S := S \cup \{p + q \geq 0\}$;                    ▷ application of CR
 8:             $k :=$ the level of $(p + q \geq 0)$;
 9:             if $k = 0$ then return "unsatisfiable"
10:         end while
11:         $\sigma := \sigma^v_{x_k}$, where $v$ is an arbitrary value in $I(S, \sigma, k)$    ▷ application of AR
12:     end if
13:     $k := k + 1$
14: end while
15: return $\sigma$
```

constraints of the form $x_k + p \geq 0$ in S (and hence in $S_{=k}$) we have $v \geq -p\sigma$. Since p may only contain variables in $\{x_1, \dots, x_{k-1}\}$ and σ agrees with $\sigma^v_{x_k}$ on all such variables, we have $-p\sigma = -p\sigma^v_{x_k}$, so $v \geq -p\sigma^v_{x_k}$. Using $x_k\sigma^v_{x_k} = v$, we obtain $x_k\sigma^v_{x_k} \geq -p\sigma^v_{x_k}$, hence $\sigma^v_{x_k}$ is a solution of $x_k + p \geq 0$, and we are done.

(iii) We will prove that $I(S, \sigma, k)$ is empty if and only if $\mathbb{S}$ contains a k-conflict. In one direction, assume $I(S, \sigma, k)$ is empty. Then $L(S, \sigma, k) > U(S, \sigma, k)$. Note that this implies that both $L(S, \sigma, k)$ and $U(S, \sigma, k)$ are finite. Since they are finite, $S_{=k}$ contains two constraints of the form $x_k + p \geq 0$ and $-x_k + q \geq 0$ such that $-p\sigma = L(S, \sigma, k)$ and $q\sigma = U(S, \sigma, k)$. This and $L(S, \sigma, k) > U(S, \sigma, k)$ implies $-p\sigma > q\sigma$, and so $0 > p\sigma + q\sigma$. Therefore, $(x_k + p \geq 0, -x_k + q \geq 0)$ is a k-conflict. The proof in other direction is similar. ❑

The *conflict resolution algorithm CRA* is given as Algorithm 1.

Let us note that the algorithm is well-defined, that is, the interval $I(S, \sigma, k)$ at line 11 is non-empty. Indeed, the algorithm reaches this line if (S, σ) contains no conflict at the level k (by line 6). Then $I(S, \sigma, k)$ is non-empty by Lemma 3.1.3 (iii).

Example 3.1. This example illustrates the algorithm. Let S_0 be the following set of constraints.

$$\begin{array}{rcrcrcrcrcl}
x_4 & - & 2x_3 & & & + & x_1 & + & 5 & \geq & 0 \qquad (1)\\
-x_4 & - & x_3 & - & 3x_2 & - & 3x_1 & + & 1 & \geq & 0 \qquad (2)\\
-x_4 & + & 2x_3 & + & 2x_2 & + & x_1 & + & 6 & \geq & 0 \qquad (3)\\
 & & -x_3 & + & x_2 & - & 2x_1 & + & 5 & \geq & 0 \qquad (4)\\
 & & x_3 & & & + & 3x_1 & - & 1 & \geq & 0 \qquad (5)
\end{array}$$

Assume that the initial assignment σ maps all variables to 0. The algorithm starts at level 0. The sets $S_{=0}, S_{=1}, S_{=2}$ are empty, so the assignment σ trivially satisfies them. However, it violates constraint (5) and so violates $S_{=3}$. The interval $I(S, \sigma, 3)$ is $[1, 5]$. It is non-empty, so by Lemma 3.1.1 we can apply the assignment refinement rule at level 3 by updating σ at x_3 by any value in $[1, 5]$. Let us choose, for example, the value 4. Let σ_1 denote the newly obtained assignment $\{x_4 \mapsto 0, x_3 \mapsto 4, x_2 \mapsto 0, x_1 \mapsto 0\}$. Now we move to the next level 4. There is a 4-conflict between constraints (1) and (2) (line 6). We make a CR-inference between these two clauses deriving a new constraint

$$- \; x_3 \; - \; x_2 \; - \; \tfrac{2}{3}x_1 \; + \; 2 \; \geq \; 0 \qquad (6)$$

added to the set S at line 7. According to line 8 of the algorithm we set the level k to the level of the new constraint, that is, to 3. Now there are no more conflicts on level 3 and we have $I(S, \sigma, 3) = [1, 2]$. We should update the assignment at x_3 by an arbitrary value in this interval. Suppose, for example, that we have chosen 1 as the value for x_3 obtaining $\{x_4 \mapsto 0, x_3 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 0\}$ and increase k by 1 proceeding to level 4. At this moment all constraints at level 4 are satisfied and the algorithm terminates returning σ. ❏

We complete the discussion of the general conflict resolution algorithm by bringing up the issue of fine-tuning the algorithm. In Example 3.1 when we detected a conflict at level 4 there was only one conflicting pair of constraints, which we resolved applying the CR rule. In general we may encounter more than one conflict at a level. In such case, we decide which conflict to resolve. Similarly, at some level k when refining the assignment $\sigma^v_{x_k}$ we have the flexibility to select the value v from the interval $I(S, \sigma, k)$. The choice of both, a conflict in the CR rule and a value for refining the assignment in the AR rule, defines the course of the run of the CRA algorithm and affects its performance. Certainly, yet another parameter central for the efficiency of the algorithm is the order on variables. Indeed, entire course of the algorithm can change when the order on variables is shuffled. Consequently, the CRA algorithm can be parametrised by various strategies for (i) selection of conflicting pairs: we can choose any conflicting pair (at line: 6), (ii) refinement of assignments: we can choose any value v inside the interval $I(S, \sigma, k)$ (at line: 11) and (iii) selection of the order on variables $\succ$. We consider these strategies in the Implementation Chapter of the thesis, in Section 4.2.2, where we discuss the details of various implemented heuristics that we used for fine-tuning the algorithm.

In the next section we prove correctness and termination of our algorithm.

3.2 Correctness and Termination

In the following we give a proof of correctness and termination of the *conflict resolution algorithm*.
The following theorem proves that the algorithm is correct and terminating.

Theorem 3.2.1 *The conflict resolution algorithm CRA always terminates. Given an input set of constraints S_0, if CRA outputs "unsatisfiable", then S_0 is unsatisfiable. If CRA outputs an assignment σ, then σ is a solution of S_0.*

The proof requires establishing a series of lemmas that we will state below. Some of these lemmas establish properties of the CRA algorithm and will be proved in the next chapter along with the other properties. In these lemmas we always denote the input set of constraints by S_0.

The following is a key lemma for our method, it was introduced earlier when describing the algorithm and we will formulate it here for readers' convenience.

Lemma 3.1.1 *(Proved in Section 3.3) Let (S, σ) be a state and $1 \leq k \leq n$. Let σ satisfy all constraints in S at levels $0, \ldots, k-1$ and violate at least one constraint at level k. If $I(S, \sigma, k)$ is empty, then the conflict resolution rule at level k is applicable and the assignment refinement rule at this level is not applicable. If $I(S, \sigma, k)$ is non-empty, then the assignment refinement rule at the level k is applicable and the conflict resolution rule at this level is not applicable.*

Lemma 3.2.2 *(Proved in Section 3.3) At any step of the algorithm the set S is equivalent to S_0, that is, S and S_0 have the same set of solutions.*

The following lemma is obvious.

Lemma 3.2.3 *Every constraint occurring in S at any step of the CRA algorithm belongs to the set of constraints derived by the Fourier-Motzkin algorithm applied to S_0.* ❑

Lemma 3.2.4 *(Proved in Section 3.3) The assignment σ at lines 4 and 6 satisfies $S_{<k}$.*

Lemma 3.2.5 *(Proved in Section 3.3) Let (S, σ) contain a conflict $(x_k + p \geq 0, -x_k + q \geq 0)$ at line 6. Then we have $(p + q \geq 0) \notin S$.*

This lemma means that the same constraint will never be added again to S. In fact, the algorithm has a much stronger property formulated below in Lemma 3.3.1.

Let us now give the proof of Theorem 3.2.1.

Proof. We start with proving termination. By Lemma 3.2.5 the algorithm never adds the same constraint twice. By Lemma 3.2.3 we can add only a finite number of different constraints. Therefore, the condition on line 6 can hold only a finite number of times. From the moment this condition becomes permanently false, k will always increase by 1, so the outermost while-loop will terminate.

Suppose now that the algorithm returns "unsatisfiable". If this happens at line 1, then $\perp \in S_0$, so S_0 is unsatisfiable. Otherwise, this happens at line 9. Then $\sigma \not\models p+q \geq 0$ by Lemma 3.1.2. Since $k = 0$, then the constraint $p+q \geq 0$ contains no variables, so this constraint is trivial and unsatisfiable. By Lemma 3.2.2, this constraint is implied by S_0, hence S_0 is unsatisfiable too.

It remains to consider the case when the algorithm returns an assignment σ. This only can happen at the last line of the algorithm. At this line, $k = n + 1$. By Lemma 3.2.4, σ satisfies $S_{<n+1}$. Note that $S_{<n+1} = S$, so σ also satisfies S. By Lemma 3.2.2, S is equivalent to S_0, hence σ also satisfies S_0. ❑

3.3 Properties of the CRA Algorithm

In this chapter we prove properties of the CRA algorithm. Some of the properties have been introduced in the previous chapter as the auxiliary lemmas needed for the proof of correctness and termination of the algorithm.

In the following lemmas we always denote the input set of constraints by S_0. We start by proving the key lemma for our algorithm.

Lemma 3.1.1 *Let (S, σ) be a state and $1 \leq k \leq n$. Let σ satisfy all constraints in S at levels $0, \ldots, k-1$ and violate at least one constraint at level k. If $I(S, \sigma, k)$ is empty, then the conflict resolution rule at the level k is applicable and the assignment refinement rule at this level is not applicable. If $I(S, \sigma, k)$ is non-empty, then the assignment refinement rule at the level k is applicable and the conflict resolution rule at this level is not applicable.*

Proof. Suppose $I(S, \sigma, k)$ is empty. By Lemma 3.1.3 (iii) $\mathbb{S}$ contains a k-conflict, so the conflict resolution rule is applicable at the level k. Since $I(S, \sigma, k)$ is empty, by Lemma 3.1.3 (ii) condition (3) of the assignment refinement rule is violated, so the assignment refinement rule at this level is not applicable.

Suppose that $I(S, \sigma, k)$ is non-empty. Then by Lemma 3.1.3 (iii) $\mathbb{S}$ contains no conflict, so the conflict resolution rule at the level k is not applicable. Take an arbitrary value $v \in I(S, \sigma, k)$. By Lemma 3.1.3 (ii) condition (3) of the assignment refinement holds. Conditions (1) and (2) of this rule hold by the assumptions of this lemma, so the assignment refinement rule is applicable. ❑

Lemma 3.2.2 *At any step of the algorithm the set S is equivalent to S_0, that is, S and S_0 have the same set of solutions.*

Proof. Observe that line 7 in the Algorithm 1 is the only line that changes S. It is easy to see that the application of this line does not change the set of solutions of S since the constraint $p + q \geq 0$ added to S is implied by S. ❑

The following lemma is obvious.

Lemma 3.2.3 *Every constraint occurring in S at any step of the CRA algorithm belongs to the set of constraints derived by the Fourier-Motzkin algorithm applied to S_0.* ❑

Lemma 3.2.4 *The assignment σ at lines 4 and 6 satisfies $S_{<k}$.*

Proof. By induction on the number of iterations of the outermost while-loop. Before the first iteration the property is obvious since $k = 1$ and $\perp \notin S$. So we assume that the property holds before an iteration of the loop and show it holds after this iteration. If $\sigma \models S_{=k}$ at line 5, then by $\sigma \models S_{<k}$ we have $\sigma \models S_{<k+1}$. It

remains to consider the case when $\sigma \not\models S_{=k}$ at line 5. In this case the algorithm may enter the internal while-loop starting at line 6. It is easy to see that at the exit of this loop the property is satisfied as well, since k only decreases in the loop and the new constraint $p + q \geq 0$ is at the level k. So it remains to show that after line 11 we have $\sigma \models S_{=k}$. But this is guaranteed by Lemma 3.1.3 (ii), so we are done. ❏

Lemma 3.2.5 *Let* (S, σ) *contain a conflict* $(x_k + p \geq 0, -x_k + q \geq 0)$ *at line 6. Then we have* $(p + q \geq 0) \notin S$*.*

Proof. By Lemma 3.2.4 at line 6 in the Algorithm 1 we have $\sigma \models S_{<k}$. But we have $\sigma \not\models (p + q \geq 0)$, hence $(p + q \geq 0) \notin S_{<k}$. Since the level of $(p + q \geq 0)$ is strictly less than k this implies $(p + q \geq 0) \notin S$. ❏

This lemma means that the same constraint will never be added again to S. In fact, the algorithm has a much stronger property formulated below in Lemma 3.3.1.

We say, that a CR-inference at a level k is *redundant* w.r.t. a state (S, σ) if the conclusion of this inference is a consequence of constraints in $S_{<k}$. Let us prove a key property that distinguishes our algorithm from the Fourier-Motzkin method.

Lemma 3.3.1 *Every CR-inference performed by the CRA algorithm is non-redundant.*

Proof. Suppose that the algorithm performs a redundant inference adding $p+q \geq 0$ at line 7. Then by the definition of redundancy $p + q \geq 0$ is implied by $S_{<k}$. By Lemma 3.2.4 we have $\sigma \models S_{<k}$, then σ must also satisfy $p + q \geq 0$. This contradicts the definition of a conflict. ❏

To illustrate this lemma, let us come back to Example 3.1. Note that in this example we have not applied the conflict resolution inference between constraints (1) and (3). It is easy to see that the conclusion of this inference is implied by constraints (4) and (5) at smaller levels, therefore *this inference would not be applied independently of the choices of assignments made by the algorithm.*

The interested reader can find yet another example of not applying redundant inferences in Appendix C.

Let us, now, extend our notion of redundancy to constraints. We call a constraint c at a level k *redundant* if this constraint is implied by $S_{<k+1} - \{c\}$.

It is not hard to prove that constraints $x_k + p \geq 0$ such that $-p\sigma = L(S, \sigma, k)$ are "almost" non-redundant in the following sense.

Lemma 3.3.2 Consider the set S^+ of all constraints at a level k having the form $x_k + p \geq 0$. Consider its subset S' consisting of all constraints $x_k + p \geq 0$ such that $-p\sigma = L(S, \sigma, k)$. Then S' is not implied by $S_{<k} \cup (S^+ - S')$. ❏

One can formulate a symmetric property for constraints $-x_k + q \geq 0$ such that $q\sigma = U(S, \sigma, k)$.

Although our algorithm does not perform redundant inferences, the system may contain redundant constraints at a level k for two reasons: (i) it may contain redundant constraints initially; and (ii) addition of new constraints to a level k may make other constraints at this and higher levels redundant. Choosing at line 6 in the Algorithm 1 a k-conflict $x_k+p \geq 0$ and $-x_k+q \geq 0$ in S (i.e. $p\sigma+q\sigma < 0$), such that $-p\sigma = L(S, \sigma, k)$ and $q\sigma = U(S, \sigma, k)$ does not, in general, guarantee, that the constraints forming the conflict are non-redundant but it guarantees that they are "almost" non-redundant in the sense of Lemma 3.3.2.

3.4 Extensions of the CRA Algorithm

In this section we briefly mention two extensions of the method: one is for working with strict inequalities and the other for linear programming.

3.4.1 Extensions With $\{>, =\}$

Extension with $\{>\}$.

The modification of the algorithm for working with *strict inequalities* $p > 0$ is straightforward. First, when we consider the interval

$$I(S, \sigma, k) = [L(S, \sigma, k), U(S, \sigma, k)]$$

if any endpoint of this interval corresponds to a strict inequality, we use a semi-open or an open interval instead. For example, if there is a strict inequality

$(x_k + p > 0) \in S$ such that $-p\sigma = L(S, \sigma, k)$ but no strict inequality $(-x_k + q > 0) \in S$ such that $q\sigma = U(S, \sigma, k)$, then we use the semi-open interval

$$I(S, \sigma, k) = (L(S, \sigma, k), U(S, \sigma, k)].$$

Second, the result of the conflict resolution rule is a strict inequality if at least one of the premises is strict.

For the CR rule, we also have a slight modification. Namely, if at least one of the constraints is a strict inequality then the resolvent will be a strict inequality too.

Extension with $\{=\}$.

Now we describe the extension of the algorithm for equalities. As mentioned earlier, when introducing the notion of a level, we split inequalities into two groups according to the sign of the coefficient of the highest variable. In case, when equalities are added to the system we add a third group.

Equalities at a level are treated as an explicit assignment to the corresponding highest variable. In the following we show how we deal with equalities.

Let (S, σ) be a state, and $k > 0$. Suppose, at a level k we have equalities. Suppose, all the constraints in $S_{<k}$ are satisfied by the current assignment σ.

Now, we try to adjust the assignment at the level k. First, we need to make sure there are no k-conflicts. In case of equalities, conflict may occur between: (i) a pair of equalities, (ii) inequality and equality, and (iii) a pair of inequalities.

Generally, presence of equalities at a level is very useful, as they serve as explicit assignments to the corresponding highest variables and we can always update their value with better precision.

However, if we have several equalities at a level they may produce conflicts between each-other and with other constraints at that level as well. We extend the notion of the k-conflict for equalities in a natural way as follows:

A pair of constraints $(x_k + p = 0, (-)x_k + q \diamond 0)$ is a k-conflict in a state (S, σ) if it satisfies the conditions: (i) both constraints $(x_k + p = 0$ and $(-)x_k + q \diamond 0)$ are linear constraints in S and (ii) $(-)p\sigma + q\sigma \diamond 0$. Here $\diamond \in \{\geq, >, =, \neq\}$.

In case of a conflict $(x_k + p = 0, (-)x_k + q = 0)$, we resolve the conflicting by applying an extension of the CR rule for equalities:

The conflict resolution rule (CR) for $\{=\}$ (at level k):

$$(S, \sigma) \Rightarrow (S \cup \{(-)p + q = 0\}/\{(-)x_k + q = 0\}, \sigma),$$

where (S, σ) contains a k-conflict $(x_k + p = 0, (-)x_k + q = 0)$, and $\diamond \in \{\geq, >, =\}$.

It is easy to show, that the modified system is equivalent to S. Indeed, it is enough to show that the sets $\{x_k + p = 0, (-)x_k \diamond q = 0\}$ and $\{x_k + p = 0, (-)p + q \diamond 0\}$ are equivalent.

If equalities at a level k do not induce any conflicts, we check for conflicts between inequalities. For this, we calculate the interval $I(S, \sigma, k)$. If it is empty, there is a conflict at level k and we resolve this conflict the same way as is done in the original CRA algorithm. If the interval is non-empty, we calculate the value $v = p\sigma$ for the variable x_k.

Since equalities do not induce k-conflict here, we are guaranteed that $v \in I(S, \sigma, k)$ and we can update the assignment σ_x^v.

3.4.2 Conflict Resolution and Linear Programming

To use our algorithm for linear programming, we can use the following trick. Suppose, for example, that we want to find a maximum of a linear function p. To this end we assume that the constraints do not contain the variable x_1 and add the equality $p - x_1 = 0$. After that we use our algorithm with the only modification that we always select the maximal possible value for x_1 in the assignment refinement rule. Special care should be taken when we have no *a priory* upper bound on x_1.

3.4.3 Conflict Resolution and the Fourier-Motzkin Method

In this section we compare the conflict resolution algorithm with the Fourier-Motzkin variable elimination method. This comparison is of particular interest since the use of the conflict resolution rule in the CRA algorithm makes our method similar to the Fourier-Motzkin variable elimination method.

As mentioned in Section 2.4.1 the main disadvantages of the Fourier-Motzkin elimination method that makes it perform poorly in practice are the following: (i) the Fourier-Motzkin method generates a large amount of redundant constraints, which are hard to check for redundancy, consequently a problem becomes large (in general exponential) and hard to deal with very quickly, (ii) even if there is a trivial solution of a problem, the method of Fourier-Motzkin still needs to perform all possible inferences in order to derive the solution.

Our method alleviates both of these problems. It does not derive redundant constraints (Lemma 3.3.1) and stops immediately if a solution is found (guaranteed by the while-loop in the Algorithm 1).

We illustrated the comparison of the Fourier-Motzkin method and the conflict resolution method on a randomly generated problem:

$$\begin{array}{rcrcrcrcrcl} x_4 & - & 2x_3 & & & + & x_1 & + & 5 & \geq & 0 \qquad (1) \\ x_4 & + & 2x_3 & + & x_2 & & & + & 3 & \geq & 0 \qquad (2) \\ -x_4 & - & x_3 & - & 3x_2 & - & 3x_1 & + & 1 & \geq & 0 \qquad (3) \\ -x_4 & + & 2x_3 & + & 2x_2 & + & x_1 & + & 6 & \geq & 0 \qquad (4) \\ & & x_3 & & & + & 3x_1 & - & 1 & \geq & 0 \qquad (5) \\ & & -x_3 & + & x_2 & - & 2x_1 & + & 5 & \geq & 0 \qquad (6) \end{array}$$

This problem is satisfiable and consists of 4 variables and 6 constraints. In Appendix A and Appendix B we depict the running process of the Fourier-Motzkin method and the conflict resolution method step-by-step. We consider this example very interesting and easy to follow. Here we present only the results of the comparison and refer interested readers to Appendixes A, B for more details.

As one can see, the comparison gives rather striking result. The Fourier-Motzkin method derives 22 constraints before it constructs a solution. While the conflict resolution method needs to derive one constraint only to construct the same solution.

Let us now show that the Fourier-Motzkin algorithm cannot polynomially simulate our algorithm in a very strong sense. This example is taken from [44]. It contains all inequalities of the form $\pm x_k \pm x_l \pm x_m \geq 0$, where $n \geq k > l > m \geq 1$. Evidently, the size of the system is $O(n^3)$ and there exists only a single solution assigning 0 to all variables. It is shown in [44] that the Fourier-Motzkin method generates exponentially many (in n) inequalities for this example. Let σ be an arbitrary assignment. Our method will start generating conflicts from level 3 containing 8 inequalities until it updates σ so that $x_1\sigma = x_2\sigma = x_3\sigma = 0$. After that it will proceed to level 4, where the interval $I(S, \sigma, 4)$ will consist of a single point 0. The assignment refinement will set $x_4\sigma$ to 0 and no conflicts will be generated. The same will happen with all levels greater than 4, so the algorithm will terminate in a linear number of steps. Essentially, apart from the initial work on level 3, the conflict resolution algorithm will only evaluate every inequality once and so work in time linear in the size of the system, that is $O(n^3)$. Note that this running time does not depend on either the choice of the initial assignment or the choice of values in the assignment refinement inferences.

3.4.4 Conflict Resolution and Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) [3] is a new and rapidly developing branch of automated reasoning dedicated to reasoning in first-order logic with various theories. The SMT problem concerns deciding satisfiability of a quantifier-free first-order formula with predicates over certain background theories and their combinations.

Examples of typical theories involved in SMT problems are linear arithmetic with integer and real variables, theory of arrays, equality and uninterpreted functions, theory of bit vectors and others.

Most of the current SMT solvers follow a so-called *DPLL(T)* approach [18]. This approach provides a framework for combining a general *propositional solver (SAT solver)* with so-called *theory solvers.*

The SAT solver deals with the propositional structure of formula and searches for a propositional model for it. Once a model is found the theory solvers check if it is consistent with the theories involved. Checking consistency implies checking satisfiability of sets of theory predicates.

In order to integrate well in DPLL(T) framework, the theory solvers are also required to interact in the process of constructing propositional models. For such interaction to work in the DPLL(T) system, the theory solvers must have a number of necessary properties: incrementality and the ability to generate explanations for unsatisfiability. These properties, along with the explanation of their purpose, are described in detail in [18].

A considerable number of SMT problems arising from real-life applications involve theories of linear real and integer arithmetic. For solving such instances an incremental version of a linear arithmetic solver is plugged in, as a theory solver, to the DPLL(T) system. This linear-arithmetic solver is responsible for deciding satisfiability of the sets of linear arithmetic constraints, it is incremental and can generate explanations to the theory inconsistencies.

As we mentioned in the introductory chapter, our research was motivated by the problems of Satisfiability Modulo theories with theories of linear real and integer arithmetic. The work on searching for new methods for integrating a linear arithmetic solver into SMT solving diverted us to searching for new ways of operating with systems of linear constraints. As a result of this research we developed the conflict resolution algorithm, which solves systems of linear constraints, and along with this, has the features important for SMT integration.

Indeed, the CRA algorithm (and our implementation) can easily be made incremental: after adding/removing constraints we can always continue with the current assignment, moreover the CRA never performs redundant inferences and in particular, never performs the same inference twice (unless the conclusion was removed). Explanations can be generated from the proofs of unsatisfiability which are easily extractable from runs of the CRA algorithm.

As future work we are keen to integrate our algorithm with SMT solving.

Chapter 4

Implementation

In order to evaluate the efficiency of the conflict resolution method we implemented a solver based on this method. This chapter describes the system design and details of the major components of the implementation.

The implementation of the solver was done in the C++ language, within the framework designed for, and used in, the automated reasoning system Vampire, developed in the department of Computer science at the University of Manchester by Prof. Andrei Voronkov, that regularly wins annual World Cups in automated theorem proving, since 1999.

In our implementation we used the GMP library for arbitrary precision arithmetic [1]. Thus, all computations with rational numbers are done with arbitrary precision.

Input to the solver is in a standard SMT format used in the SMT-LIB benchmarks [2].

In the following we first describe the basic data structure utilised in our solver and then discuss details of our implementation.

4.1 Data Structure

The first component evoked by the solver is a *parser*, it makes the necessary transformation of the input data and passes it through, to a *Sort Input Constraints* block, to be sorted and prepared for the run of the CRA algorithm. The schema

[1] `http://gmplib.org/`

[2] C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.

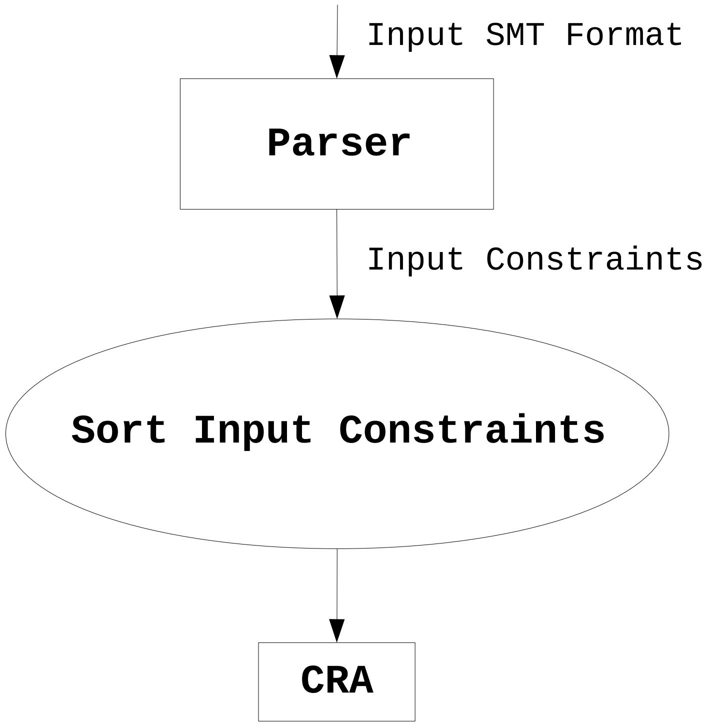


Figure 4.1: Implementation Schema

of this process is shown in Figure 4.1.

4.1.1 Parser

An input to the parser is a system of linear rational constraints written in the SMT format, that is a conjunction of the constraints in the system. The parsing process consists of generating the constraints of the system and creating a data structure essentially used in further processing of constraints.

- *Constraint*

  All constraints are located in one chunk of memory of sufficient size. Each constraint is represented by a vector of coefficients. The structure of constraints is defined by a *pointer* to a corresponding vector of coefficients, a *type* of the constraint ($\geq, >, =$) and a *counter* of the variables indicating the size of the vector. Depending on the nature of the problem, it is possible to chose between two different representations of constraints. One is a vector of fixed size equal to the number of variables in the system, and the other is a vector containing non-zero coefficients only. In the first case it is enough to mere store coefficient values in the vector, so that each coefficient is stored under the index of its variable. In this case the structure of constraints does not include a counter of variables, since all constraints have the same number of variables. In the second case a coefficient is stored as a pair of an *index* of the corresponding variable and a numeric *value* of the coefficient. In the following we will call the data structure description of constraints simply *constraints*

```
struct COEFF {
    size_t IND;
    mpq_class VALUE;
}
```

Not surprisingly, on randomly generated problems the first implementation is slightly better in both time and space while the second one can be much faster (and consume much less space) on non-random problems, where vectors are normally sparse.

The variable counter is used when processing the initial constraints only and will be discussed later in this chapter.

- *Input Constraints*

  As we said, each constraint is allocated to a chunk of memory somewhere in the whole memory. Input constraints are defined at parse time of the solver. To access them we use a simple data structure, called *Input Constraints*. This data structure represents a stack of pointers to the input constraints, see Figure 4.2. The use of a stack here was motivated by the observation: input constraints are acquired at parse time, without knowing their number in advance, consequently gathering them in one data structure is easier without controlling the number of constraints, which we realised by simply pushing to a stack.

- *Trace Variables*

  While processing the input constraints we often need to quickly find all the constraints containing some variable. To handle this problem a data structure called *trace variables* is introduced. It links each variable to a set of input constraints containing this variable (Figure 4.2.).

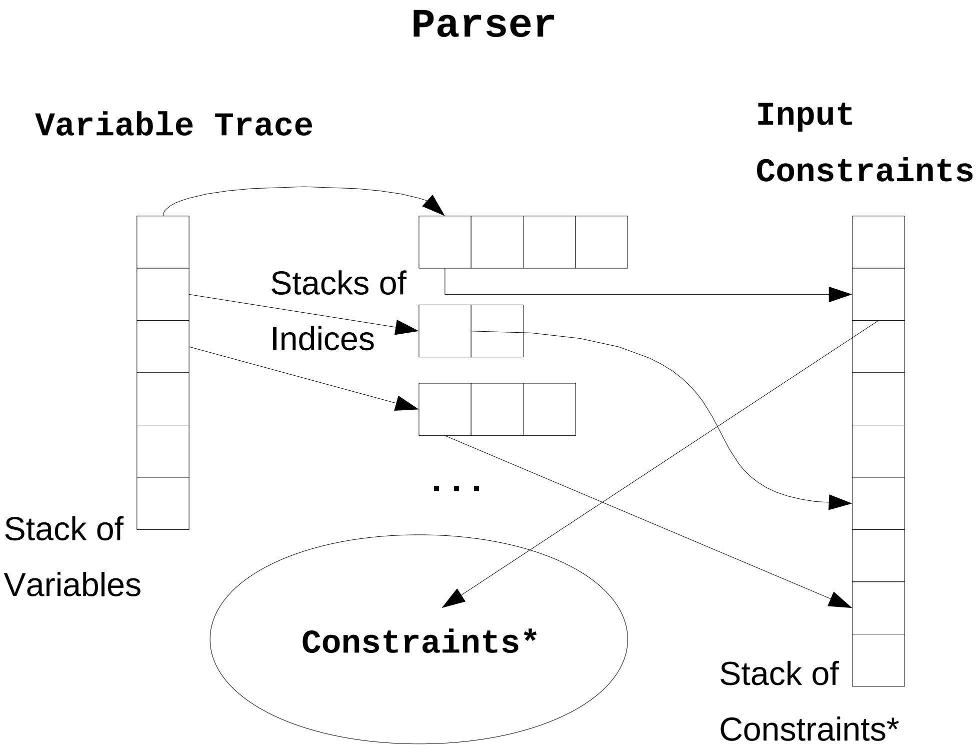


Figure 4.2: Data Structure Design

After parsing is done the solver calls a subroutine responsible for sorting and processing the input constraints. Before describing this part of the implementation we need to present another module of the system, called *Levels*.

4.1.2 Levels

The notion of a *level* was defined earlier, in Chapter 3. Each constraint is at some level, depending on the highest variable occurring in it. The number of levels corresponds to the number of variables in the system. Let $k > 0$. Each constraint at level k is of one of the types: (i) $x_k + p \diamond 0$, (ii) $-x_k + q \diamond 0$ with $\diamond \in \{\geq, >\}$, and (iii) $x_k + r = 0$. We group constraints of the same type, so that on each level we have three groups of constraints.

During the run-time of the conflict resolution algorithm we iteratively adjust the assignment to the variables. This demands keeping updated the bounds on the values of variables. In many cases the bounds on a variable remain unchanged. This makes it reasonable to store the current bound values for each variable and update them only when needed. Thus we characterise a level with a structure comprised of three groups of constraints and two bounds for the corresponding variable. If the interval is half-bounded the bound of the corresponding end is set to *'unbounded'*.

```
struct BLRI {
    Stack <Constraint*> LI;          // Stack of inequalities of type (i)
    Stack <Constraint*> RI;          // Stack of inequalities of type (ii)
    Stack <Constraint*> MI;          // Stack of equations (iii)
    Bound BLI;                       // Bound for LI
    Bound BRI;                       // Bound for RI
}
```

We store the values of bounds together with the indices of the constraint giving these bounds. See Figure 4.3. This information is kept and used for effective choice of conflicts when applying the Conflict Resolution (CR) rule (see Section 3.1).

Indeed, the bounds of the interval are used either for updating an assignment of the corresponding variable (AR rule in Section 3.1), or for concluding that the level is conflicting. If for some positive integer k at level k we have more than one k-conflict, choice of a conflict can be realised with various heuristics. One of the heuristics we studied selects a conflict with maximal overlap, i.e. selects the one defining the bounds on the variable x_k in the interval $I(S, \sigma, k)$ (see Section 3.1). Within such choice it becomes essential to keep together with bounds the indices of the constraints defining them.

For the implementation of some other heuristics for choosing a conflicting pair of constraints in the CR rule, instead of two bounds, it is essential to keep bounds exposed on the variable by all the constraints. We keep these bounds in two sorted stacks. Further details of these heuristics and their implementation are given later in this chapter.

Now we describe how the processing of input constraints is implemented.

4.1.3 Order

The last, and one of the most important, design component of our algorithm is *Order*. This component is responsible for ordering the variables.

In order to invoke the conflict resolution algorithm we first initialise it, i.e. populate levels with the input constraints. In order to populate the levels, we should have an order on variables. Thus, order of variables plays a crucial role for the efficiency of the conflict resolution algorithm: different choices of the order define different initialisation of the conflict resolution algorithm and substantially

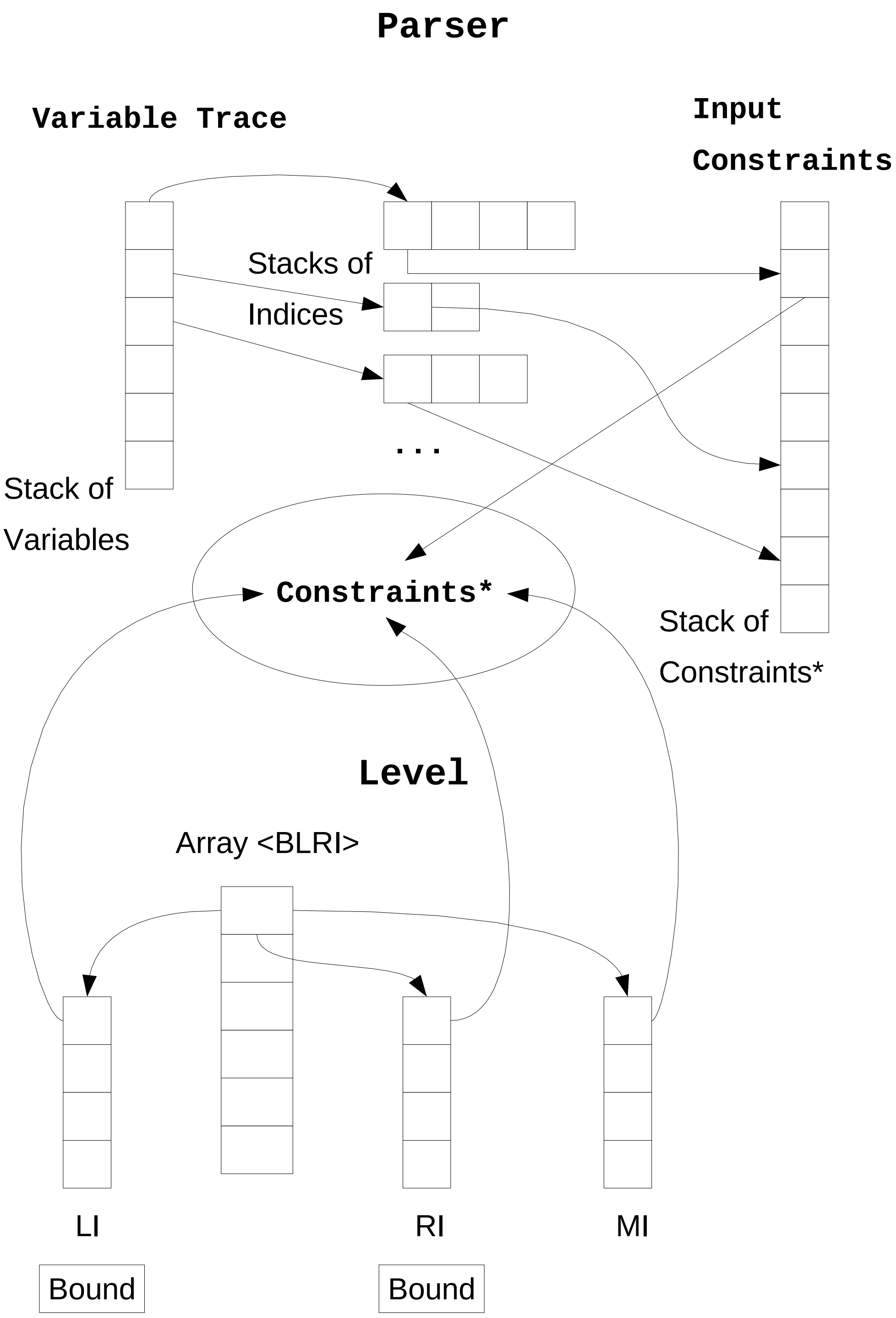


Figure 4.3: Data Structure Design with Levels

affect the course of the algorithm. This means, for each instance of the input data some variable orders might be more efficient than others and we are interested in obtaining the most effective one.

Also, in order to incorporate a linear arithmetic solver in various applications it is essential and desirable that order on variables has a flexibility to be defined (i) at run-time and (ii) dynamically. The conflict resolution algorithm allows such flexibility.

(i) Let us first see how we can set variable order at run-time. For this we first define order partially (on some of the variables) and successively extend it to all variables. This is possible due to an important feature of the conflict resolution algorithm: CRA can be invoked on subsets of constraints as well. In such case we introduce order only on the variables present in the subset. Based on such a *partial* order we populate levels *partially*, namely, populate only those corresponding to the variables present in the current order. We say that CRA is initialised *partially* if an order on the variables is set, and levels are populated, partially. Otherwise, initialisation of CRA is complete.

We explain the partial initialisation of CRA in more detail below: say for some positive integer k we have selected variables $x_1, \dots, x_k$, let $S_k \subset S$ be a subset of constraints containing these variables only. Without loss of generality we can assume order $\succ$ to be defined partially on these variables only, as follows: $x_k \succ, \dots, \succ x_1$. Then, we partially populate levels from 1 to k with constraints from S_k and invoke CRA being partially initialised.

If S_k turns out to be unsatisfiable, the entire system S is also unsatisfiable. Otherwise, we can extend the order on one more variable by introducing a new highest variable, say x_{k+1}, and populate level $k+1$. Constraints populating level $k+1$ will be those constraints in S which contain only the variables $x_1, \dots, x_k, x_{k+1}$, thus have the highest variable x_{k+1}. This allows defining the order on variables at run-time of the CRA algorithm using partial orders on variables.

(ii) Note that the CRA algorithm does not require fixing an initial order on variables. When processing to level k it allows dynamically changing the ordering if the following property of the current ordering is maintained, namely: all variables exceeding x_k in the current ordering remain exceeding x_k in the new ordering, and all variables preceding x_k in the ordering remain preceding x_k in the ordering. This allows dynamically changing the ordering during the run-time.

Possible advantages offered by this flexibility are subject to further studies. However, one obvious benefit of such a feature of the algorithm is for SMT solving, where DPLL-based techniques incorporate the idea of dynamic changes of variable orderings.

Based on the above observations we implemented our algorithm so that it allows defining the order on variables both statically and dynamically. Note, that the order can be set statically at different stages: before initialisation of the CRA algorithm, during initialisation, and at run-time (partial initialisation of CRA). Run-time setting of the order is also used for dynamic ordering. In all cases we need to populate levels at least partially. The next section shows how we order the variables at run-time, process constraints and populate levels.

4.1.4 Processing Input Constraints

In this section we discuss a part of the implementation responsible processing constraints and populating the levels. There are two things we need to do to populate levels:

1. Sort the coefficients in each constraint according the order on variables,
2. Process (assign) each constraint the level it belongs to.

In order to process constraints and populate levels, we need to have at least a partial order on the variables. We implemented a run-time setting of variable order which can also be used for ordering variables before and during a complete initialisation of CRA. The process is following.

We populate levels one-by-one starting from the lowest level and sort the input constraints right before allocating them at the corresponding level. For this, we use the variable counter included in the implementation of a constraint, see Section 4.1.1. Once a constraint is created its variable counter is set to the number of variables with non-zero coefficients in the constraint. When processing the constraints we start with the lowest variable, to populate the lowest level first, and walk through our *trace variables* data structure, see Section 4.1.1. We decrease the value of the variable counter in each constraint containing the lowest variable. We proceed the same way with each variable. As soon as a counter in a constraint becomes 0 for some variable x_k, this means the level of this constraint is k. At this point we sort variables in this constraint and push it to level k,

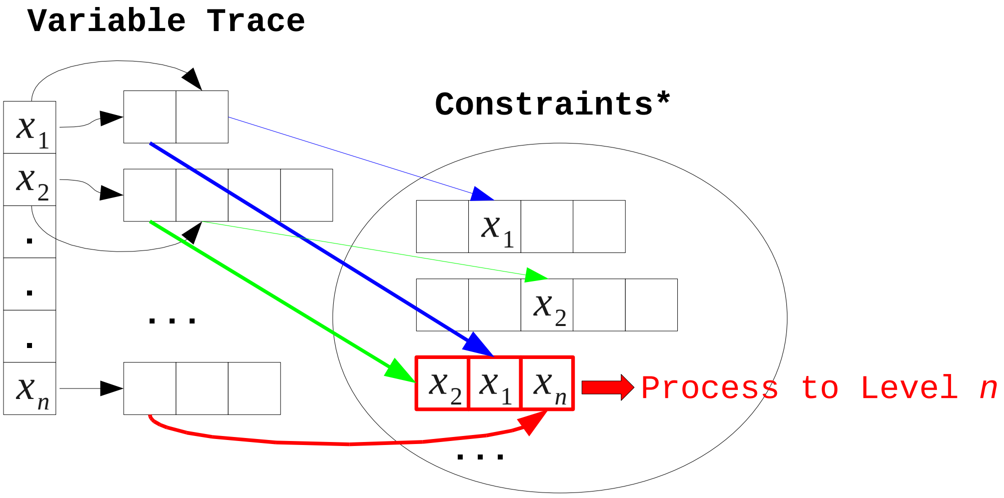


Figure 4.4: Sorting Input Constraints

thus populating all levels consecutively starting from the lowest one. Numeric constraints are processed in place; If a numeric constraint is true it is simply removed from the system, otherwise the system is unsatisfiable and the solver terminates with the output UNSATISFIABLE. Sorting and processing of the initial constraints is illustrated on the Figure 4.4.

In our experiments we studied two strategies for selecting the order of variables. The first strategy defines order randomly and statically, before invoking the CRA algorithm on the entire problem (before complete initialisation of CRA), the second, also defines the order statically and before invoking the CRA algorithm on the entire problem, but analyse the input data and sets the order during the complete initialisation of the CRA. This strategy is discussed in more detailed later in Section 4.2.3.

4.2 Implementation

As a part of the research presented, we implemented a solver based on the conflict resolution method. The implementation process spanned three phases. The first phase, Phase I, embodied a straightforward, early stage, implementation of the solver. It aimed at constructing an effective framework, that would allow easy expansion of the implementation on demand for further refinement of the algorithm. This phase of the implementation allowed the conducting of the first experiments, and yielded the first estimates of the performance of our algorithm.

The second phase, Phase II, was a pilot implementation of the solver. It enhanced the initial implementation from Phase I with various heuristics. It provided an empirical environment for the development and study of different heuristics for fine-tuning the algorithm and improving its efficiency. The third phase, Phase III, included the implementation of simple pre-processing in combination with fine-tuned heuristics.

Next, we discuss the implementations of each phase in more detail.

4.2.1 Phase I - Early Stage Implementation

Our implementation follows Algorithm 1. The algorithm allows fine-tuning with the following key parameters reflecting the choice of the strategies for: (i) selecting conflicts, (ii) selecting values in the assignment refinement rule and (iii) the order of variables.

The design of the implementation addresses the issue of tuning these parameters by including separate modules responsible for the selection in each case. The main algorithm works independently of these modules, but integrates them into the implementation.

The first phase of the implementation incorporated the heuristics used when describing the conflict resolution algorithm in the previous chapter. For readers' convenience, we recall the chosen heuristics here:

- select conflicts with maximal overlap, i.e. a pair of constraints defining the bounds of the interval for the value of the corresponding variable;
- update the assignment with the rational number closest to the mid point of the interval with the least power of 2 in the denominator;
- set the order of variables at random.

In our implementation we deal with linear constraints over the rationals. For handling arbitrary-precision rational numbers we use the GNU Multiple Precision Arithmetic Library (GMP).

The implementation works with linear constraints of the form $q \diamond 0$, for $\diamond \in \{\geq, >, =\}$.

4.2.2 Phase II - Implementation with Various Heuristics

In the second phase of the implementation we enhanced our solver with several heuristics for choosing the key parameters introduced earlier in this chapter. The following presents our implemented heuristics.

4.2.2.1 Strategies for Selecting Conflicts

The issue of selecting a conflicting pair of constraints arises naturally when more than one conflicting pair occurs at a level. We implemented a number of different strategies.

1. *Algebraic approach.* One of the strategies we tried is based on *maximal overlaps*, defined as follows. At line 7 of Algorithm 1 we select a k-conflict $x_k + p \geq 0$ and $-x_k + q \geq 0$ in S (i.e. $p\sigma + q\sigma < 0$), such that $-p\sigma = L(S, \sigma, k)$ and $q\sigma = U(S, \sigma, k)$. To explain the rationale behind this strategy we refer to Lemma 3.3.2. This lemma defines a notion of "almost" non-redundant constraints. Based on it we conclude that a choice of such a conflict guarantees that the constraints $x_k + p \geq 0$ and $-x_k + q \geq 0$ are "almost" non-redundant in the sense of Lemma 3.3.2.

2. *Geometric approach.* Another strategy comes from the geometrical ideas behind the relaxation method described in Section 2.4.4. As we know, an assignment σ represents a point M in n-dimensional space. The relaxation method iteratively changes the assignment, trying to get inside the polyhedron defined by the constraints in the system S. A new assignment is chosen by reflecting M over a hyperplane that (i) is defined by a constraint in S violated by M, i.e. M is outside the feasible area defined by a hyperplane of one of the facets of the polyhedron, and (ii) is at a maximal distance from M. A constraint defining such a hyperplane is called a *most violated* constraint. This method has a substantial drawback – each iteration leads to the solving of a new problem. However, the idea of reflection over the hyperplane of the most violated constraint is itself geometrically attractive. We integrated this idea into our algorithm as a conflict selection criterion: choose a conflicting pair of constraints with the most violated resolvent. In contrast to the relaxation method, our algorithm with the same conflict selection criterion does not require solving a new problem after each iteration.

3. *Random choice approach.* Yet another option we implemented is drawing a conflict with equal probability, which is a natural option and a rather essential one in experiments.

4. *Take the first.*

   The last strategy we tried simply takes the first detected conflict. It obviously saves the calculation time needed for evaluating all conflicts at a level, but undermines some efficiency related qualities of the conflict resolution algorithm.

4.2.2.2 Strategies for Selecting Assignment Values

We tried several strategies for selecting values in the assignment refinement rule, as listed bellow:

1. *Middle point.* Select the mid point if the interval $I(S, \sigma, k)$ (line 11).

2. *Maximal (minimal) point.* Always select the maximal (or always the minimal) endpoint of the interval $I(S, \sigma, k)$ (line 11).

3. *Random choice.* Another strategy we tried is a random choice of the assignment value.

4. *Interleaved points.*

   Select alternately the maximal and the minimal endpoints of the interval $I(S, \sigma, k)$ each time the interval is updated.

   Our experiments show that using these strategies frequently results in a rapid growth of the sizes of numerators and denominators of rational values in the assignment. In order to avoid this problem we used the following strategy for selecting assignment values in the assignment refinement rule.

5. *Closest binary or middle point.*

   First, if the endpoints of $I(S, \sigma, k)$ coincide, then we select one of them. Otherwise, we select a rational number n/m in $I(S, \sigma, k)$ such that (i) m is the least power of 2 among denominators of all rationals in $I(S, \sigma, k)$, and (ii) n is such that, n/m is the closest rational to the middle point of the interval, among all rationals satisfying (i). It is possible to show that a rational satisfying both (i) and (ii) always exists. In particular, if $I(S, \sigma, k)$

contains integer points, then our strategy will select an integer in $I(S, \sigma, k)$ closest to the middle point. As our experiments show, such choice of values considerably simplifies the assignment values and constraint evaluation. See more details of the experiment results in Chapter 5.

6. *Other strategies.*

   Other strategies may also appear useful. For example one can choose the conflict leading to the resolvent with the least possible level.

Note, heuristics for setting the order on the variables have been studied at the third phase of implementation and will be discussed in the next paragraph. For the implementation at the second phase we used the randomly set order on the variables.

4.2.2.3 Other Heuristics

There are also other heuristics that we considered interesting to study. One of them concerns *adding resolvents* to the current system during the run-time of the conflict resolution algorithm. Processing a resolvent in the CRA algorithm may result in a new conflict instantiated by this resolvent at the level it belongs to. Adding all resolvents obtained by resolving such consecutive conflicts may result in a quick expansion of the system and adding only the final resolvent (which instantiates no conflict) may improve the performance of the algorithm. We studied the following heuristics:

1. *Adding resolvents*

   - Add all resolvents derived during the run-time of the algorithm.
   - Do not add a resolvent if it instantiates a new conflict at the level it belongs to. Keep resolving conflicts without adding this resolvent and add only the final resolvent, that is, when it instantiates no conflict. This process is associated with 'jumping' to the lowest level (the first non-conflicting level) and adding the last resolvent at this level only.

Two other heuristics concern the issues of (i) dealing with half-bounded intervals in the AR rule and (ii) reducing constraints by the greatest common divisor of their coefficients. In experiments half-bounded intervals occur very frequently in the assignment refinement rule when assigning a value to a variable. In such case we introduce an artificial bound to form the interval. If the CRA keeps returning to a level with a half-bounded interval, it may become essential to increase the size of the interval repeatedly. We tried two heuristics for dealing with half-bounded intervals:

2. *Dealing with half-bounded intervals in the AR rule*

   – Increase the length of the interval exponentially, by a power of 2 each time, when returning to the same half-bounding level.

   – Keep the length of the interval constant, equal to 10, on every return to the same half-bounding level.

The last heuristic implemented responds to an important problem of decreasing size of numerators and denominators during the calculations. Namely, if all coefficients of some constraint have the greatest common divisor different from 1 then the constraint can be reduced by dividing each of the coefficients by their GCD. This simple idea evolves into the following heuristic:

3. *Reducing constraints by GCD*

   – Always reduce each constraint by the GCD of its coefficients;

   – Never reduce any constraint by the GCD.

4.2.3 Phase III - Simple Preprocessing with Highly-Tuned Heuristics

The third phase of the implementation included experiments with the dynamic ordering of the variables and simple preprocessing of the input data. We came up with the idea of simple preprocessing when observing real-life benchmarks during our experiments. These benchmarks contained several hundreds of variables and constraints. On many occasions our solver continuously passed a considerable number of levels, which expanded the system, without any contribution to the solving process. To avoid such long runs we came up with preprocessing ideas that are very natural and simple. In the following we first present our heuristic for the dynamic variable ordering, then describe the above mentioned circumstances and discuss the details of the preprocessing we used.

4.2.3.1 Strategies for Setting the Order

The last parameter of the CRA algorithm that we consider here is the order on variables. In the current implementation we tried two different approaches for setting the order on variables. The first sets the order on all variables randomly, statically, before initialising the CRA algorithm and invoking it on the entire problem. The second, also sets the order statically before invoking the CRA algorithm for the entire problem, but while initialising CRA and takes into account the structure of the input set of constraints.

We analyse the input data and define the order on variables as follows. We characterise each variable x_i with a pair (l_i, t_i), where l_i is the length of the shortest constraint containing the variable x_i and t_i is the number of such constraints. We set ordering $x_i \succ x_j$ for some $0 \geq i, j \geq n$ if (i) $l_j < l_i$ or (ii) $l_i = l_j$ and $t_i < t_j$.

Thus strategies for *setting the order on variables* are:

- set the order at random;
- set the order based on the analysis of the input data;

We believe that a careful selection of the order on variables based on the properties of the input problem may have a considerable impact on the performance of the implementation and studying order selection is one of the future research directions.

4.2.3.2 Preprocessing to Avoid Half-Bounding Levels

In this section we define a notion of *a half-bounding level* and describe our preprocessing idea to avoid such levels. Let us note, that if all occurrences of some variable x_k in the system are of the same sign, then x_k will never be eliminated with the CR rule. Moreover, if a derived constraint contains the variable x_k, the coefficient of x_k in this constraint will be of the same sign too. We will call such a level k – *half-bounding level.* If a level k is half-bounding it may expand during the run-time of the CRA algorithm but will never give a bounded interval for x_k.

In such instances the conflict resolution algorithm keeps running along half-bounding levels without obtaining both bounds for the variables during the whole run-time.

To avoid such situation we can remove from the system all constraints containing x_k. Solve the remaining system of constraints. If it has no solution, then the original system has no solution either. Otherwise, we assign a value to the variable x_k based on the values assigned to the remaining variables. Since the interval for x_k is half-bounded such an assignment always exists.

Removing all such variables from the system reduces both the number of variables in the system, and the number of initial constraints, as well as those derived at run-time.

4.2.3.3 Preprocessing to Avoid Almost Half-Bounding Levels

More preprocessing ideas came with the observation of what we called *almost half-bounding levels*. The difference between an almost half-bounding level and a half-bounding level is in one constraint. This constraint is a so called *unit* constraint, containing one variable only (the highest variable). Say at level k this unit constraint is either (i) an equality $x_k = a$ (where $a \in \mathbb{Q}$), or (ii) an inequality that has the coefficient of x_k of opposite sign to the sign of the coefficients x_k in the remaining constraints.

If all occurrences of the variable x_k in the system have the same sign and case (i) holds, the unit constraint $x_k = a$ explicitly assigns a value to the variable x_k. Therefore, it becomes useless to keep x_k in the other constraints. We can directly eliminate x_k from the system by simply substituting all occurrences of x_k by a. Obviously, this brings us to an equivalent system with one variable less. If the new system has a solution, the solution of the initial system is easily obtained by adding the assignment of the value a for the variable x_k. Note that we can eliminate from the system all such variables one after another, thus reducing the dimension of the system.

Case (ii) also allows elimination of the variable x_k from the system. The only condition we ask for is, that all occurrences of x_k in the system are of the same sign, opposite to the one of the unit constraint. In such a case, we can eliminate the variable x_k from the system by simply summing up the unit constraint with the rest of the constraints containing x_k. Obviously, this operation also results in an equivalent system of constraints, solving which we obtain a solution to the initial one. If the derived system has no solutions, the initial system has no solutions either. Otherwise, we build a solution to the initial system by extending the solution of the derived system on the variable x_k. The value for the variable

x_k is obtained from the interval defined by the initial constraints containing x_k, by simply substituting the values of the other variables. Similarly to the previous cases, it is possible to eliminate all such variables one after another, reducing the dimension of the system this way.

Elimination of both half-bounding and almost half-bounding levels from the initial system reducing the dimension of the system and can considerably simplify the initial problem. Thus, the conflict resolution algorithm appears to be quite sensitive to such preprocessing, this is also confirmed by the experiments in Section 5.3.3.

Chapter 5

Experiments

In this chapter we present results of our experimental evaluation of the conflict resolution method. As mentioned in the implementation chapter, there were three phases of the software implementation:

(i) Phase I – straightforward (early stage) implementation,

(ii) Phase II – pilot implementation (enhanced with various heuristics),

(iii) Phase III – implementation with simple preprocessing and fine-tuned heuristics.

Series of experiments were conducted at each implementation phase.

This chapter describes the benchmarks used in our experiments along with the tools used for benchmark generation, and outlines the results of the experiments in three sections spanning each of the implementation phases.

The algorithm was evaluated against different types of input data and compared to some other well-known approaches, namely the Fourier-Motzkin elimination method, a modification of the Fourier-Motzkin elimination method, and the simplex method. We evaluated our solver on two types of benchmarks: randomly generated problems and benchmarks extracted from real-life problems.

The first type of benchmarks is comprised of randomly generated systems of linear constraints, which we generated using GoRRiLA – a tool for the generation of random benchmarks [31].

The second type of benchmarks consists of real-life benchmarks, namely systems of linear constraints extracted from the SMT-LIB – a library of real-life benchmarks for Satisfiability Modulo Theories [2].

In general, real-life problems differ considerably from randomly generated ones. They differ not only in size but also in structure. First, the number of variables and constraints is considerably higher in real-life problems; most of the problems contain several hundreds of variables and constraints. On the other hand real-life problems deal with sparse matrices and relatively simple coefficients.

We used three sets of real-life benchmarks: Two of them were generated using the Hard Reality Tool (HRT) [31]. The difference between these two benchmarks is in their difficulty levels.

Both of these benchmarks, as well as the randomly generated benchmarks, are available on the web [1].

The third set was provided by Leonardo de Moura for experimental purposes, and is also extracted from the SMT-LIB real-life benchmarks. These benchmarks were also used in other works (e.g. [36]) and are available on the web [2].

In our experiments we compared the conflict resolution algorithm with various implementations of the Fourier-Motzkin elimination method, including the Chernikov modification, and the simplex method. Some of these algorithms were reimplemented by us in the same framework as the conflict resolution algorithm, to make the results more comparative. To evaluate the performance of our implementation some algorithms were taken from the state of the art SMT solvers: CVC3 [4], Barcelogic [40] and Z3 [11].

In the following sub-sections we describe the benchmarks we used and give results of our experiments for each phase of implementation.

5.1 Benchmarks with Randomly Generated Problems

For our experiments with random problems we used random benchmarks with integer coefficients generated using the GoRRiLA tool. GoRRiLA was developed at the Department of Computer Science at the University of Manchester by K. Korovin and A. Voronkov. It is a generator of random problems for propositional logic and for systems of linear constraints over the rational or integer numbers (in SMT format).

[1] http://www.cs.man.ac.uk/~tsiskarn/CRA_bench

[2] http://www-verimag.imag.fr/~monniaux/simplexe/

GoRRiLA permits generating the bunches of random problems with the number of variables increasing within a specified range. Characteristics of random problems generated by GoRRiLA depend on several parameters listed below:

- range for the number of variables in the system;
- number of problems generated with the same number of variables;
- range for the values of the coefficients in a constraint;
- range for the number of variables with non-zero coefficients in a constraint;
- ranges for the number of constraints of each type (equalities, disaqualities, non-strict inequalities, strict inequalities);

Note, that (i) the number of constraints in a system is the total of the numbers of constraints of each type, (ii) ranges for the number of constraints of each type can be specified proportional to the number of variables.

The randomness of size and structure of the generated problems is provided by selecting randomly values of the following parameters:

- The values of non-zero coefficients are randomly selected from the specified range (excluding 0).
- The number of variables with non-zero coefficients in a constraint is randomly selected from the specified range.
- The numbers of constraints of each type are randomly selected from the specified range.

Our random generated experimental benchmarks are available on the web [3].

5.2 Benchmarks Extracted From SMT-LIB With Hard Reality Tool

In order to study the performance of our solver on real-life problems, we ran a series of experiments with real-life benchmarks extracted from the SMT-LIB.

We obtained a set of real-life benchmarks using the Hard Reality Tool (HRT).

[3] `http://www.cs.man.ac.uk/~tsiskarn/CRA_bench`

HRT allows randomly extracting hard and realistic theory problems from SMT problems. The extracted theory problems are given as a conjunction of constraints from this theory.

For our experiments we used the benchmarks from the QF_LRA division of the SMT-LIB – the benchmarks consisting of quantifier free SMT problems with the theory of linear real arithmetic.

The following describes how the HRT works to extract linear arithmetic problems from the QF_LRA benchmarks.

1. HRT searches for random theory problems
2. From the selected theory problem it extracts linear arithmetic terms;
3. Reduces linear arithmetic terms to the form:

   $$(\textbf{and}\ [(>\ \mathit{lin}\ 0)]...[(>=\ \mathit{lin}\ 0)]...[(=\ \mathit{lin}\ 0)]...)$$

   where *lin* represents various linear forms over the variables $x_1, x_2, \ldots, x_k$ with the coefficients $c_0, c_1, \ldots, c_k$ expressed as a sum $(+(*c_i x_i)...(*c_1 x_1) c_0)$ where all variables with 0 coefficients are omitted and no repetition of variables occurs.
4. Optionally, a difficulty level of the extracted problem can be specified as follows:
   (a) Accept only those input SMT problems which require a time to solve, longer than a predefined time bound.
   (b) Output only those extracted problems which require a time to solve, longer or equal to the predefined time bound.

For more options, the HRT also allows the extraction of a maximal satisfiable subset and a minimal unsatisfiable subset of constraints (conjunction of constraints) among the obtained random theory problems.

Our experimental benchmarks extracted with HRT are available from the web [4].

[4] `http://www.cs.man.ac.uk/~tsiskarn/CRA_bench`

5.3 Experimental Results

In the following we present the results of our experiments for each implementation phase.

5.3.1 Phase I - Early Stage Implementation

The experiments at the early stage of the implementation were presented in our paper where the conflict resolution method was introduced [30]. To recall, early stage implementation had neither any heuristics crucial for efficiency nor any kind of preprocessing. All experiments at this stage were run on a Linux laptop with CPU 2.8GHz and memory 4Gb.

The conflict resolution algorithm was compared with our implementation of the Fourier-Motzkin elimination method and its modification - the Chernikov algorithm, both implemented in the same data structures as CRA for better transparency in comparison [30]. The conflict resolution algorithm was compared with these implementations as well as with CVC3 [4] and Barcelogic [40] – well-developed solvers for satisfiability modulo theories (SMT). CVC3 incorporates a variant of the Fourier-Motzkin algorithm and Barcelogic incorporates the simplex algorithm for reasoning with linear arithmetic.

First experimental results were very encouraging, showing that the conflict resolution algorithm is considerably more efficient in solving linear constraints than the standard Fourier-Motzkin algorithm. For example, an order of magnitude difference occurs already on small problems.

5.3.1.1 Randomly Generated Problems

We had two sets of random benchmarks:

1. 4000 randomly generated problems with the number of variables ranging from 3 to 12

2. 400 randomly generated problems with the number of variables ranging from 13 to 22

Results for the randomly generated problems are shown in Table 5.1.

The conflict resolution algorithm has solved all 4000 randomly generated problems with the number of variables ranging from 3 to 12 (within the total time of

4000 problems vars 3-12 (unsat/sat)				
	CRA	**CVC3**	**FM**	**Ch**
timeout (20s)	0/0	11/9	790/329	149/10
av. time (s)	0/0	0/0	0.4/0.1	0.6/0.1

400 problems vars 13-22 (unsat/sat)				
	CRA	**CVC3**	**FM**	**Ch**
timeout (20s)	5/2	21/33	183/144	155/65
av. time (s)	0.2/0.3	0/0	0.1/0.5	1.9/0.6

Table 5.1: Randomly Generated Problems

7 seconds) and on the problems with the number of variables ranging from 13 to 22 fails only on 7 (5 unsatisfiable and 2 satisfiable problems).

The CVC3 implementation of the Fourier-Motzkin algorithm fails to solve 20 (11 unsat / 9 sat) problems from the first set of benchmarks and 54 (21 unsat / 33 sat) problems from the second set. Our implementation of the Fourier-Motzkin algorithm solves considerably fewer problems than CRA. The Chernikov algorithm improves over the Fourier-Motzkin but solves considerably fewer problems than CRA and even than CVC3, see the Table 5.1.

The difference in performance between Chernikov and CVC3, i.e. between the enhanced and elaborated realisation of the Fourier-Motzkin elimination method respectively, point to the expected difference in performance between the straightforward and more elaborate realisation of CRA.

One of the most striking examples showing the algorithmic difference in performance of the Fourier-Motzkin elimination method and CRA is shown in Figure 5.1. The problem in this figure was randomly generated and contains 5 variables and 10 linear constraints.

The standard Fourier-Motzkin algorithm run on this problem generated over 280 million linear constraints, while the conflict resolution algorithm generated only 21 constraints. However, this example is not exceptional as compared to our other experiments with early stage implementation.

$$
\begin{array}{rcrcrcrcrcrcl}
2x_5 & - & 3x_4 & + & x_3 & - & 3x_2 & - & 2x_1 & + & 3 & \geq & 0 \\
2x_5 & + & x_4 & - & 2x_3 & & & - & 2x_1 & + & 2 & \geq & 0 \\
-x_5 & & & & & + & 3x_2 & + & x_1 & + & 2 & \geq & 0 \\
-3x_5 & & & + & 2x_3 & & & - & 3x_1 & - & 2 & \geq & 0 \\
x_5 & - & 2x_4 & & & - & 2x_2 & + & 3x_1 & - & 2 & \geq & 0 \\
-2x_5 & + & 2x_4 & - & 3x_3 & - & x_2 & + & 2x_1 & + & 3 & > & 0 \\
3x_5 & - & 2x_4 & + & 2x_3 & + & 3x_2 & + & 2x_1 & + & 1 & > & 0 \\
x_5 & & & & & & & + & 2x_1 & + & 2 & > & 0 \\
 & & 2x_4 & - & x_3 & - & 3x_2 & - & x_1 & + & 3 & = & 0
\end{array}
$$

Figure 5.1: A randomly generated problem

304 problems (unsat)				
	CRA	**CVC3**	**FM**	**Ch**
timeout (60s)	1	4	44	42
av. time	0.2	0.13	0.1	0.12

Table 5.2: Hard Reality Problems

5.3.1.2 Real-Life Problems

Table 5.2 compares the solvers on the problems extracted from SMT benchmarks using the Hard Reality Tool.

As in randomly generated benchmarks, the CRA also solves more problems in the real-life benchmarks than any of CVC3, Fourier-Motzkin, and Chernikov algorithms. The average time of the CRA is a bit higher than of CVC3 due to additional time needed to solve extra problems that were not solved by CVC3. Indeed, in a pairwise comparison on all solved problems in these benchmarks the CRA is faster than CVC3.

Compared to the Simplex algorithm, the conflict resolution algorithm already at its early stage (non-optimised) implementation showed promising potential. In Table 5.3 the CRA is compared to Barcelogic. In the low-range problems (with 12-22 variables) we had 400 problems with 197 unsat and 203 sat. Both solvers showed equal performance in about 78% of the problems (74% of unsat problems and 82% of sat). In the remaining 22% of the problems CRA timed out on 7% and was faster than Barcelogic in about 45% of unsat problems and 19% of sat. In the high-range problems (with 23-32 variables) we had 400 problems with 204 unsat and 196 sat. Both solvers showed equal performance in about 30% of

400 problems vars 13-22 (unsat/sat)				
	faster	**same**	**av. time**	**timeout (20s)**
Barcelogic	28/29	146/167	0.04/0	0/0
CRA	23/7	146/167	0.2/0.3	5/2

400 problems vars 23-32 (unsat/sat)				
	faster	**same**	**av. time**	**timeout (20s)**
Barcelogic	110/67	31/88	0.25/1.0	0/0
CRA	63/41	31/88	0.7/1.6	60/37

Table 5.3: CRA vs Barcelogic on real-life HR benchmarks

all problems, though there is a considerable difference in handling sat and unsat problems. Only 15% of unsat problems were solved with equal performance while for sat the percentage reached 45%. From the remaining 70% of the problems Barcelogic was faster in 63% (64% unsat and 62% sat) while CRA timed out in about 35% and solved faster in 37% (36% unsat and 38% sat).

We can conclude that CRA was faster than Barcelogic on a considerable number of the problems, although Barcelogic has solved more problems than CRA within 20 seconds.

To summarise, our experiments showed that an early stage implementation of the conflict resolution algorithm outperformed both the Fourier-Motzkin and the Chernikov algorithms in solving systems of linear constraints, and had promising potential compared to the simplex algorithm.

5.3.2 Phase II – Implementation with Various Heuristics

The second phase of the implementation extended the CRA algorithm with various heuristics (i) for choosing conflicts in the conflict resolution rule and (ii) for choosing values in the assignment refinement rule, (iii) for adding resolvents to the system, (iv) for dealing with half-bounded intervals in the assignment refinement rule, and (v) using optional reduction of constraints by the greatest common divisor.

The implemented heuristics are listed below:

(i) *Selecting conflicts*

1. Select a conflict at random;
2. Select the first conflict detected;
3. Select a conflict with the maximal bound overlap;
4. Select a conflict with maximal violated resolvent in the lower dimensional space (the Relaxation Method criterion);

(ii) *Selecting assignment values*

1. Select an assignment at random;
2. Select a closest binary assignment (an approximate mid point of the interval);
3. Select the maximal bound of the interval;
4. Select the minimal bound of the interval;
5. Swap maximal and minimal bounds at each call at a level;

(iii) *Adding resolvents*

We studied the following heuristics:

1. Add each resolvent independently of whether it instantiates a new conflict or not;
2. Do not add a resolvent if it instantiates a new conflict at the level it belongs to, keep resolving conflicts without adding this resolvent and add only the final resolvent, that is when it instantiates no conflict. This process is associated with 'jumping' to the lowest level (the first non-conflicting level) and adding the last resolvent at this level only.

(iv) *Dealing with half-bounded intervals in the AR rule*

As mentioned in Chapter 4, in experiments, half-bounded intervals occur very frequently in the assignment refinement rule when assigning a value to a variable. In such case we generate an artificially bounded interval that may serve as a substitution of half-bounded intervals in the assignment refinement rule. If the CRA keeps returning to a level with a half-bounded interval often, it may become essential to increase the size of the interval repeatedly. We tried two heuristics for dealing with half-bounded intervals:

1. Double the size of an artificial substitution of the half-bounded interval, each time when returning to the same half-bounding level;
2. Bound the half-bounded interval to the constant size of 10, every time when returning to the same half-bounding level.

(v) *Reducing constraints by the GCD*

1. Always reduce each constraint by the GCD;
2. Never reduce each constraint by the GCD.

We studied various combinations of these heuristics integrated into the CRA algorithm. We call the *major heuristics* the heuristics (i) for selecting conflicts, and (ii) for selecting assignment values. Two other heuristics (a) for dealing with half-bounded intervals in the assignment refinement rule and (b) for reducing constraints by the GCD of their coefficients are *general* in their nature and can be combined with any major heuristic mentioned above. Such a combination entails four different heuristic bundles for each of the major heuristics:

1. Bound the half-bounded interval to the constant size (size of 10). Do not reduce constraints by the GCD of their coefficients.

2. Bound the half-bounded interval to the constant size (size of 10). Reduce constraints by the GCD of their coefficients.

3. Increase artificial bounds in half-bounded intervals exponentially (by powers of 2). Do not reduce constraints by the GCD of their coefficients.

4. Increase artificial bounds in half-bounded intervals exponentially (by powers of 2). Reduce constraints by the GCD of their coefficients.

In the sequel we will use the term *'quadruple'* (or *quadruple of implementations*) when we refer to such a group of four implementations.

During the evaluation of the experiments we grouped the implementations into such quadruples. From each of the quadruples we selected the one with the best performance, and pairwise compared the selected implementations.

The full list of the bundles of the heuristics of the CRA algorithm, together with their abbreviations used throughout the thesis, is given in Table 5.4

We evaluated the implementations on both randomly generated benchmarks and real-life benchmarks extracted from SMT-LIB. The results of the experiments were plotted on three types of graph:

1. *'Number of variables versus number of solved problems.'* In such a graph the Y-axis plots the number of variables and the X-axis plots the number of solved problems. A point (x, y) on the graph indicates that x problems were solved among the problems with a number of variables less than or equal to y.

2. *'Time versus number of solved problems.'* In such a graph the Y-axis plots the time from 0 to the timeout limit, and the X-axis plots the number of problems solved. A point (x, y) on the graph indicates that x problems were solved in y time or less.

3. *'Total run-time versus number of solved problems.'* In such a graph the Y-axis plots the run-time, and the X-axis plots the total number of problems solved within the corresponding time. A point (x, y) on the graph indicates that in total x problems were solved within y time since the start of a solver.

In the following we present the results of the experimental evaluation. We ran experiments on the second and the third phases of implementation on Intel Xeon Quad Core machines with 2.33 GHz and 12 GB of memory.

5.3.2.1 Randomly generated benchmarks

For our experiments on Phase II we generated three sets of random benchmarks:

1. Randomly generated problems with a number of variables ranging from 3 to 10, 400 problems per variable number, 3200 problems in total;

2. Randomly generated problems with a number of variables ranging from 11 to 18, 200 problems per variable number, 1600 problems in total;

3. Randomly generated problems with a number of variables ranging from 19 to 26, 50 problems per variable number, 400 problems in total.

Bundles of heuristics of the CRA Algorithm

Abbreviations	**Select Conflict**	**Select Assignment**	**Add Resolvents**	**Half-Bounded**	**GCD**
MP	Maximal Overlap	Middle Point[a]	All	Const	No
MP_gcd	Maximal Overlap	Middle Point	All	Const	Yes
MP_pow2	Maximal Overlap	Middle Point	All	Exp	No
MP_pow2_gcd	Maximal Overlap	Middle Point	All	Exp	Yes
MAX	Maximal Overlap	Maximal Bound	All	Const	No
MAX_gcd	Maximal Overlap	Maximal Bound	All	Const	Yes
MAX_pow2	Maximal Overlap	Maximal Bound	All	Exp	No
MAX_pow2_gcd	Maximal Overlap	Maximal Bound	All	Exp	Yes
MIN	Maximal Overlap	Minimal Bound	All	Const	No
MIN_gcd	Maximal Overlap	Minimal Bound	All	Const	Yes
MIN_pow2	Maximal Overlap	Minimal Bound	All	Exp	No
MIN_pow2_gcd	Maximal Overlap	Minimal Bound	All	Exp	Yes
SW	Maximal Overlap	Swap Minimal and Maximal Bounds	All	Const	No
SW_gcd	Maximal Overlap	Swap Minimal and Maximal Bounds	All	Const	Yes
SW_pow2	Maximal Overlap	Swap Minimal and Maximal Bounds	All	Exp	No
SW_pow2_gcd	Maximal Overlap	Swap Minimal and Maximal Bounds	All	Exp	Yes
RA	Maximal Overlap	Random	All	Const	No
RA_gcd	Maximal Overlap	Random	All	Const	Yes
RA_pow2	Maximal Overlap	Random	All	Exp	No
RA_pow2_gcd	Maximal Overlap	Random	All	Exp	Yes
FC	First Conflict	Middle Point	All	Const	No
FC_gcd	First Conflict	Middle Point	All	Const	Yes
FC_pow2	First Conflict	Middle Point	All	Exp	No
FC_pow2_gcd	First Conflict	Middle Point	All	Exp	Yes
RM	Relaxation Method	Middle Point	All	Const	No
RM_gcd	Relaxation Method	Middle Point	All	Cons	Yes
RM_pow2	Relaxation Method	Middle Point	All	Exp	No
RM_pow2_gcd	Relaxation Method	Middle Point	All	Exp	Yes
RC	Random	Middle Point	All	Const	No
RC_gcd	Random	Middle Point	All	Const	Yes
RC_pow2	Random	Middle Point	All	Exp	No
RC_pow2_gcd	Random	Middle Point	All	Exp	Yes
CRJ	Maximal Overlap	Middle Point	Last only	Const	No
CRJ_gcd	Maximal Overlap	Middle Point	Last only	Const	Yes
CRJ_pow2	Maximal Overlap	Middle Point	Last only	Exp	No
CRJ_pow2_gcd	Maximal Overlap	Middle Point	Last only	Exp	Yes

[a] Rational closest to the interval middle point with the least power of 2 in the denominator

Table 5.4: List of the bundles of heuristics of the CRA algorithm

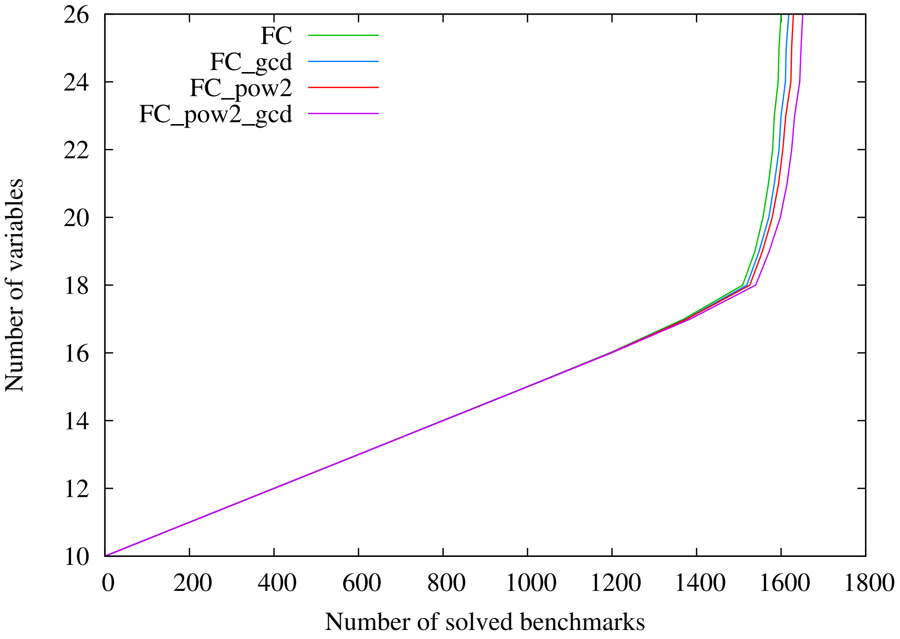


Figure 5.2: Implementations of the CRA for the major heuristics 'FC' on randomly generated set of benchmarks, 'number of variables versus number of solved problems'.

On random problems all implementations had certain similarities in performance and behaviour within one quadruple as well as for the same sets of major heuristics.

- The low-dimensional problems (with the number of variables upto 10), were solved in 0.0 seconds with all heuristic bundles listed above.
- In the middle-sized problems (with the number of variables ranging between 11 and about 18) all bundles of heuristics showed an insignificant difference in the number of solved problems and in performance.
- The difference in the number of solved problems became more significant as the number of variables in the problems increased, ranging between 19-26.

We plotted experimental data from problems with the number of variables ranging between 11 and 26. This observation is illustrated in Figure 5.2, plotting experimental results on 'number of variables versus number of solved benchmarks' graph for a quadruple of one of the major heuristics (abbreviated as FC in Table 5.4): (i) select the first conflict in the CR rule, and (ii) select as an assignment the middle point of the interval in the AR rule.

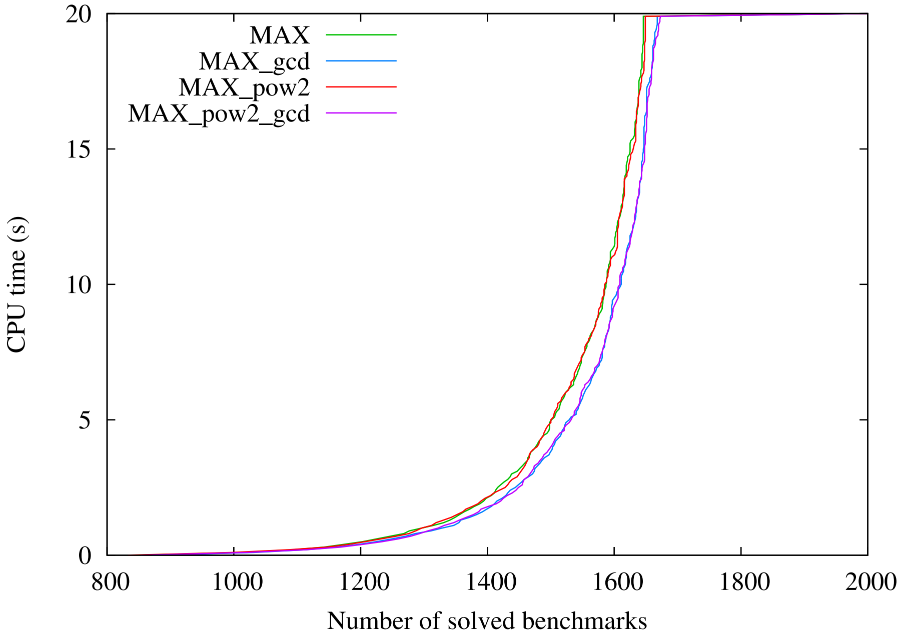


Figure 5.3: Implementations of the CRA for the major heuristics 'MAX' on randomly generated set of benchmarks, 'time versus number of problems solved'

Comparison of implementations of different quadruples resulted in the following observation.

Equal and nearly equal performances. In some quadruples we observed equal or nearly equal performances of several heuristics. These heuristics mainly differed only in choice of one of the general heuristics.

An example of such a nearly equal behaviour can be observed for a quadruple of implementations of the CRA algorithm with the set of major heuristics (abbreviated as MAX in Table 5.4): (i) select a conflict with the maximal overlap in the CR rule and (ii) select for an assignment the maximal bound of the interval in the AR rule.

Figure 5.3. plots experimental data for this quadruple of implementations on a 'time versus number of problems solved' graph. As we can see, two pairs of plots (*MAX*, *MAX_pow2*) and (*MAX_gcd*,*MAX_pow2_gcd*) indicate almost equal performances with slight superiority in the option of doubling the size of substituting bounded intervals.

This leads to the conclusion, that for this particular set of major heuristics (MAX), the difference in performances between heuristics of bounding substituting intervals to a constant size and bounding them by doubling their size is not

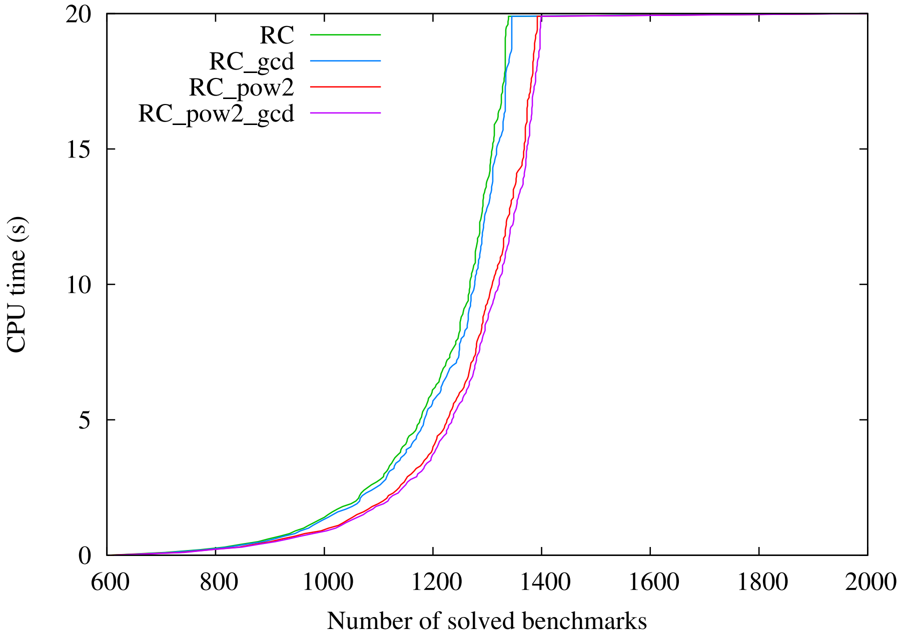


Figure 5.4: Implementations of the CRA for the major heuristics 'RC' on randomly generated set of benchmarks, 'time versus number of problems solved'

essential. Even though, there is, still, a slight superiority with doubling. Similar behaviours were observed for quadruples of CRJ and SW.

Yet another conclusion from these quadruples is that reducing constraints by the GCD of their coefficients leads to the best performance. This sounds reasonable. However, for a number of other sets of major heuristics, the effect of reducing constraints by the GCD was negligible.

For example, Figure 5.4. depicts the performance of a quadruple of the implementations of the CRA algorithm with the set of major heuristics (abbreviated as RC in Table 5.4): (i) select a conflict at random in the CR rule, and (ii) select as an assignment the mid point of the interval in the AR rule.

The chart shows nearly equal performance of the pairs of bundles of heuristics: (*RC*, *RC_gcd*) and (*RC_pow2*, *RC_pow2_gcd*), with slight superiority with reductions by the GCD. This means, that the improvement in performance caused by reduction of constraints by the GCD is negligible. However, this time doubling the size of the substituting bounded intervals appears to be considerably more efficient than bounding intervals to a constant size. The same behaviour is observed for the quadruple RA.

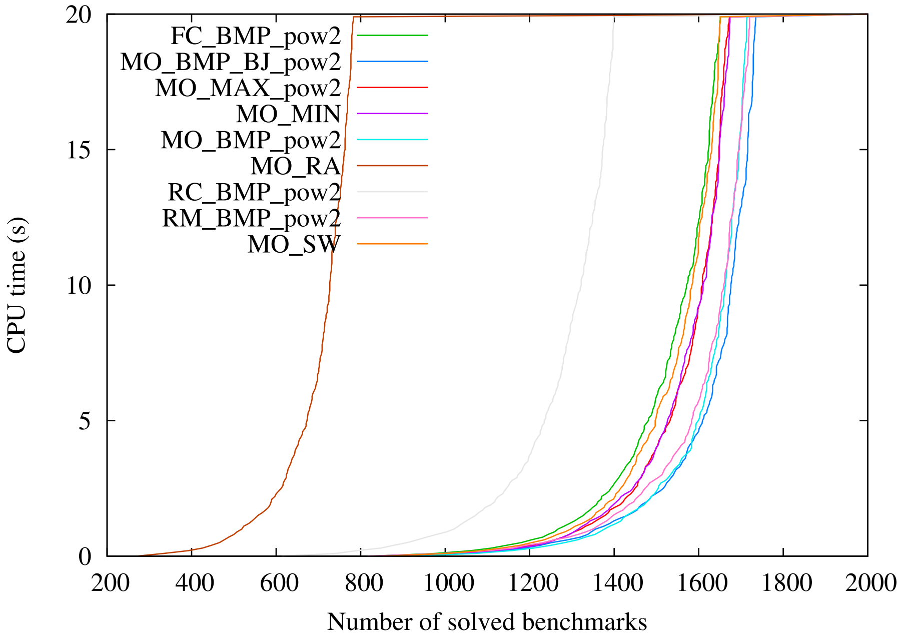


Figure 5.5: Implementations of CRA with the best performance of all major heuristics for the randomly generated set of benchmarks, ‘time versus number of problems solved’

Other quadruples had different performances for all four bundles of heuristics, (but still close to each-other) with the best performances shown by either ‘*pow*2*_gcd*’ or ‘*gcd*’.

Plots for each of the quadruples are presented in Appendix D.

The best performance within the quadruples. In almost all quadruples the best performance was shown when the constraints were reduced by the GCD, although, in some cases the difference is insignificant. Within the constraint reducing option, as a rule, the best performance was when substituting bounded intervals were doubled.

In almost half of the quadruples, selecting a constant artificial interval or doubling it showed nearly equal performance.

Conclusion. We plotted the best performances from each of the quadruples on one chart, to define the best choice of the set of heuristics (including the general ones). The result is depicted in Figure 5.5.

As wee see, the best bundle of heuristics appeared to be *CRJ_pow2_gcd*, followed by *MP_pow2_gcd* and then by *RM_pow2_gcd*, all close to each other.

As one could expect, non-random heuristics showed better performance compared to the random ones. For the heuristic when the assignment was selected at random (RA) one of the reasons (among others) causing poor performance might be the fact that in randomly chosen assignments, the size of numerator and denominator is larger than in other heuristics for choosing the assignment.

For the heuristic when a conflict was chosen at random (RC) poor performance might be explained by the fact that the choice of resolving conflicts was not related to any particular aim.

A full set of charts with our experiments is included in Appendix D.

5.3.2.2 Real-Life Benchmarks

As mentioned earlier in this chapter, in our experiments we used three sets of Real-Life benchmarks. All of them were extracted from the real-life SMT benchmarks. The first set was used in the experiments in phase I and was generated by us using HRT, and consists of 305 problems with variable numbers ranging from 37 to 1416.

The second set was also generated by us using HRT, but compared to the first, the problems had a considerably higher difficulty level and consisted of 128 problems of several variable numbers in the range between 251 and 1067.

The third set of benchmarks was by Leonardo de Moura and consisted of 688 problems with the variable numbers between 147 and 163.

We discuss results of the experiments on quadruples first.

Equal and nearly equal performances. Almost all sets of heuristics showed that reduction of constraints by the GCD of their coefficients had almost no effect on all three sets of real-life benchmarks.

Along with this observation, on the first two sets of benchmarks the effect of exponentially increasing the length of artificially bounded half-bounded interval was slightly less efficient compared to constant interval size. For example, Figure 5.6. and Figure 5.7. depict the performance of quadruples for MP heuristics, for the first and the second sets of benchmarks respectively, plotted on ‘time versus number of solved problems’ graph. We see, that the difference is almost insignificant.

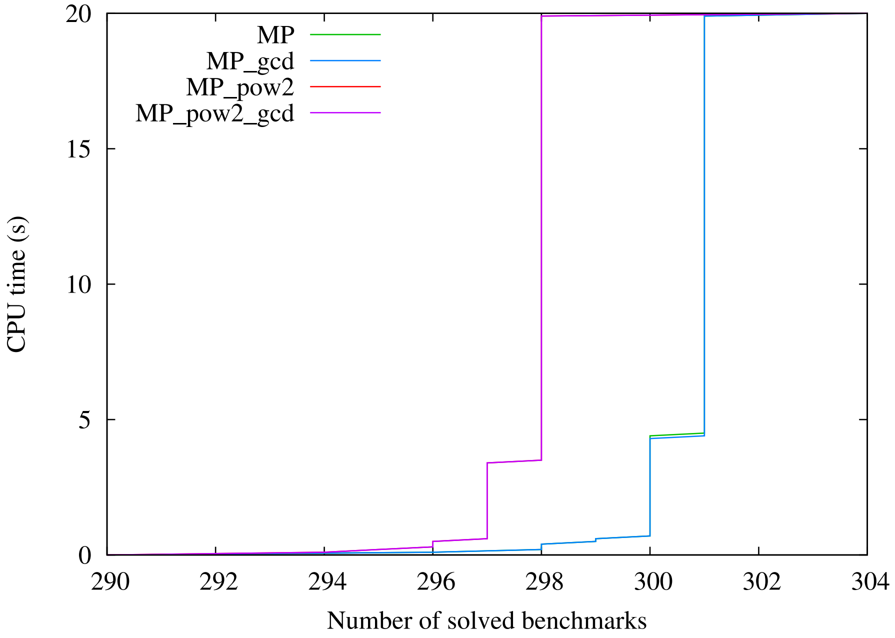


Figure 5.6: A quadruple of implementations of CRA with ‘MP’ heuristics for the first set of benchmarks, ‘time versus number of problems solved’

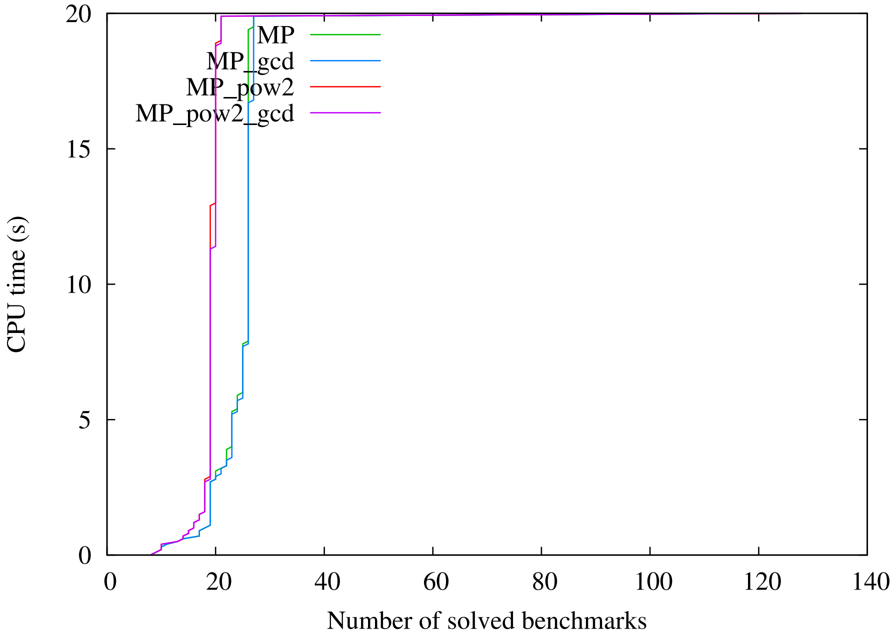


Figure 5.7: A quadruple of implementations of CRA with ‘MP’ heuristics for the second set of benchmarks, ‘time versus number of problems solved’

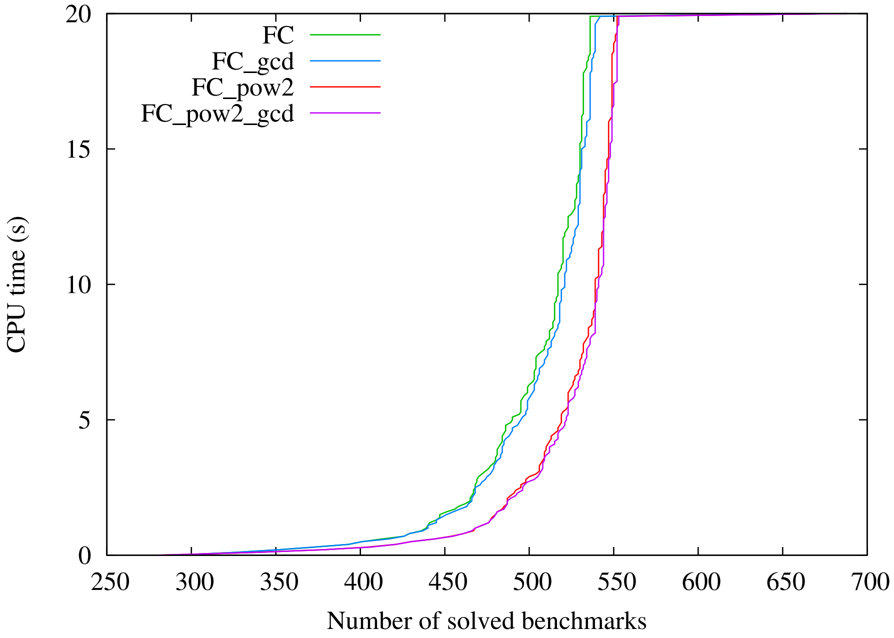


Figure 5.8: A quadruple of implementations of CRA with 'FC' heuristics for the third set of benchmarks, 'time versus number of problems solved'

However, on the third set of benchmarks, by Leonerdo de Moura, exponential increase of half-bounded intervals appeared to be rather efficient in some quadruples (leads to solving up to 15% more problems) and, similarly rather inefficient in others (leads to solving up to 15% less problems). For instance, Figure 5.8. plots performance of a quadruple for the set of major heuristics FC (take first conflict) on 'time versus number of solved problems' chart. As we see, doubling the size of substituting bounded intervals leads to better performance. While Figure 5.9. plotting a quadruple for MIN, shows that it is better to bound half-bounded intervals to a constant size.

The best performance in quadruples. Based on the above observations, both for the first and the second sets of the benchmarks the best performances were shown by the implementations where half-bounded intervals were bounded to a constant size and reduction by the GCD had almost no effect.

In the benchmarks by Leonardo de Moura, the best performance was achieved when reduction by the GCD was used. Regarding bounding half-bounded intervals, some quadruples had the best performance when these intervals were doubled, while others were faster when intervals were bounded to a constant size.

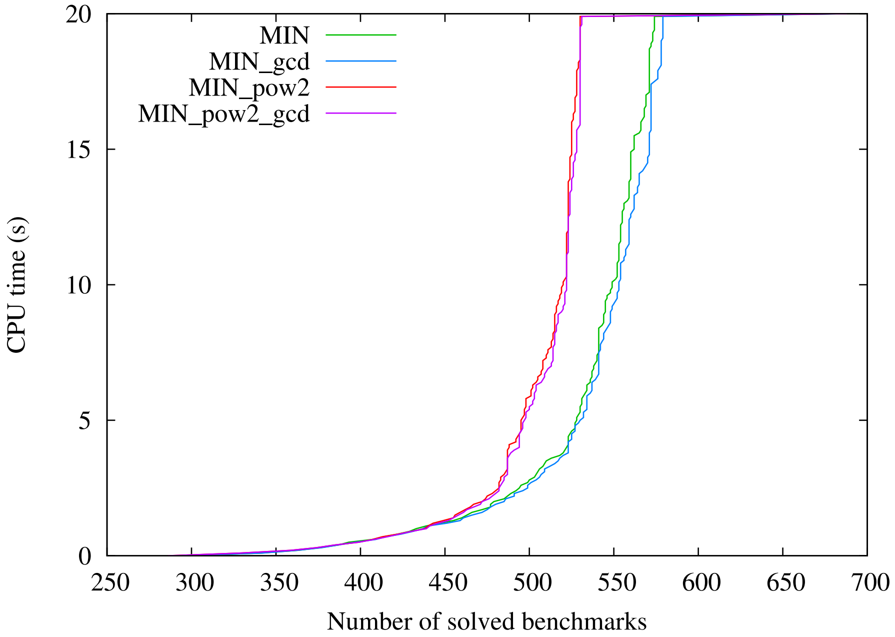


Figure 5.9: A quadruple of implementations of CRA with 'MIN' heuristics for the third set of benchmarks, 'time versus number of problems solved'

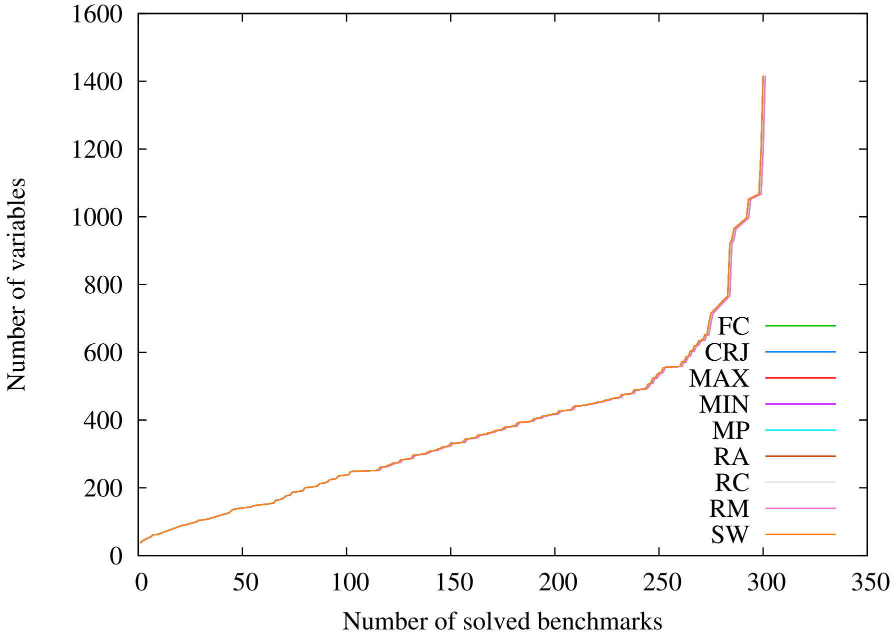


Figure 5.10: Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, 'number of variables versus number of solved problems'

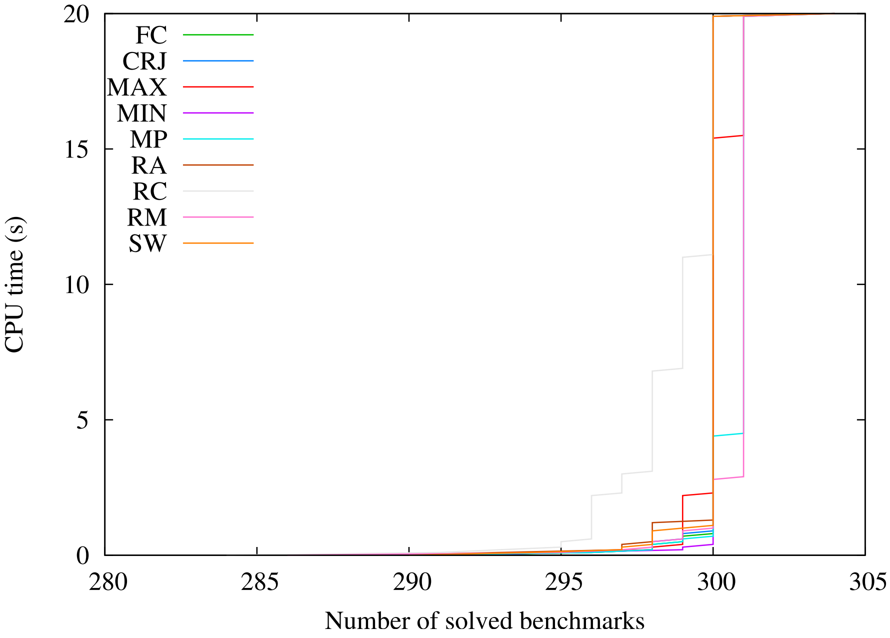


Figure 5.11: Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, 'time versus number of problems solved'

All quadruples. As in case of randomly generated benchmarks, in all three sets of real-life benchmarks, the worst performance was shown by RA and RC (randomly choosing assignment and conflict, respectively).

On the first set of benchmarks, all implementations solved almost the same number of problems with each number of variables (see Figure 5.10. with 'number of variablea versus number of problems' plotting). However, there are slight differences in times spent by the implementations on each problem. We can see from Figure 5.11. and Figure 5.12. that the top three performances were by MP, RM and MAX, all very similar to each other.

On the second set of benchmarks, all sets of major heuristics (apart from RA and RC) had very similar performance, with the only main difference in shifts on one problem (see Figure 5.13. with 'number of variables versus number of solved problems' plotting). These shifts were caused mainly only by solving different numbers of problems with 301 variables. The best performance was shown by *FC_gcd*. Then with an insignificant gap comes *MP_gcd*, followed by *CRJ_gcd*, *MAX_gcd* and *RM_gcd* with hardly distinguishable performances.

On the third set of benchmarks, as in case of the first set, all sets of heuristics (apart from RA and RC) solved the same number of problems for each number of variables, see Figure 5.14.

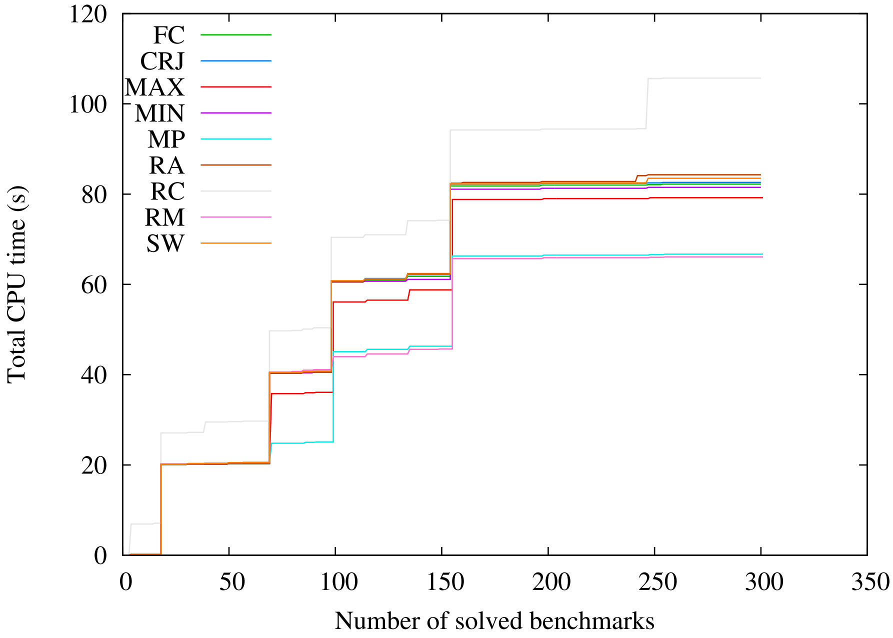


Figure 5.12: Implementations of CRA with the best performance of all major heuristics for the first set of benchmarks, ‘total run-time versus number of solved problems’

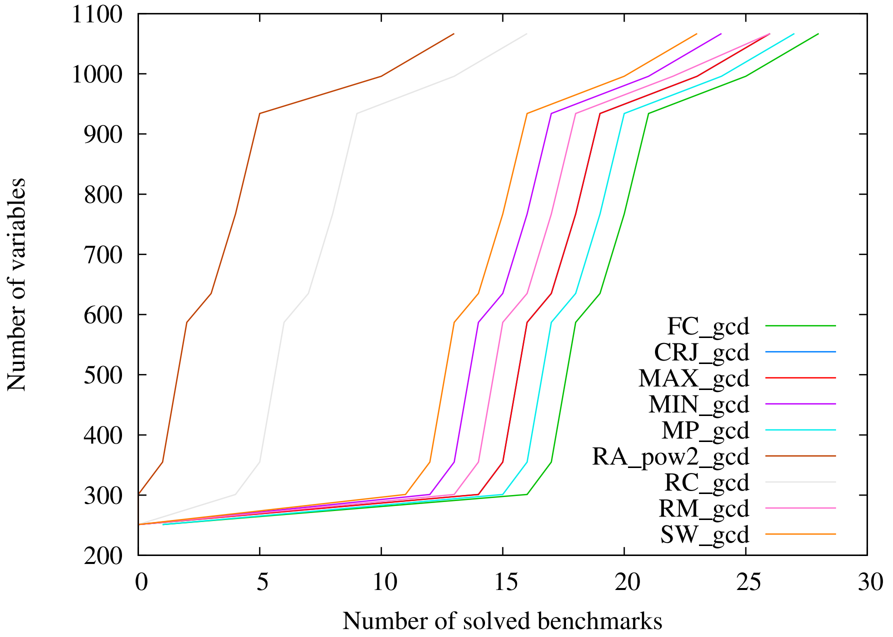


Figure 5.13: Implementations of CRA with the best performance of all major heuristics for the second set of benchmarks, ‘number of variables versus number of solved problems’

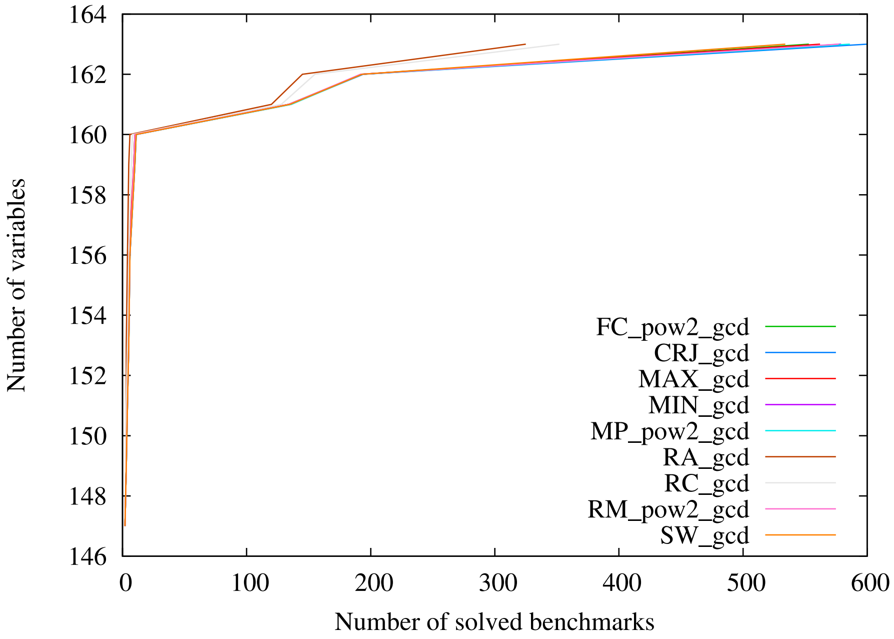


Figure 5.14: Implementations of CRA with the best performance of all major heuristics for the third set of benchmarks, 'number of variables versus number of solved problems'

The difference was in times spent on each problem, which is visible on Figure 5.15. plotting total time against the number of benchmarks solved. We see, that the top performances were by *MP_pow2_gcd*, *CRJ_gcd* and *RM_pow2_gcd*, close to the performance of *MIN_gcd*.

Summary of Phase II experiments. To conclude, as it was expected, the performance of various bundles of heuristics of the CRA algorithm was different on randomly generated benchmarks and real-life benchmarks. We observed, that our non-random heuristics considerably outperformed the random ones (RA and RC).

For randomly generated problems the top three performers were major heuristics: CRJ, MP and RM. However, all three were quite close to each other. For real-life benchmarks, on the first set of benchmarks the best major heuristics were MP, RM and MAX; on the second set of benchmarks MP, CRJ, and two sets of heuristics MAX and RM with similar performance; on the third set, FC, MP, and three sets of heuristics with similar performance CRJ, RM and MIN. As we can see, three of the heuristics, MP, CRJ and RM were present almost always in the top three performances in both randomly generated and real-life problems.

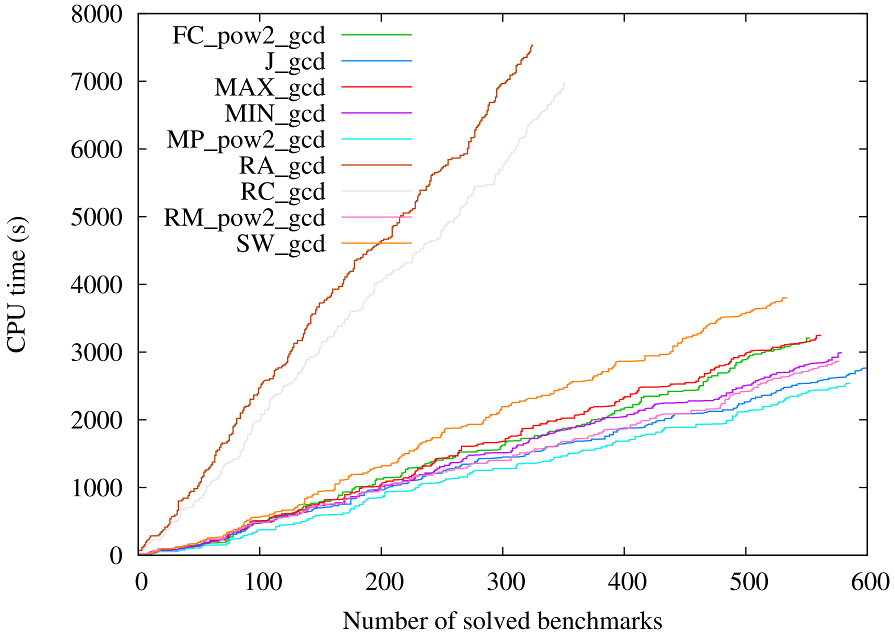


Figure 5.15: Implementations of CRA with the best performance of all major heuristics for the third set of benchmarks, ‘total run-time versus number of solved problems’

Regarding the general heuristics (for dealing with half-bounded intervals and for reducing by the GCD) the results of experiments were, again, different for randomly generated and real-life benchmarks. On randomly generated benchmarks there was not much difference in performance for these heuristics. In combination with some major sets of heuristics, reduction by the GCD was very effective, in others it did not show much improvement. The same went for doubling the size of artificially bounded intervals. However, the use of reduction by the GCD and exponential increase of artificial bound of half-bounded intervals were two heuristics leading to better performance.

On real-life benchmarks, the first two benchmarks showed better performance when intervals were kept at constant length. Regarding reduction by the GCD, it did not lead to any great difference in performance. For the third set of benchmarks some sets of major heuristics showed better performance in combination with general heuristics: reduction by the GCD and constant length of half-bounded intervals; while others were more effective in combination with: reduction by the GCD and doubling the size of artificially bounded intervals.

Based on these observation, for the experiments in the third phase of the implementation, we selected the CRA with the major set of heuristics MP.

688 problems (unsat)				
	faster	**same**	**timeout (20s)**	**av. time**
CRA + simple preproc	395	263	88	0.8
CRA	30	263	476	0.005

Table 5.5: CRA vs CRA with simple preprocessing on the third set of real-life benchmarks

5.3.3 Phase III – Simple Preprocessing with Highly-Tuned Heuristics

In the third phase, we ran experiments on the implementation of the CRA algorithm with the major heuristics MP enhanced with some simple preprocessing. To recall MP stands for the set of heuristics: (i) choose a conflict with the maximal overlap, and (ii) choose the closest binary assignment – an aproximate mid point of the interval.

As for general heuristics, we used the best heuristics from the Phase II experiments: reduce constraints by the GCD and exponentially increase intervals.

For setting the order of variables we used the strategy that is based on the analysis of the input data. Both, the order selection strategy used and the implemented preprocessing are described in Section 4.2.3.

We compared the implementation of the CRA algorithm enhanced with preprocessing with (i) its predecessor implementation from the Phase II (with the same bundle of heuristics), and also (ii) three state-of-the-art SMT solvers: Barcelogic, CVC3 and Z3.

In general the performance of solvers differ considerably on randomly generated and real-life benchmarks. It is more important to have good performance on real-life benchmarks, rather than on randomly generated ones. For this reason when comparing our optimised implementation with the state-of-the-art SMT solvers we decided to present results with experiments on real-life benchmarks. We present results of experiments on the real-life benchmarks.

CRA with simple preprocessing vs its predecessor. First, we compare the implementation of CRA with simple preprocessing with its predecessor. Table 5.5 presents results of such a comparison on the third set of real-life benchmarks.

As we see, the number of timeouts has decreased more than 5 times when adding the preprocessing. Also, it improved the performance on more than half of

CRA	Better	Same	Worse
Barcelogic	161	151	120
CVC3	150	278	3
Z3	10	370	52

Table 5.6: CRA with simple preprocessing vs Barcelogic, CVC3, Z3 on the first and the second sets of real-life benchmarks.

the problems. The CRA algorithm is sensitive to the implemented preprocessing and to the variable ordering strategy chosen.

CRA with simple preprocessing vs state-of-the-art SMT solvers. As for comparison with the linear arithmetic solvers incorporated in Barcelogic, CVC3 and Z3 the experiments showed the following.

In Table 5.6 we present results of experiments with our real-life benchmarks, extracted with the HRT tool. CRA with simple preprocessing is faster than CVC3 on about one third of the problems and has the same performance for almost all other problems. Compared to Barcelogic, CRA performs better again on about one third of the problems, has the same performance on the second third, and Barcelogic is faster than CRA on the other third of the problems. As for Z3, on both sets of real-life problems CRA showed a competitive performance – on about 93% of the problems it showed similar performance, and outperformed it on a number of instances.

We also present results of comparison of CRA with simple preprocessing with Z3 on the third set of benchmarks.

In the third set of real-life benchmarks Z3 solved each of the problems in 0 seconds. Our implementation of the CRA algorithm with preprocessing showed the same performance on a little more than a half of the problems. On the rest of them Z3 was superior. To illustrate the difference in performance on problems where Z3 was faster, we plotted the data from such problems on 'time spent per each problem versus number of problems' graph, see Figure 5.16.

As we see, in the majority of problems where Z3 was superior, the difference between the times spent on each problem was less than 5 seconds. More precisely, about 65% of the problems where Z3 was superior were solved by CRA with simple preprocessing in less than 5 seconds.

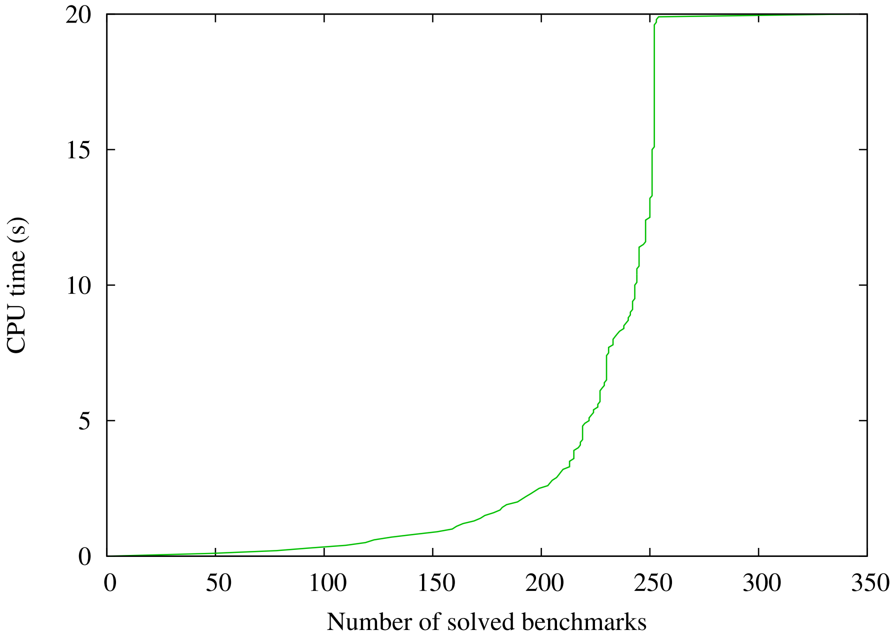


Figure 5.16: Benchmarks where CRA with simple preprocessing and Z3 showed different performances, 'time versus number of problems solved'

Summary of Phase III experiments. To conclude, the performance of the CRA algorithm enhanced with simple preprocessing and optimised with fine-tuned heuristics improved considerably over its predecessor without preprocessing. Compared to the linear arithmetic solvers incorporated in the state-of-the-art SMT solvers, it was faster than the Fourier-Motzkin based linear arithmetic solver of CVC3 and was competitive with the simplex based linear arithmetic solvers incorporated in Barcelogic and Z3. On certain problems CRA was faster than the three solvers, and on a significant part of the problems it showed the same performance.

Conclusions. The experiments showed that choosing different parameters has a significant impact on the performance of the solver. Also, depending on the nature of the problem, different heuristics may appear preferable. At the same time, one can outline the most preferable strategies in general: in selecting a conflicting pair the reasonable choice seems to be the maximal overlap strategy, or the relaxation approach, as for selecting the assignment one could recommend the so called mid point strategy (based on binary approximation). Using boundary assignments may also appear successful if additional information on the system is taken into account. For instance, the ratio of the number of positive and negative coefficients at the highest variable on the current level.

Among the general heuristics, choosing the combination of the '*gcd*' with '*pow*2' in most cases appears to be the best choice.

The algorithm appeared to be sensitive to the implemented preprocessing and order of variables used. Choosing an appropriate ordering on variables may also yield a significant benefit. This problem needs further detailed study.

On the whole, considering that we used some of the best SMT solvers for comparison, conflict resolution showed itself to be potentially competitive with the simplex method, and definitely outperforms the Fourier-Motzkin method with modifications.

Chapter 6

Conclusions

In this thesis, we presented a new method for solving systems of linear constraints over the rational numbers. We call the method the conflict resolution method. As a part of the research presented we implemented a solver based on this method and evaluated its performance in a range of experiments. In this chapter we summarise the research conducted, pointing out its significance, and outline future research directions.

6.1 Thesis Achievements

We presented a new algorithm for solving systems of linear constraints, called conflict resolution. The method successively refines an initial assignment with the help of newly derived constraints, until either the assignment becomes a solution of the system or the inconsistency of the initial system is proved. We have shown that this method is correct and terminating. The conflict resolution method has a number of attractive properties such as blocking of redundant inferences. We implemented our method and evaluated its performance in a series of experiments, compared it with various existing methods, and studied the problem of improving its efficiency.

A historical survey of the problem and existing methods were introduced in Section 2.3. Some of these methods, the most important and relevant to our research, were briefly discussed in Section 2.4.

The main results of the dissertation were presented in Chapter 3, Chapter 4 and Chapter 5. In Chapter 3 we introduced our method and proved its correctness and termination. We presented the properties of the algorithm, supplied with the necessary proofs, and discussed its extensions, worst case complexity, and application to SMT solving.

Then, we described the design of our implementation system (in Chapter 4) outlining in detail all intermediate phases of the implementation. Along with this, we introduced the heuristics that we developed for improving the efficiency of our solver.

We devoted Chapter 5 to the experimental part of our research. We described the full set of benchmarks we used. The results of the experiments evaluated on these benchmarks were presented for each of the phases of implementation. We concluded the chapter with a summary of the experimental results.

6.2 Significance of the Research

The subject of this dissertation is a well-known classical problem in mathematics and computer science. It concerns deciding satisfiability of systems of linear constraints over the rationals. The problem had been studied since the 19th century, when Fourier proposed the first method for solving it, now known as the Fourier-Motzkin elimination method. Because of a number of drawbacks, his method turned out to have poor performance in practice.

Since then, many attempts have been made to find more effective ways to solve this problem. As a result a number of modifications and alternative methods have been proposed but their number is still very small. Thus, there are the simplex and interior points methods developed in the 20th century. Among them the best performance has been shown by simplex. However, the worst case complexity of simplex is exponential, even though it is very fast in practice.

It is essential to mention, that despite the fact that some methods (in this case the Fourier-Motzkin method, see Section 2.4.1) have worse performance in practice, there are complex applications were they might be of a clear superiority than others. This, obviously, is entailed by the structural characteristics of the applications, and one particular method can be better suited to an application than others. Consequently, scarcity of options available for solving a particular problem may be restricting on some occasions.

Significance. The significance of the presented research is thus obvious. It presents a new method for this problem, that not only showed itself to be very competitive with (sometimes even outperforming) the existing methods, but also has a number of substantial properties.

To conclude:

- We presented a new algorithm for solving systems of linear constraints – CRA.
- The method works well for both satisfiable and unsatisfiable problems.
- One of the important properties of the CRA algorithm is that it never performs redundant inferences, as defined in Chapter 3. Our notion of redundancy seems to be orthogonal to others developed in similar situations for the Fourier-Motzkin method (see Section 2.4.2). In particular our redundancy criterion is based on constraint ordering and semantic entailment from smaller constraints.
- Our implementation of the CRA is orders of magnitude better than the Fourier-Motzkin method and Chernikov algorithm.
- CRA shows good potential when compared to the simplex method. On a whole range of problems it showed performance similar to state-of-the-art solvers. On certain problems it even outperforms some of them.
- Our method can be easily made incremental and can easily generate explanations for unsatisfiability (which is important for SMT).

6.3 Future Work

As future work, we have several research perspectives that can be grouped into two main directions – SMT solving and Linear Programming itself.

SMT solving.

1. *Integration into SMT.* A future direction, that we have been constantly outlining throughout the thesis, concerns integration of CRA into SMT. The conflict resolution method has been discovered while working on SMT problems with theories of linear real and integer arithmetics. It has a number of attractive properties including those necessary for SMT integration. We

aim to plug the conflict resolution solver in existing state-of-the-art SMT solvers and also study possible advantages of the conflict resolution method (drawn by its specific characteristics) for SMT solving purposes, this mainly refers to using the conflict resolution method for model searching directly in the space of numeric structure of SMT problems with linear arithmetic.

2. *Combination with reasoning methods in other theories.* Research on combining reasoning methods for multiple theories is of high interest and significance for SMT solving. Our future work towards the integration in SMT solving also considers perspectives of combining conflict resolution methods with reasoning methods in other theories.

3. *Research towards dynamic change of variable orderings.* Currently the most efficient approach for SMT solving is DPLL(T). This approach uses dynamic change of the order on variables. Therefore we consider modifications of the conflict resolution method with dynamic orderings on variables as one more future research direction towards SMT solving.

Research directions related to Linear Programming involve further modifications and extensions of the conflict resolution method.

Linear programming.

1. *Modification for integer arithmetic.* Extension of the CRA algorithm towards the integer and mixed (integer and rational) problems may also be considered.

2. *Modification for solving systems of Pseudo-Boolean linear constraints.* We may also consider extension of the CRA algorithm for solving systems of Pseudo-Boolean linear constraints.

3. *Modification for non-linear arithmetic.* Generally, theories involved in SMT problems are decidable theories. However, due to the increasing need in dealing with non-linear arithmetic over the real numbers involving special functions a need of using non-linear arithmetic constraints in SMT solving became natural. Introduction of such constraints in general leads to undecidable theories, therefore extension of SMT to undecidable theories became essential. However, there is a very little success in this direction.

As a further research direction we may suggest modification of CRA for non-linear arithmetic, considering that development of a non-linear version of the conflict resolution method can be better suited for some particular types of non-linear constraints.

4. *Extension of the implementation.* The current implementation works with strict and non-strict inequalities, and equalities. The implementation may be extended for constraints with $\{\neq\}$. Extension for disequalities makes it natural to consider clauses containing disjunctions of inequalities and multi-interval domains targeting variables assignment.

5. *Optimisation of the implementation.* One obvious future topic is to optimise the implementation of our solver. This, first of all, includes adding a wrapper for rational numbers that is utilised in almost all state-of-the-art solvers today in order to improve the performance of the solvers.

6. *Further development of heuristics.* New heuristics for improving the performance of our algorithm may keep being under development constantly. For now, we have certain ideas related to:

   - Changing the order of the variables at run-time (discussed in Section 4.2.3.1);
   - Developing methods for avoiding unnecessary re-evaluation of constraints;
   - Investigating whether it is possible to combine our notion of redundancy with restrictions used by other methods.

Bibliography

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[2] C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.

[3] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[4] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV '07*, volume 4590 of *Lecture Notec in Computer Science*, pages 298–302. Springer Verlag, 2007. Berlin, Germany.

[5] V. Chandru. Variable elimination in linear constraints. *The Computer Journal*, 36(5):463–472, 1993.

[6] S. N. Chernikov. *Linejnye Neravenstva.* Nauka, Moscow, 1968. (In Russian).

[7] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, 1983.

[8] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tj. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347, Wiley, New York, 1951.

[9] G. B. Dantzig. *Linear Programming and Extensions.* Princeton, New Jersey: Princeton University Press, 1963.

[10] L. de Moura. Invited tutorial: Applications of SMT solvers in software verification. VSTTE'08, Toronto, Canada 2008.

[11] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[12] L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In A. Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 2002.

[13] L. L. Dines. Systems of linear inequalities. *Annals of Mathematics*, 2(20):191–199, 1918 9.

[14] R.J. Duffin. On Fourier's analyse of linear inequality systems. *Mathematical Programming Study*, 1:71–95, 1974.

[15] B. C. Eaves and U. G. Rothblum. Dines-Fourier-Motzkin quantifier elimination and an application of corresponding transfer principles over ordered fields.

[16] J. B. J. Fourier. Analyse des travaux de l'Académie Royale des Sciences. *pendant l'année 1823, Partie Mathématique, Histoire de l'Acadé mie Royale des. Sciences de l'Institut de France*, 6(xxix-xli), [1823](1826).

[17] J. B. J. Fourier. Analyse des travaux de l'Académie Royale des Sciences. *pendant l'année 1824, Partie Mathématique, Histoire de l'Académie Royale des. Sciences de l'Institut de France*, 7(xlvii-lv), [1824](1827). English Translation (partially) in: D.A. Kohler, Translation of a report by Fourier on his work on linear inequalities, Opsearch, 10, 1973, pages 38-42.

[18] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.

[19] S. I. Gass. *Linear Programming.* New York: McGraw-Hill, 1975.

[20] S. Ghilardi and S. Ranise. MCMT: A model checker modulo theories. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR*, volume 6173 of *Lecture Notes in Computer Science*, pages 22–29. Springer, 2010.

[21] J. I. Goffin. The relaxation method for solving systems of linear inequalities. *Mathematics of Operations Research*, 5:388–414, 1980.

[22] J. L. Imbert and P. Van Hentenryck. A note on redundant linear constraints. Technical Report CS-92-11, CS Department, Brown University, 1992.

[23] J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Projecting CLP($\mathcal{R}$) constraints. *New Generation Computing*, 11, 1993.

[24] L. V. Kantorovich. Mathematical methods in the organisation and planning of production. *Publication House of the Leningrad State University, Leningrad*, 1939. English translation: Management Science, vol. 6, 1960, 366422.

[25] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of Sixteenth Annual ACM Symposium on Theory of Computing, Washington*, pages 302–311, The association of Computing Machinery, New York, 1984.

[26] L. G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979. English translation: Soviet Mathematics Doklady 20 (1979) 191-194.

[27] D.A. Kohler. *Projection of Convex Polyhedral Sets.* PhD thesis, University of California, Barkeley, 1967.

[28] Tj. C. Koopmans. Optimum utilization of the transportation system. In D. H. Leavens, editor, *Proceedings of the International Statistical Conferences*, volume V, pages 136–146, The Econometric Society Meeting, Washington, D.C., 1948.

[29] Tj. C. Koopmans. A note about Kantorovich's paper, M̈athematical methods of organizing and planning production". *Management Science*, 6:363–365, 1959-60.

[30] K. Korovin, N. Tsiskaridze, and A. Voronkov. Conflict Resolution. In I. P. Gent, editor, *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2009.

[31] K. Korovin and A. Voronkov. GoRRiLA and Hard Reality. In *PSI*, Lecture Notes in Computer Science. Springer, 2011. To appear.

[32] S. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '08, pages 171–182, New York, NY, USA, 2008. ACM.

[33] A. M. Lukatskii and D. V. Shapot. A constructive algorithm for folding large-scale systems of linear inequalities. *Computational Mathematics and Mathematical Physics*, 48(7):1100–1112, 2008.

[34] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[35] D. Monniaux. A quantifier elimination algorithm for linear real arithmetic. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR '08, pages 243–257, Berlin, Heidelberg, 2008. Springer Verlag.

[36] D. Monniaux. On using floating-point computations to help an exact linear arithmetic decision procedure. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 570–583. Springer, 2009.

[37] D. Monniaux. Quantifier elimination by lazy model enumeration. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2010.

[38] T. S. Motzkin. Beiträge zur theorie der linearen ungleichungen. Inaugural Dissertation Basel, Azriel, Jerusalem, 1936. English translation: Contributions to the theory of linear inequalities, RAND Corporation Translation 22, Santa Monica, California.

[39] T. S. Motzkin and I. J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:393–404, 1954.

[40] R. Nieuwenhuis and A. Oliveras. Decision procedures for SAT, SAT modulo theories and beyond. The Barcelogic Tools. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.

[41] J. Nocedal and S. Wright. *Numerical Optimization.* New York, NY: Springer, 1999.

[42] V. Riley and S. I. Gass. *Bibliography of Linear Programming.* Baltimore: Johns Hopkins Press, 1958.

[43] N. Bjørner, L. De Moura, and N. Tillmann. Satisfiability Modulo Bit-precise Theories for Program Exploration. *Fifth International Workshop on Constraints in Formal Verification*, 2008.

[44] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.

[45] N. Z. Shor. Convergence rate of the gradient descent method with dilitation of the space (in Russian). *Kibernetika (Kiev)*, 2:80–85, 1970. English translation: Cybernetics 6 (1970) 102-108.

[46] N. Z. Shor. Utilization of the operation of space dilitation in the minimization of convex functions (in Russian). *Kibernetika (Kiev)*, 1:6–12, 1970. English translation: Cybernetics 6 (1970) 7-15.

[47] N. Z. Shor. Cut-off method with space extension in convex programming problems (in Russian). *Kibernetika (Kiev)*, 1:94–95, 1977. English translation: *Cybernetics* 13 (1977) 94-96.

[48] S. Srivastava, S. Gulwani, and J. S. Foster. Vs3: SMT solvers for program verification. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 702–708. Springer, 2009.

[49] A. Stump, C. Barrett, and D. Dill. Cvc: A cooperating validity checker. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2002.

[50] J. von Neumann. Discussion of a maximum problem. Unpublished working paper, Institute for Advanced Study, Princeton, N.J.,1947. Reprinted in: John von Neumann, Collected Works, Vol. VI (A. H. Taub, ed.), Pergamon Press, Oxford, 1963, pages 89-95.

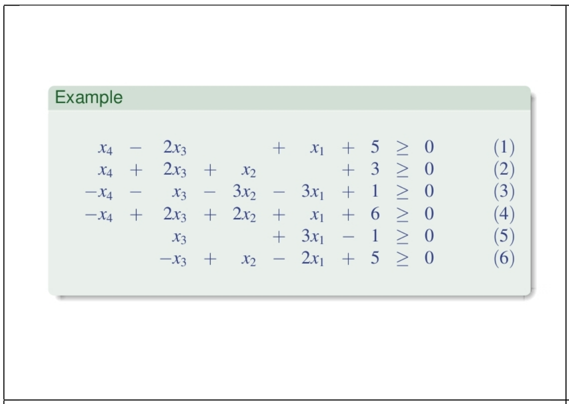
[51] D. B. Yudin and A. S. Nemirovskii. Evaluation of the informational complexity of mathematical programming problems (in Russian). *Ékonomika i*

Matematicheskie Metody, 12:128–142, 1976. English translation: Matekon 13 (2) (1976 7) 3-25.

[52] D. B. Yudin and A. S. Nemirovskii. Informational complexity and efficient methods for solution of convex extremal problems (in Russian). *Ékonomika i Matematicheskie Metody*, 12:357–369, 1976. English translation: Matekon 13 (3) (1977) 25-45.

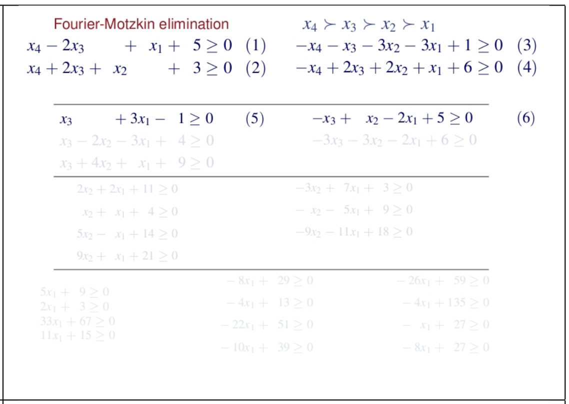
Appendices

Appendix A



Example

$$\begin{array}{rcrcrcrcrcl}
x_4 & - & 2x_3 & & & + & x_1 & + & 5 & \geq & 0 \quad (1)\\
x_4 & + & 2x_3 & + & x_2 & & & + & 3 & \geq & 0 \quad (2)\\
-x_4 & - & x_3 & - & 3x_2 & - & 3x_1 & + & 1 & \geq & 0 \quad (3)\\
-x_4 & + & 2x_3 & + & 2x_2 & + & x_1 & + & 6 & \geq & 0 \quad (4)\\
 & & x_3 & & & + & 3x_1 & - & 1 & \geq & 0 \quad (5)\\
 & & -x_3 & + & x_2 & - & 2x_1 & + & 5 & \geq & 0 \quad (6)
\end{array}$$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

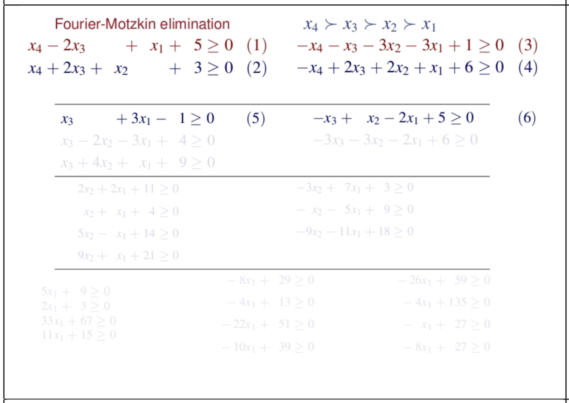
$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$11x_1 + 15 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

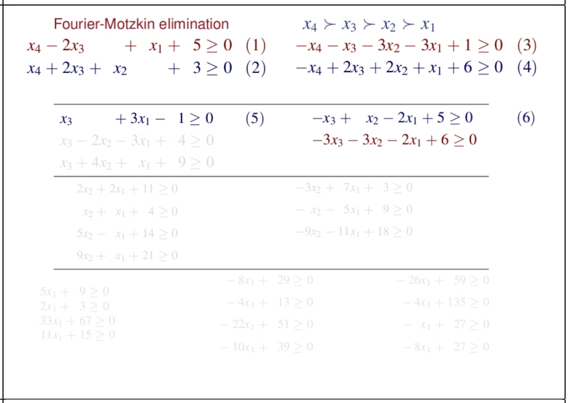
$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$11x_1 + 15 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

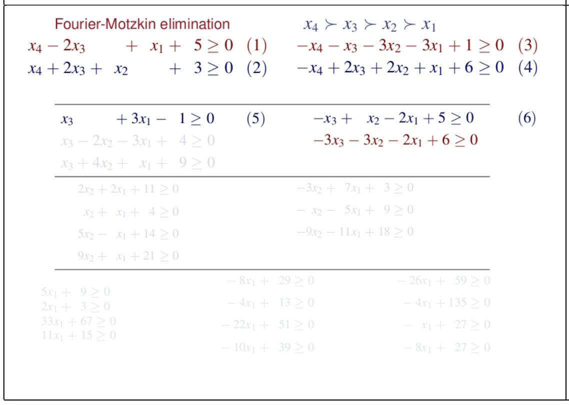
$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$11x_1 + 15 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

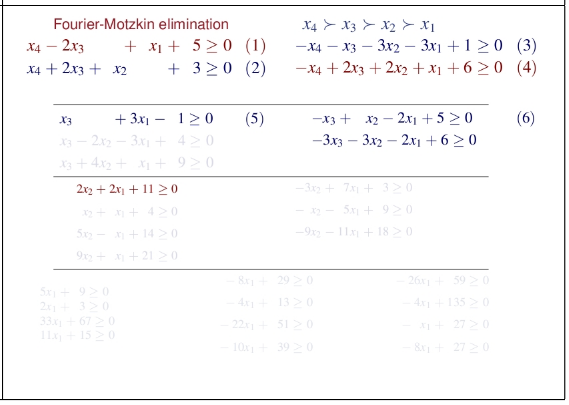
$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$11x_1 + 15 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

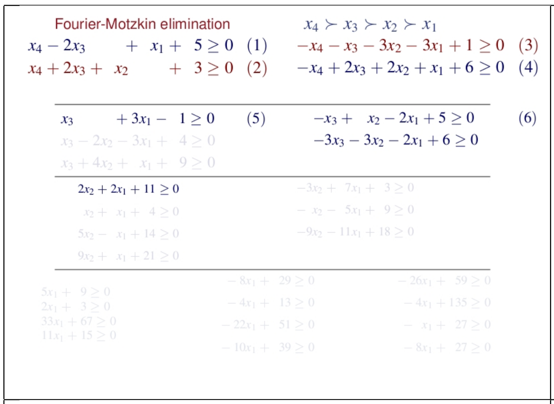
$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$11x_1 + 15 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



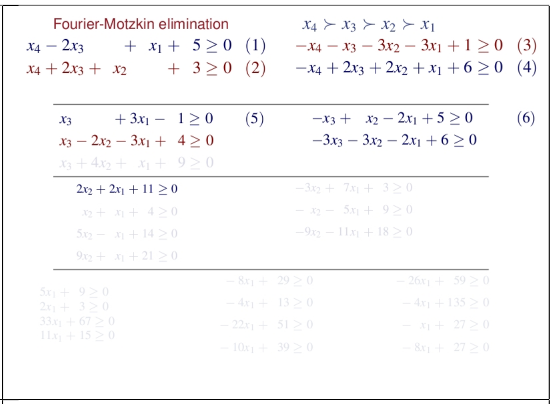
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



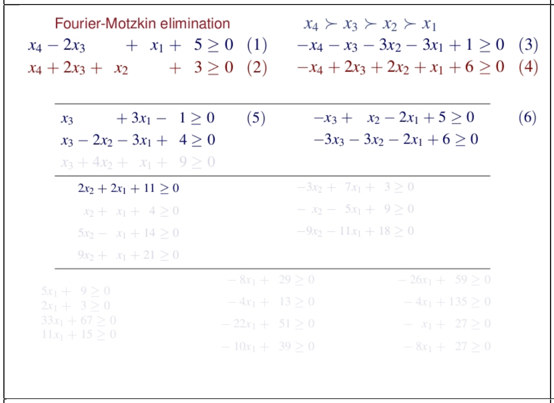
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



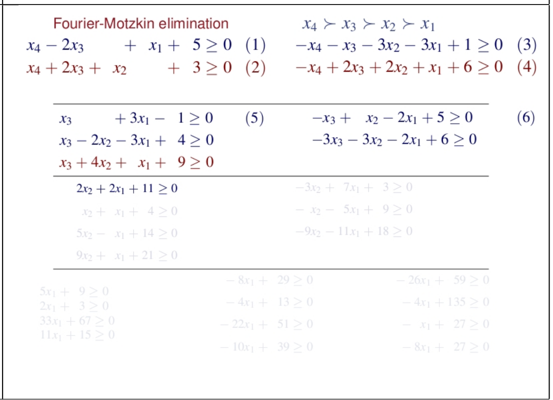
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



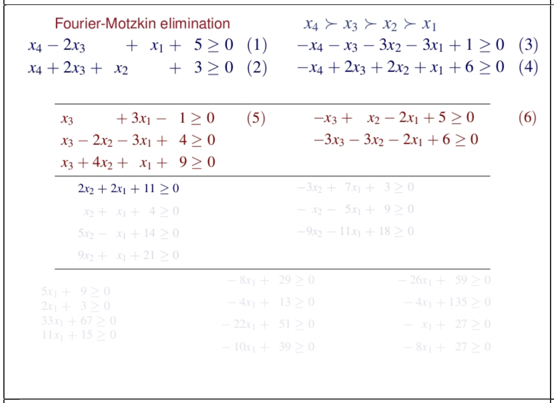
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



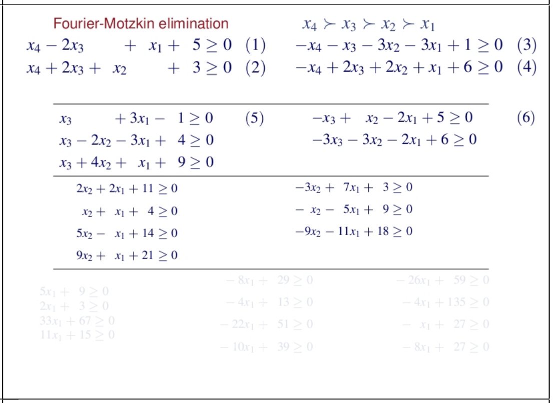
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



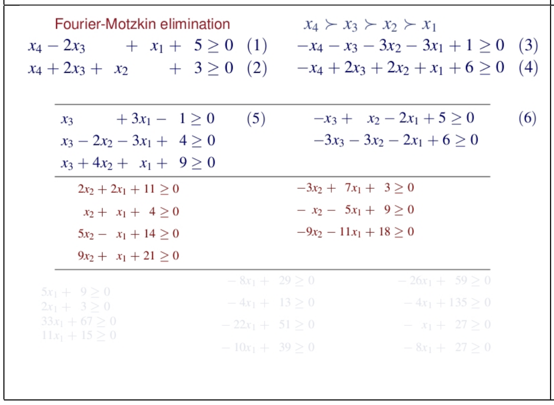
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



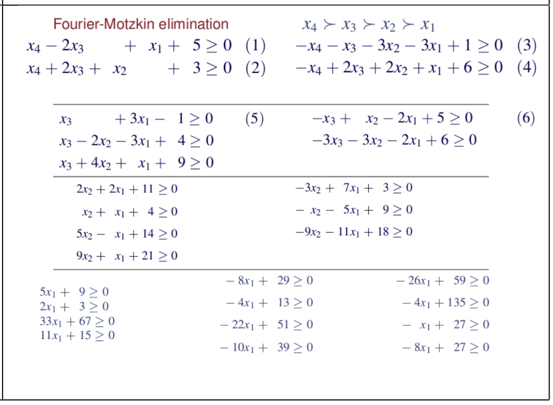
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



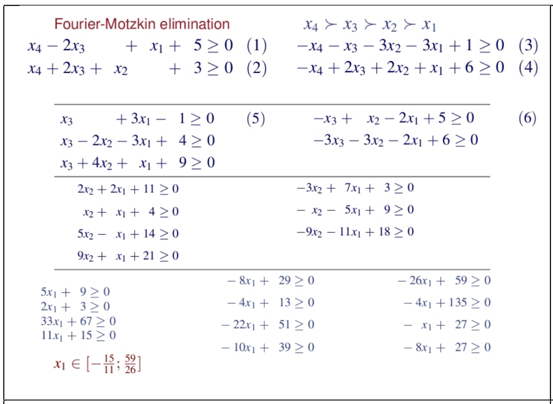
Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)
$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)
$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$
$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$
$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$
$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$
$9x_2 + x_1 + 21 \geq 0$

$5x_1 + 9 \geq 0$ $-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$
$33x_1 + 67 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$
$11x_1 + 15 \geq 0$ $-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

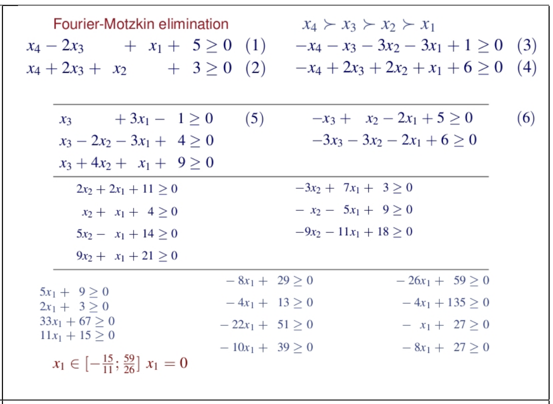
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

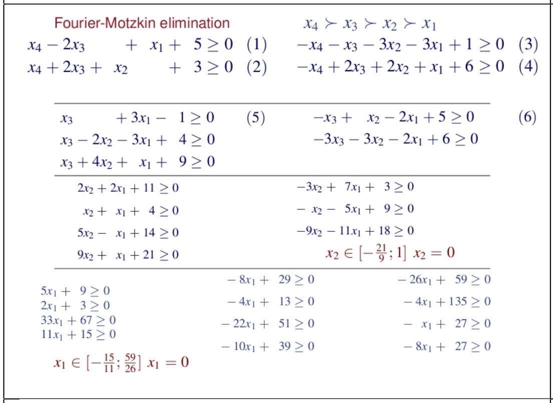
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$ $x_1 = 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$ $x_2 \in [-\frac{21}{9}; 1]$ $x_2 = 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

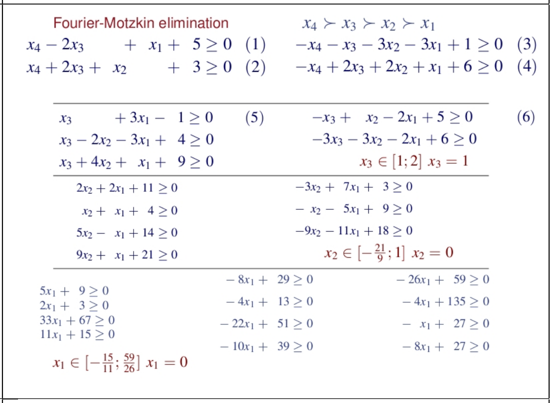
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$ $x_1 = 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$ $x_3 \in [1; 2]$ $x_3 = 1$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$ $x_2 \in [-\frac{21}{9}; 1]$ $x_2 = 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

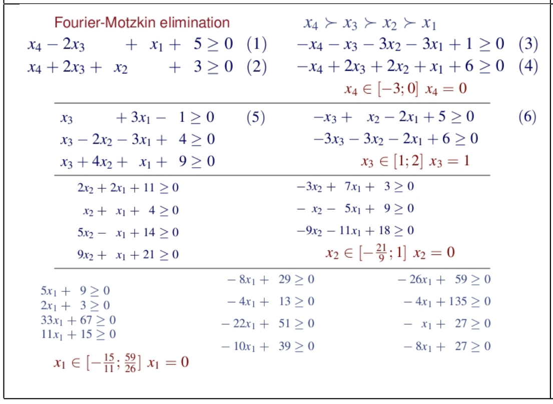
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$ $x_1 = 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_4 \in [-3; 0]$ $x_4 = 0$

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$ $x_3 \in [1; 2]$ $x_3 = 1$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$ $x_2 \in [-\frac{21}{9}; 1]$ $x_2 = 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

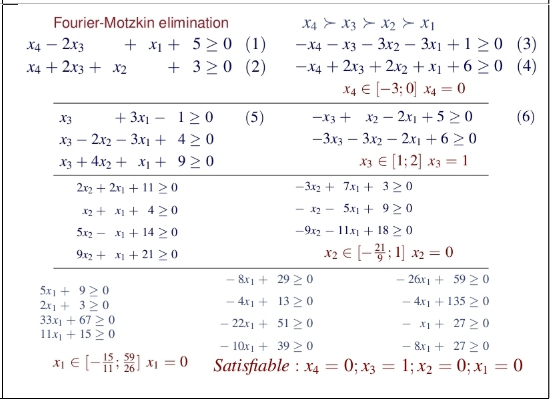
$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$ $x_1 = 0$



Fourier-Motzkin elimination $x_4 \succ x_3 \succ x_2 \succ x_1$

$x_4 - 2x_3 + x_1 + 5 \geq 0$ (1) $-x_4 - x_3 - 3x_2 - 3x_1 + 1 \geq 0$ (3)

$x_4 + 2x_3 + x_2 + 3 \geq 0$ (2) $-x_4 + 2x_3 + 2x_2 + x_1 + 6 \geq 0$ (4)

$x_4 \in [-3; 0]$ $x_4 = 0$

$x_3 + 3x_1 - 1 \geq 0$ (5) $-x_3 + x_2 - 2x_1 + 5 \geq 0$ (6)

$x_3 - 2x_2 - 3x_1 + 4 \geq 0$ $-3x_3 - 3x_2 - 2x_1 + 6 \geq 0$

$x_3 + 4x_2 + x_1 + 9 \geq 0$ $x_3 \in [1; 2]$ $x_3 = 1$

$2x_2 + 2x_1 + 11 \geq 0$ $-3x_2 + 7x_1 + 3 \geq 0$

$x_2 + x_1 + 4 \geq 0$ $-x_2 - 5x_1 + 9 \geq 0$

$5x_2 - x_1 + 14 \geq 0$ $-9x_2 - 11x_1 + 18 \geq 0$

$9x_2 + x_1 + 21 \geq 0$ $x_2 \in [-\frac{21}{9}; 1]$ $x_2 = 0$

$-8x_1 + 29 \geq 0$ $-26x_1 + 59 \geq 0$

$5x_1 + 9 \geq 0$

$2x_1 + 3 \geq 0$ $-4x_1 + 13 \geq 0$ $-4x_1 + 135 \geq 0$

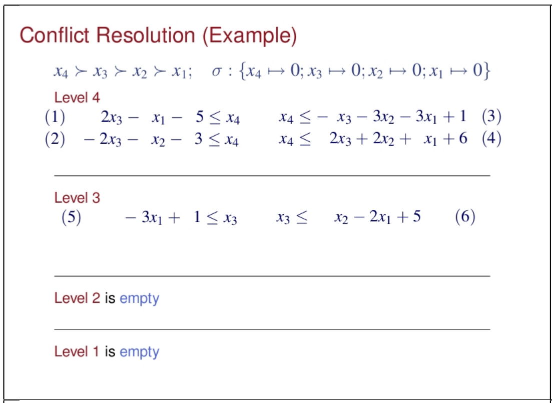
$33x_1 + 67 \geq 0$

$11x_1 + 15 \geq 0$ $-22x_1 + 51 \geq 0$ $-x_1 + 27 \geq 0$

$-10x_1 + 39 \geq 0$ $-8x_1 + 27 \geq 0$

$x_1 \in [-\frac{15}{11}; \frac{59}{26}]$ $x_1 = 0$ *Satisfiable* : $x_4 = 0; x_3 = 1; x_2 = 0; x_1 = 0$

Appendix B



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 0; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

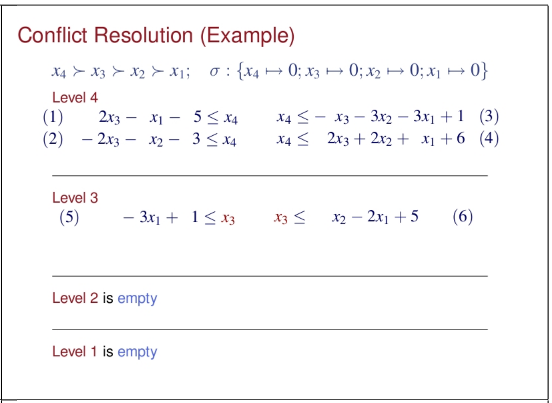
(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

Level 2 is empty

Level 1 is empty



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 0; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

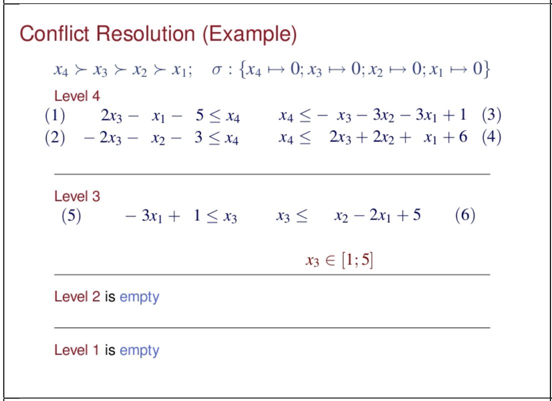
(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

Level 2 is empty

Level 1 is empty



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 0; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

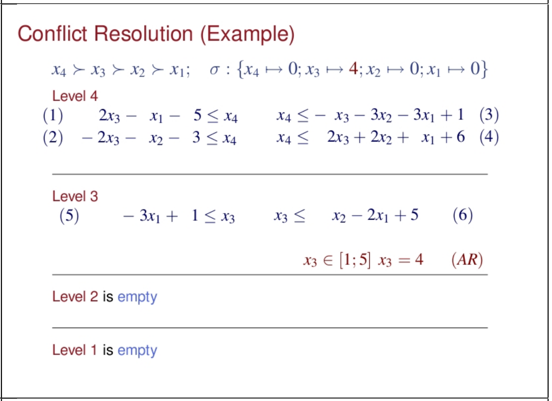
Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

$x_3 \in [1;5]$

Level 2 is empty

Level 1 is empty



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

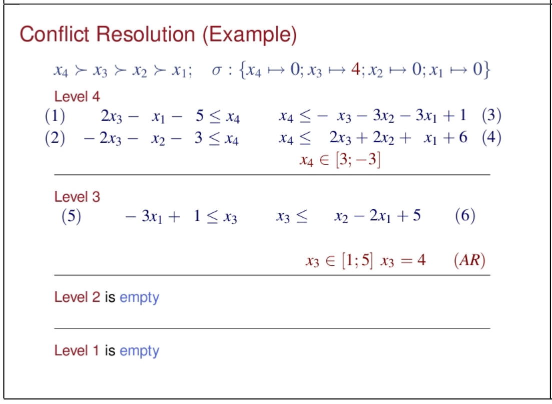
Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

$x_3 \in [1;5]$ $x_3 = 4$ $\quad (AR)$

Level 2 is empty

Level 1 is empty



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [3;-3]$

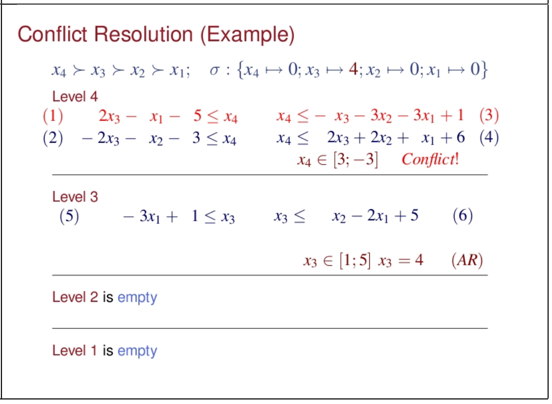
Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

$x_3 \in [1;5]$ $x_3 = 4$ $\quad (AR)$

Level 2 is empty

Level 1 is empty



Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \le x_4$ $\quad x_4 \le -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \le x_4$ $\quad x_4 \le 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [3;-3]$ *Conflict!*

Level 3

(5) $-3x_1 + 1 \le x_3$ $\quad x_3 \le x_2 - 2x_1 + 5$ (6)

$x_3 \in [1;5]$ $x_3 = 4$ $\quad (AR)$

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [3; -3]$ *Conflict!*

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

$x_3 \in [1; 5]$ $x_3 = 4$ (AR)

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

$x_3 \in [1; 2]$

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

$x_3 \in [1; 2]$ $x_3 = 1$ (AR)

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [-3; 0]$

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

$x_3 \in [1; 2]$ $x_3 = 1$ (AR)

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [-3; 0]$ $x_4 = 0$

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

$x_3 \in [1; 2]$ $x_3 = 1$ (AR)

Level 2 is empty

Level 1 is empty

Conflict Resolution (Example)

$x_4 \succ x_3 \succ x_2 \succ x_1$; $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

(1) $2x_3 - x_1 - 5 \leq x_4$ $x_4 \leq -x_3 - 3x_2 - 3x_1 + 1$ (3)

(2) $-2x_3 - x_2 - 3 \leq x_4$ $x_4 \leq 2x_3 + 2x_2 + x_1 + 6$ (4)

$x_4 \in [-3; 0]$ $x_4 = 0$

Level 3

(5) $-3x_1 + 1 \leq x_3$ $x_3 \leq x_2 - 2x_1 + 5$ (6)

$x_3 \leq -x_2 - \frac{2}{3}x_1 + 2$ (CR)

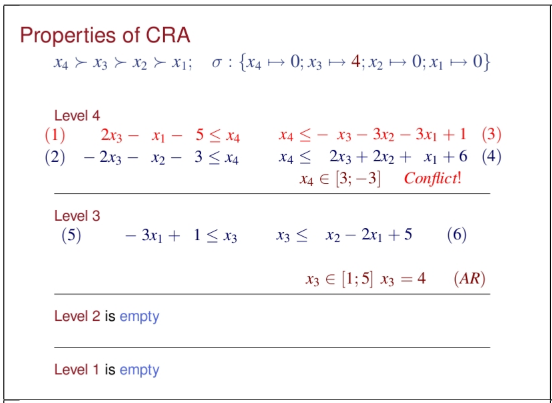
$x_3 \in [1; 2]$ $x_3 = 1$ (AR)

Level 2 is empty

Level 1 is empty

Satisfiable! $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1, x_2 \mapsto 0; x_1 \mapsto 0\}$

Appendix C



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

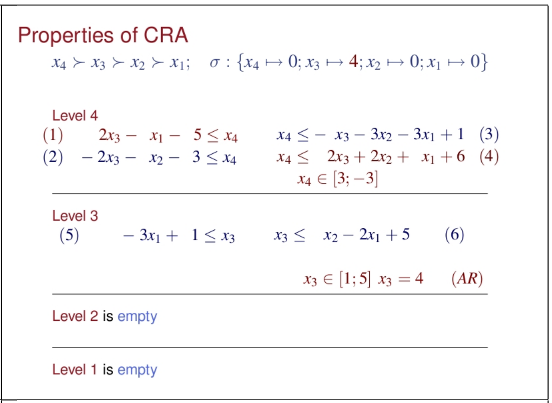
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] \quad \textit{Conflict!} & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

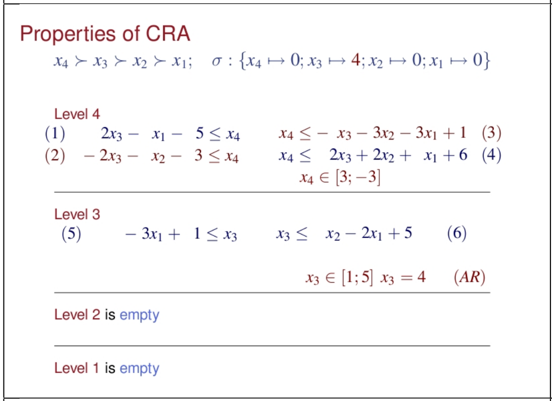
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

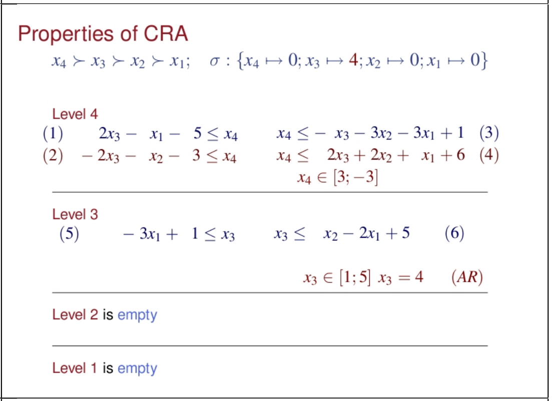
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

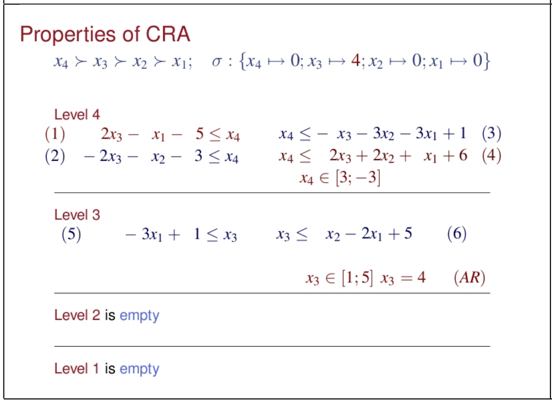
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

Level 4

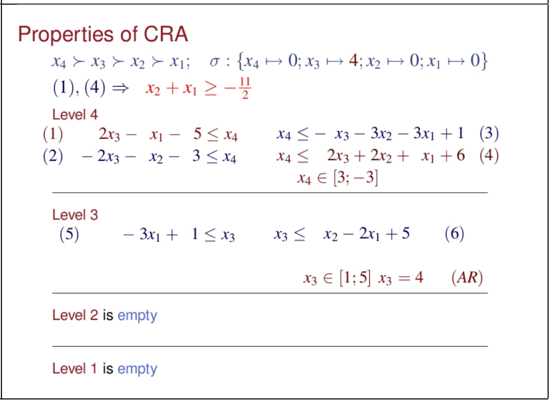
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

$(1), (4) \Rightarrow \quad x_2 + x_1 \ge -\frac{11}{2}$

Level 4

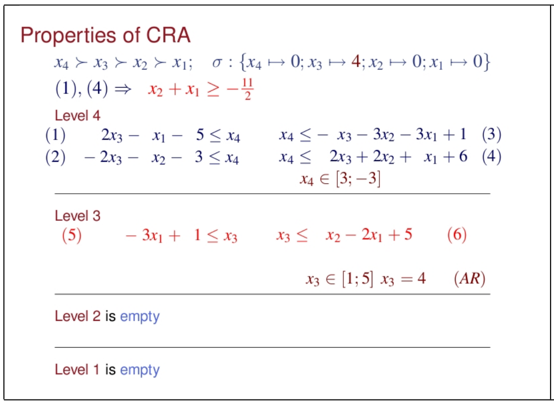
$$\begin{array}{lrlr} (1) & 2x_3 - x_1 - 5 \le x_4 & x_4 \le - x_3 - 3x_2 - 3x_1 + 1 & (3) \\ (2) & -2x_3 - x_2 - 3 \le x_4 & x_4 \le 2x_3 + 2x_2 + x_1 + 6 & (4) \\ & & x_4 \in [3; -3] & \end{array}$$

Level 3

$$\begin{array}{lrlr} (5) & -3x_1 + 1 \le x_3 & x_3 \le x_2 - 2x_1 + 5 & (6) \\ & & x_3 \in [1;5]\ x_3 = 4 & (AR) \end{array}$$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

$(1), (4) \Rightarrow \quad x_2 + x_1 \geq -\frac{11}{2}$

Level 4

$$\begin{array}{lrl}
(1) & 2x_3 - \ x_1 - \ 5 \leq x_4 & x_4 \leq - \ x_3 - 3x_2 - 3x_1 + 1 \quad (3) \\
(2) & -2x_3 - \ x_2 - \ 3 \leq x_4 & x_4 \leq \ 2x_3 + 2x_2 + \ x_1 + 6 \quad (4)
\end{array}$$

$x_4 \in [3; -3]$

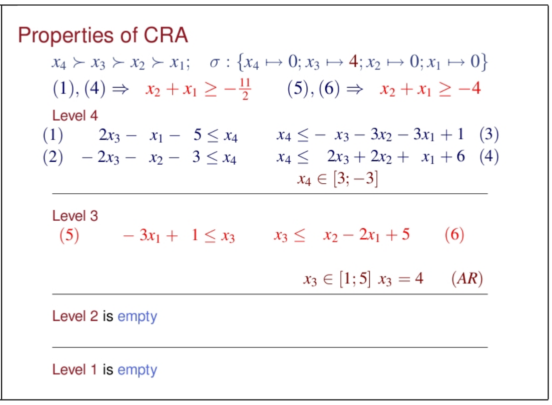
Level 3

$(5) \quad -3x_1 + \ 1 \leq x_3 \qquad x_3 \leq \ x_2 - 2x_1 + 5 \quad (6)$

$x_3 \in [1; 5] \ x_3 = 4 \qquad (AR)$

Level 2 is empty

Level 1 is empty



Properties of CRA

$x_4 \succ x_3 \succ x_2 \succ x_1; \quad \sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$

$(1), (4) \Rightarrow \quad x_2 + x_1 \geq -\frac{11}{2} \qquad (5), (6) \Rightarrow \quad x_2 + x_1 \geq -4$

Level 4

$$\begin{array}{lrl}
(1) & 2x_3 - \ x_1 - \ 5 \leq x_4 & x_4 \leq - \ x_3 - 3x_2 - 3x_1 + 1 \quad (3) \\
(2) & -2x_3 - \ x_2 - \ 3 \leq x_4 & x_4 \leq \ 2x_3 + 2x_2 + \ x_1 + 6 \quad (4)
\end{array}$$

$x_4 \in [3; -3]$

Level 3

$(5) \quad -3x_1 + \ 1 \leq x_3 \qquad x_3 \leq \ x_2 - 2x_1 + 5 \quad (6)$

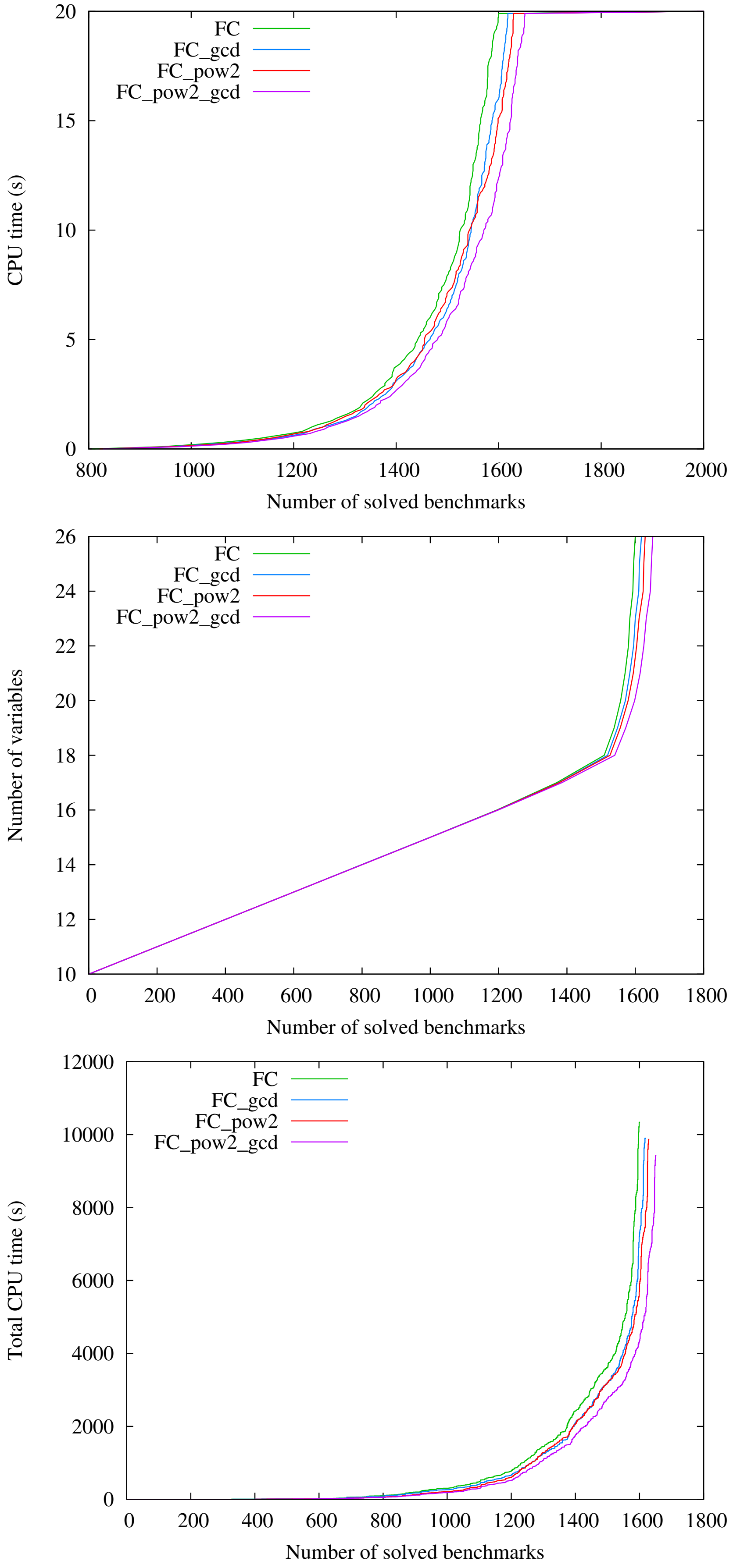
$x_3 \in [1; 5] \ x_3 = 4 \qquad (AR)$

Level 2 is empty

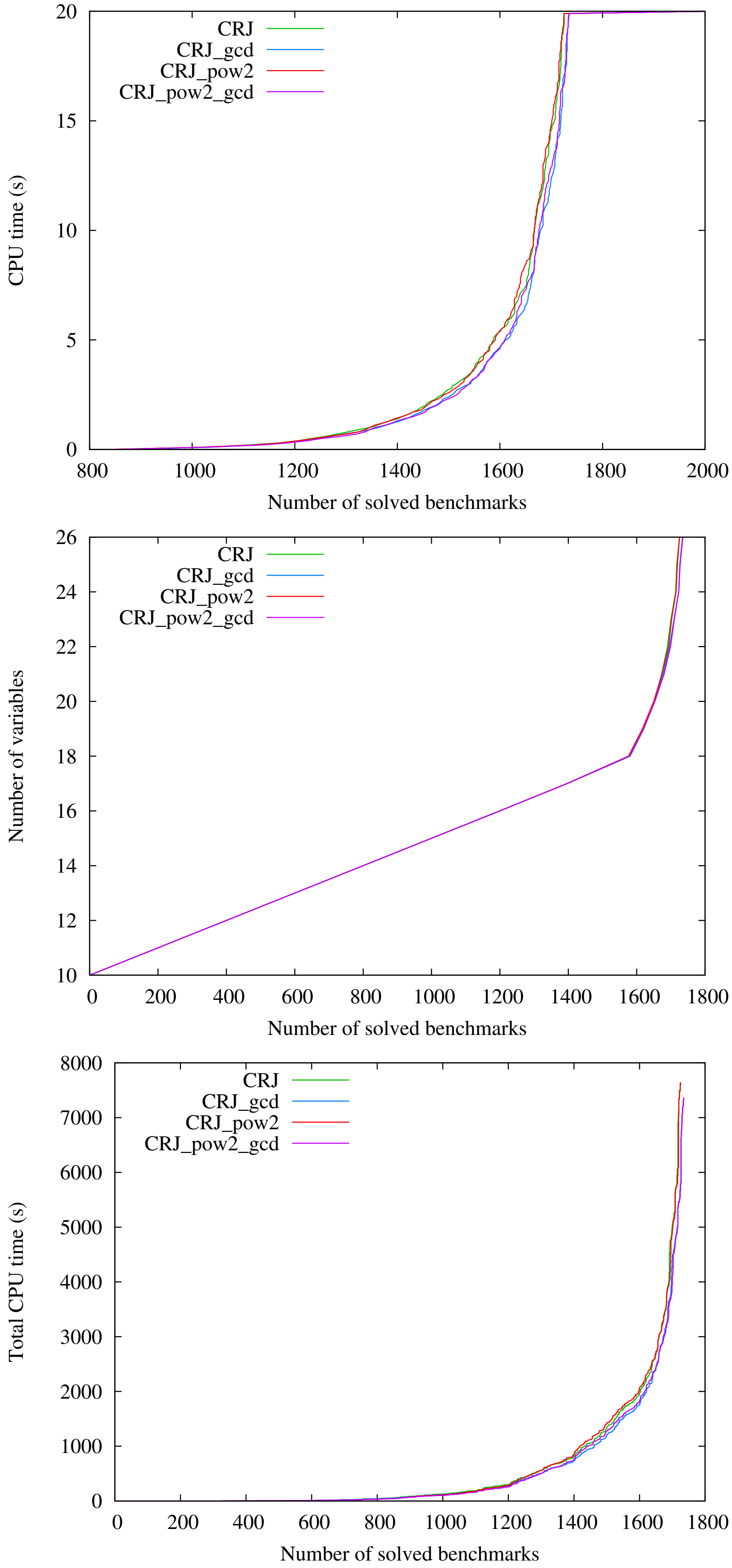
Level 1 is empty

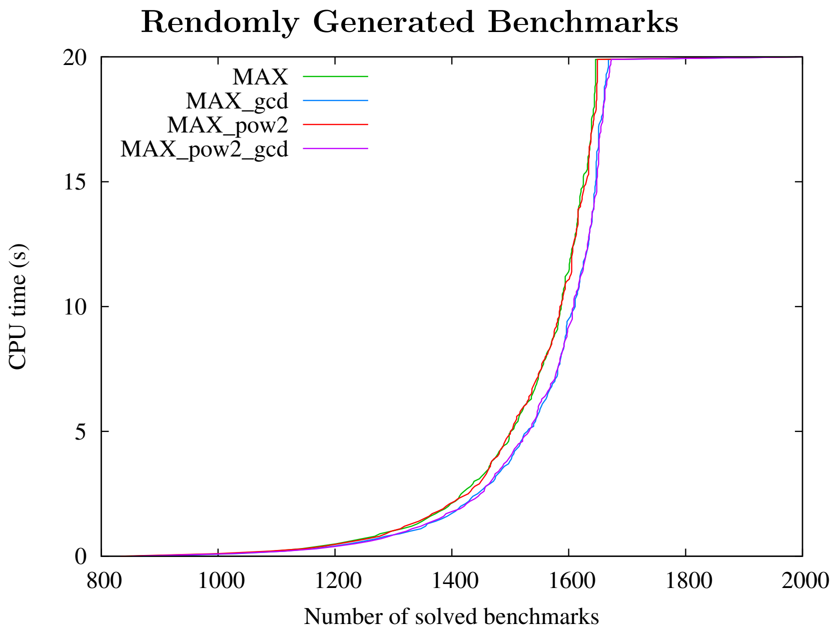
Appendix D

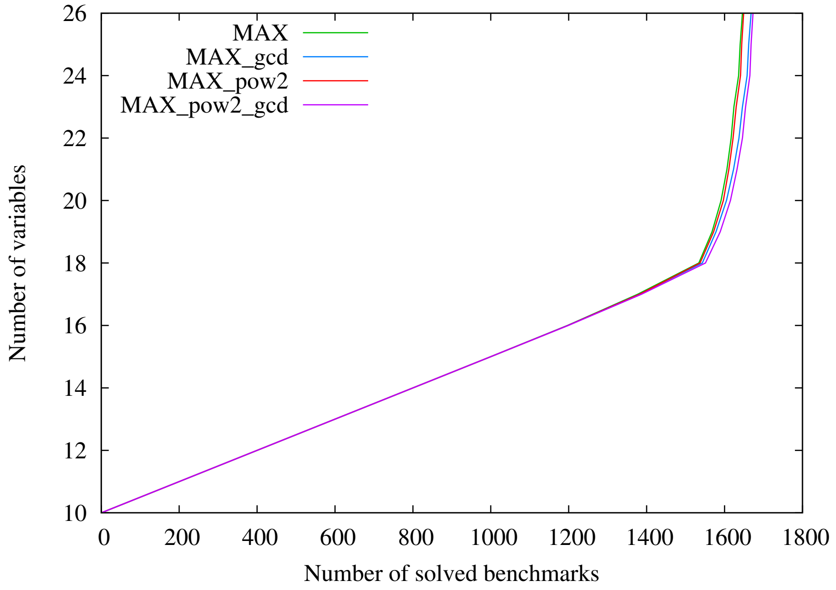
Rendomly Generated Benchmarks

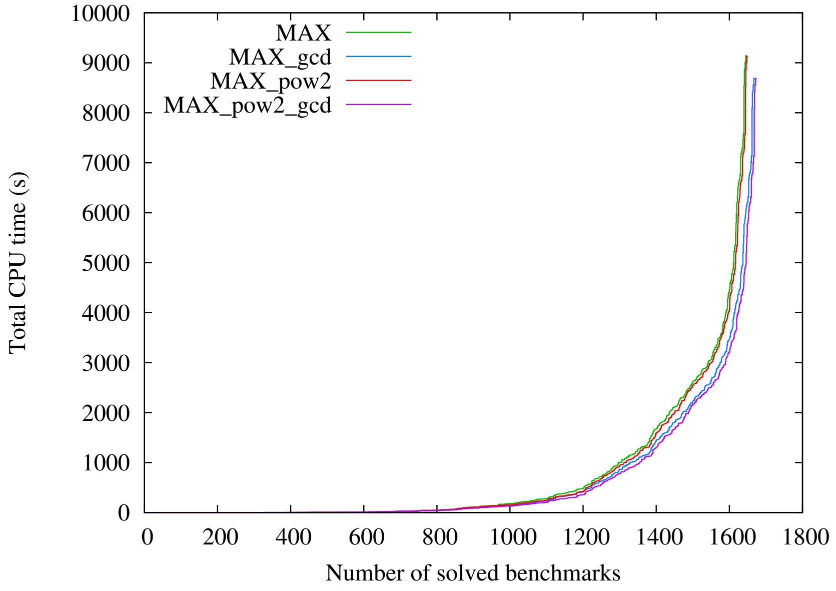


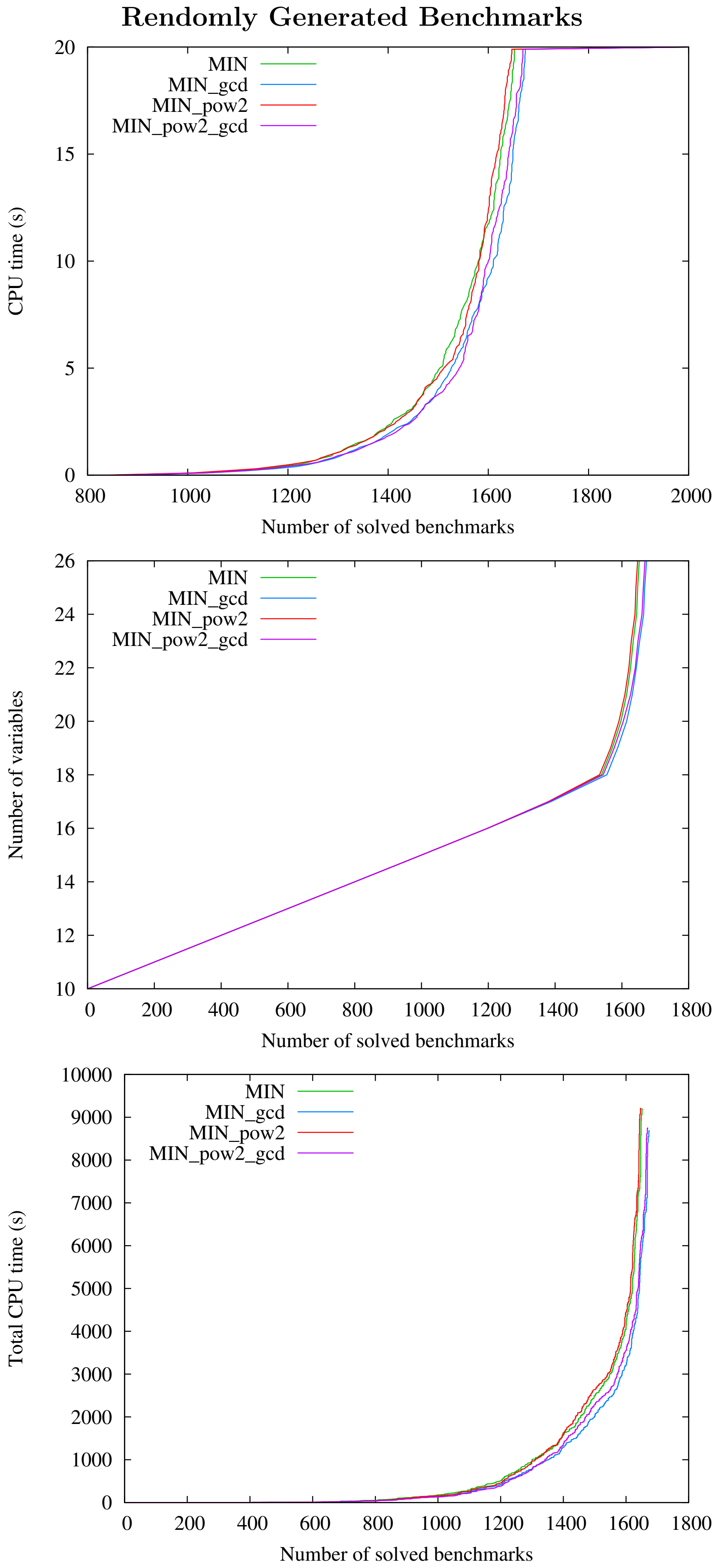
Rendomly Generated Benchmarks




Rendomly Generated Benchmarks
CPU time (s)
20
15
10
5
0
800
1000
1200
1400
1600
1800
2000
Number of solved benchmarks
MAX
MAX_gcd
MAX_pow2
MAX_pow2_gcd

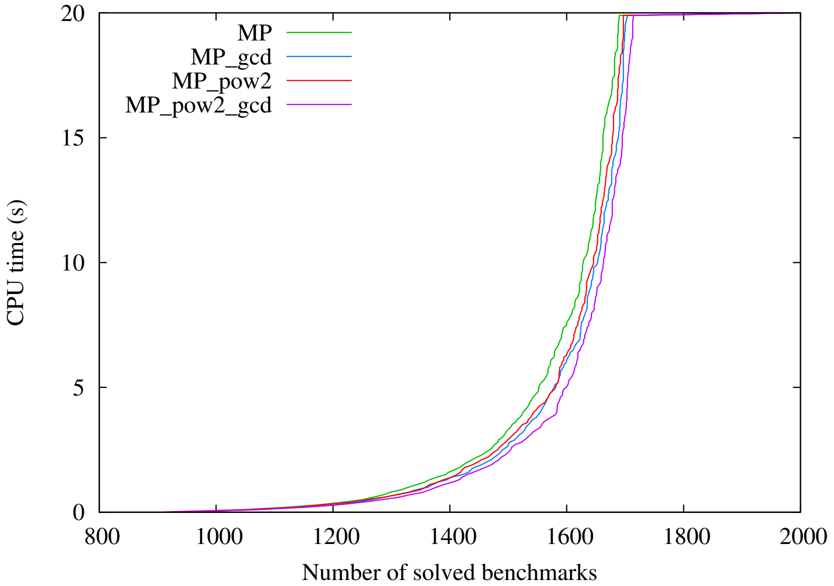

Number of variables
26
24
22
20
18
16
14
12
10
0
200
400
600
800
1000
1200
1400
1600
1800
Number of solved benchmarks
MAX
MAX_gcd
MAX_pow2
MAX_pow2_gcd

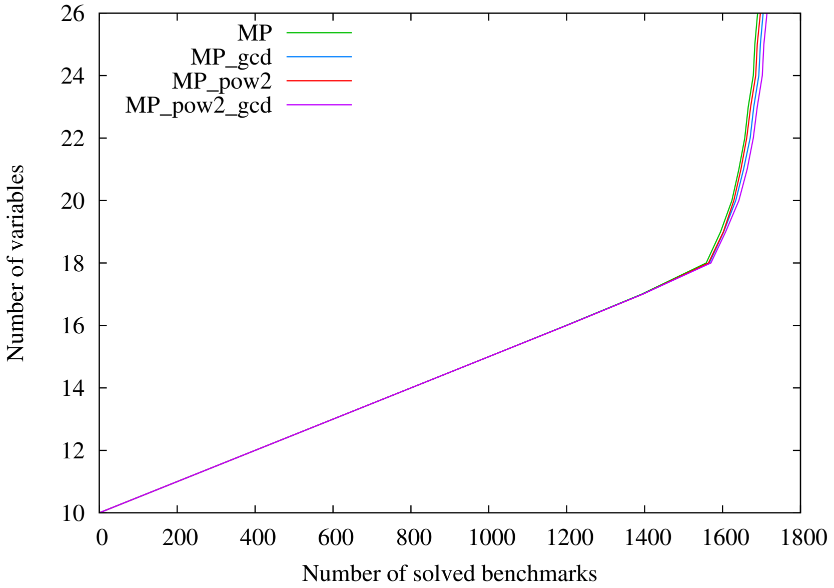

Total CPU time (s)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200
1400
1600
1800
Number of solved benchmarks
MAX
MAX_gcd
MAX_pow2
MAX_pow2_gcd

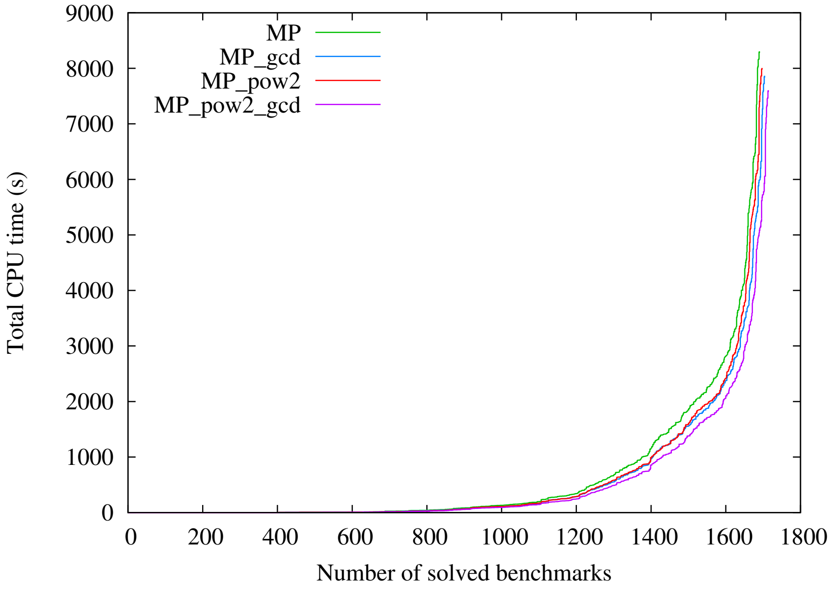


Rendomly Generated Benchmarks

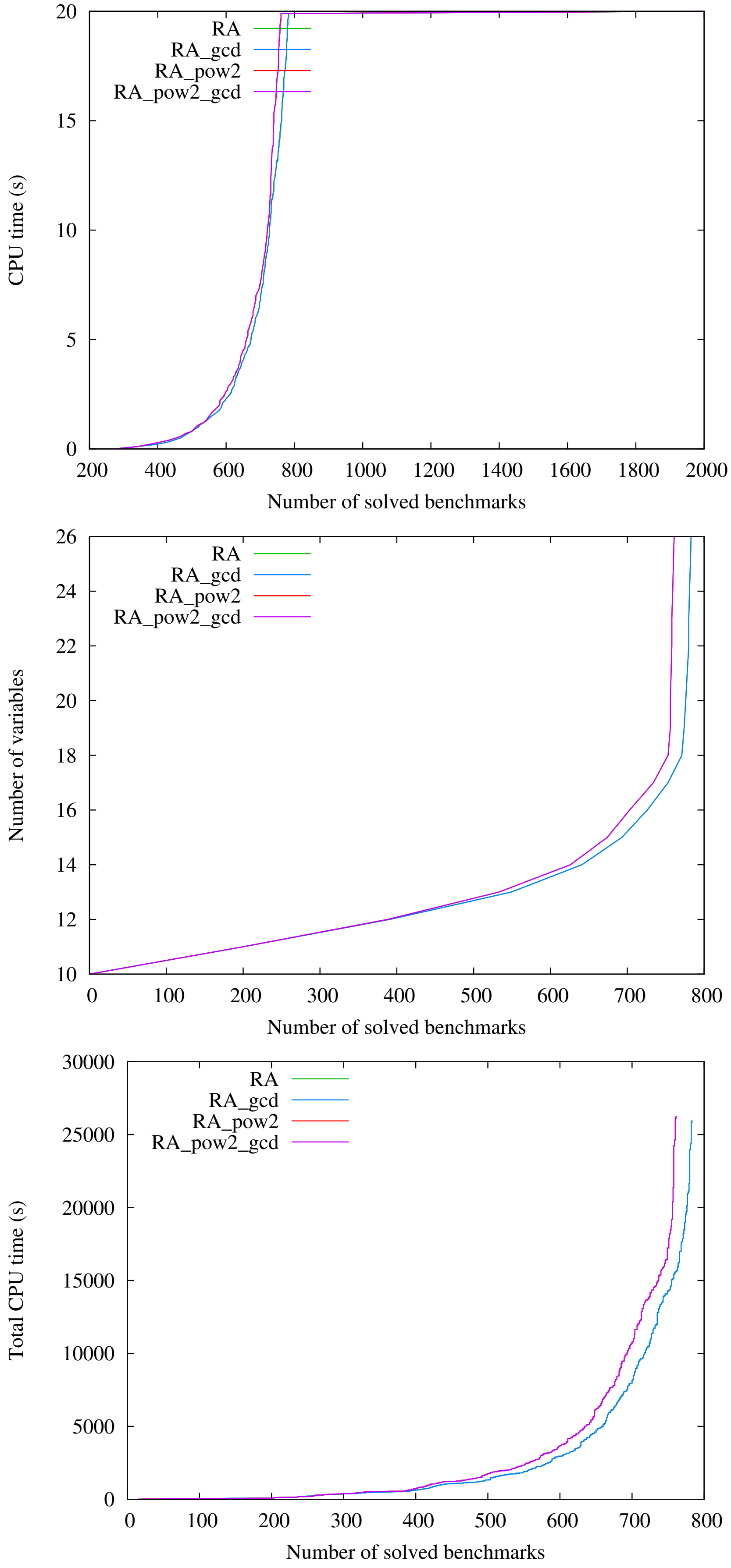
Rendomly Generated Benchmarks



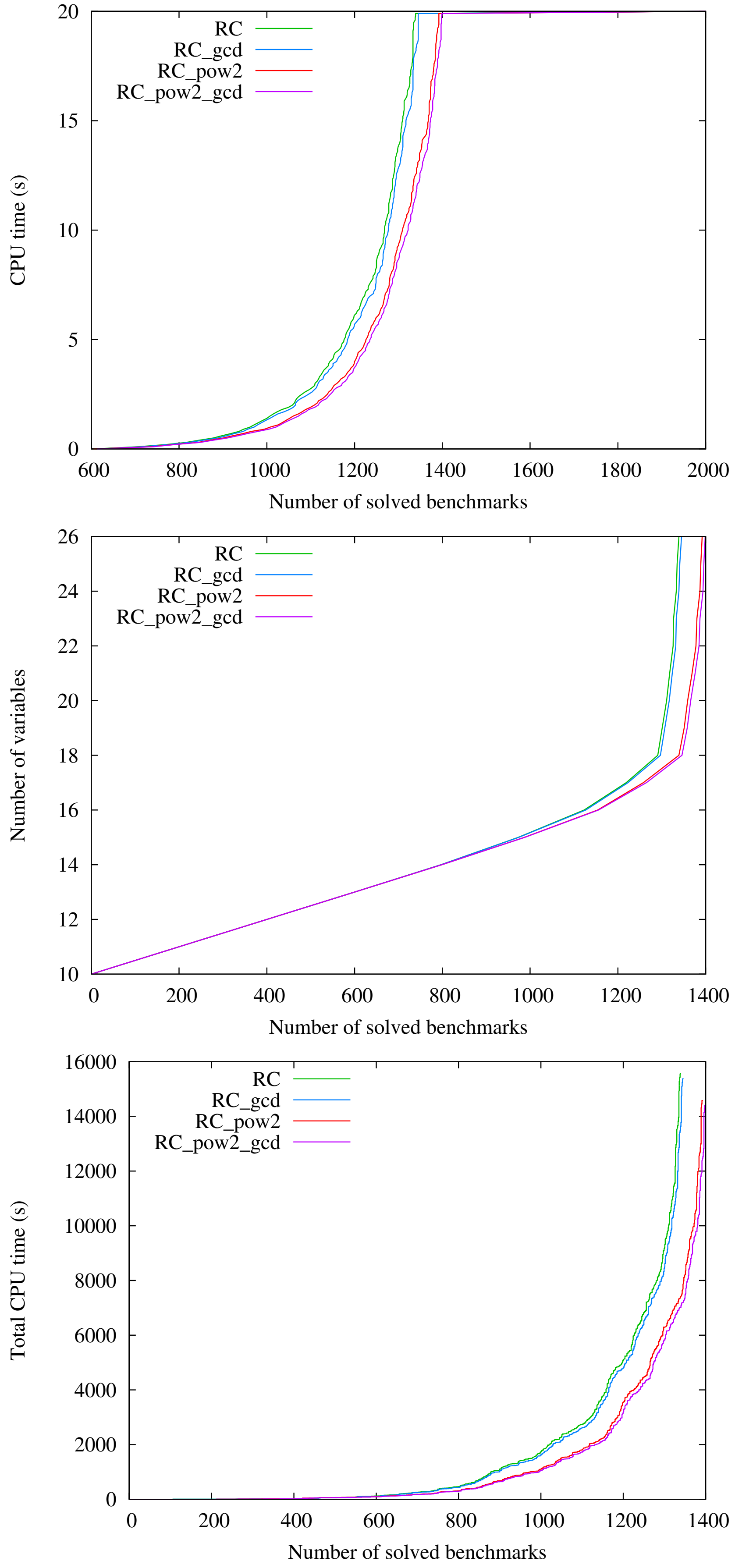




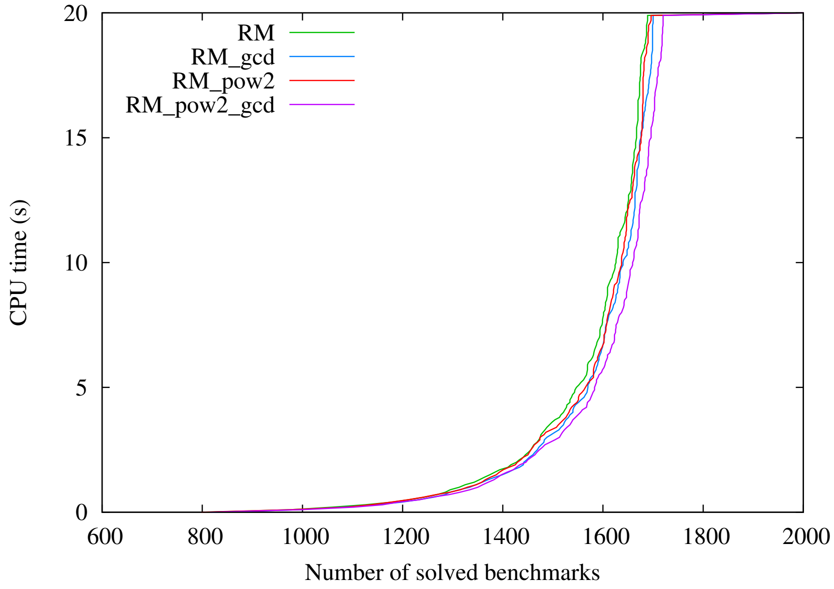
Rendomly Generated Benchmarks

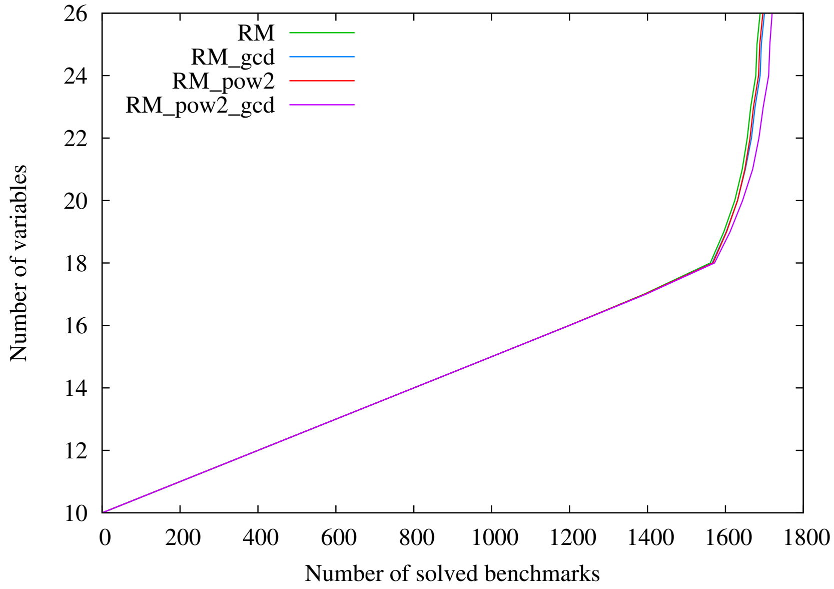


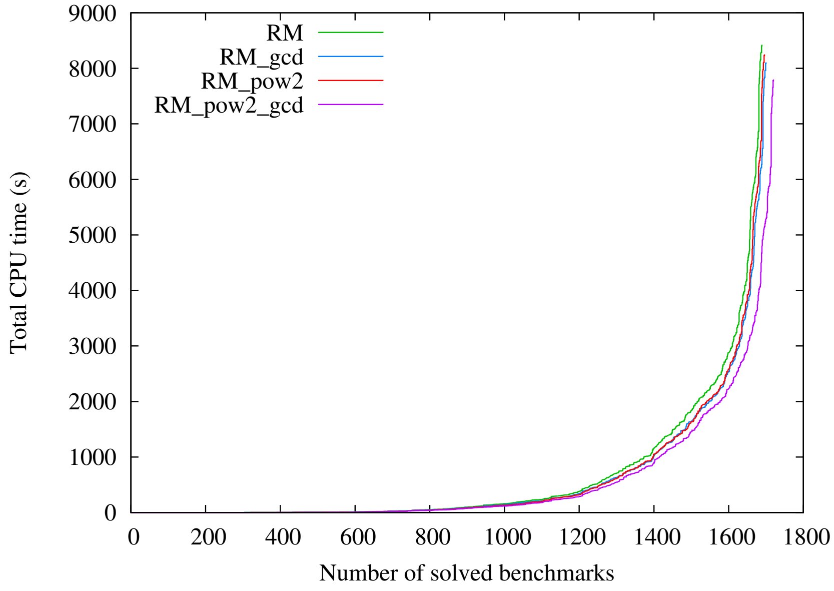
Rendomly Generated Benchmarks

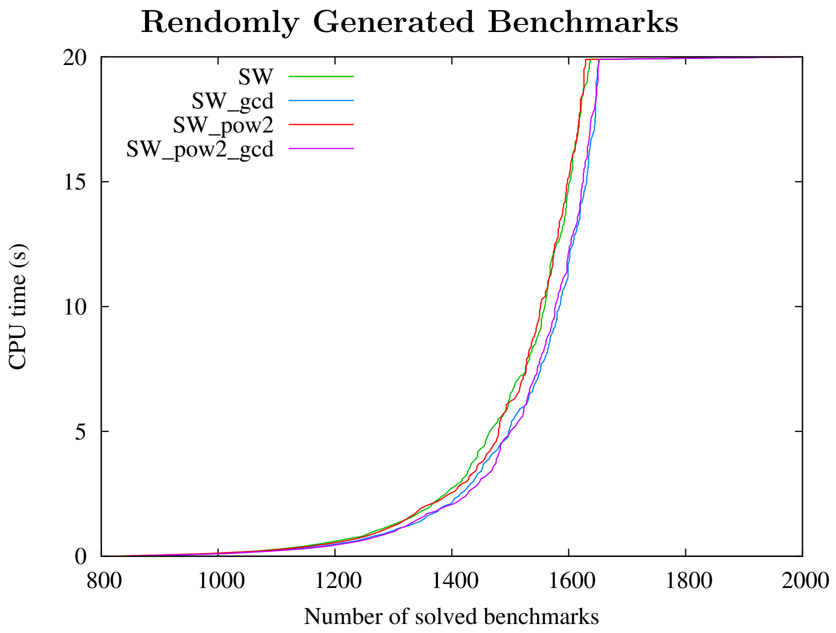


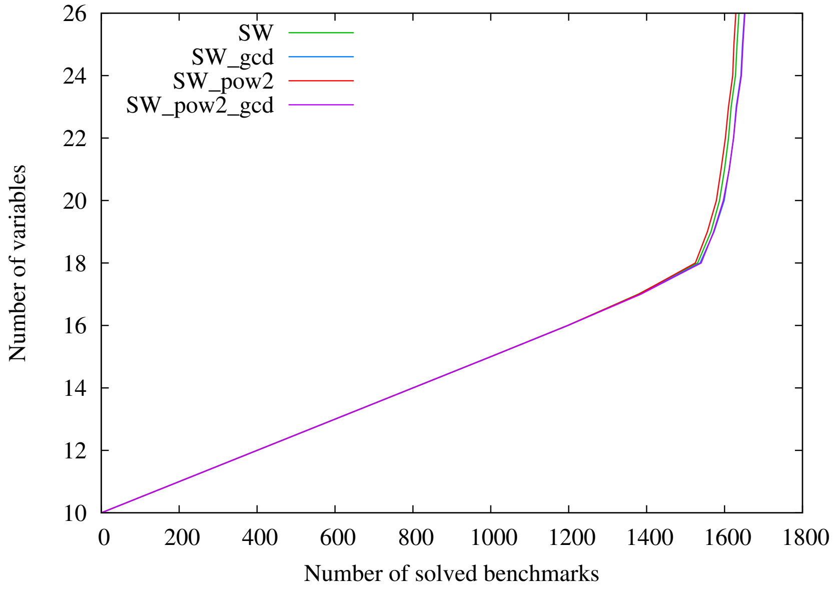
Rendomly Generated Benchmarks

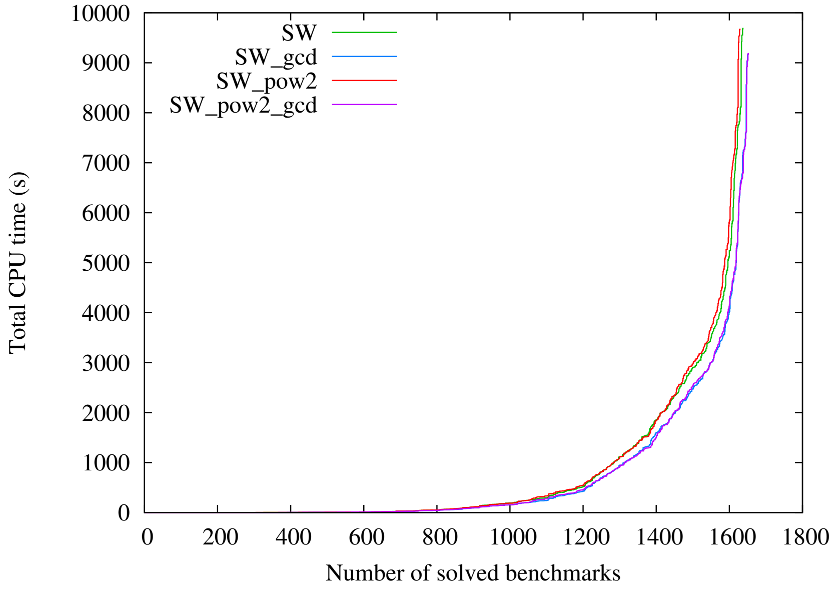




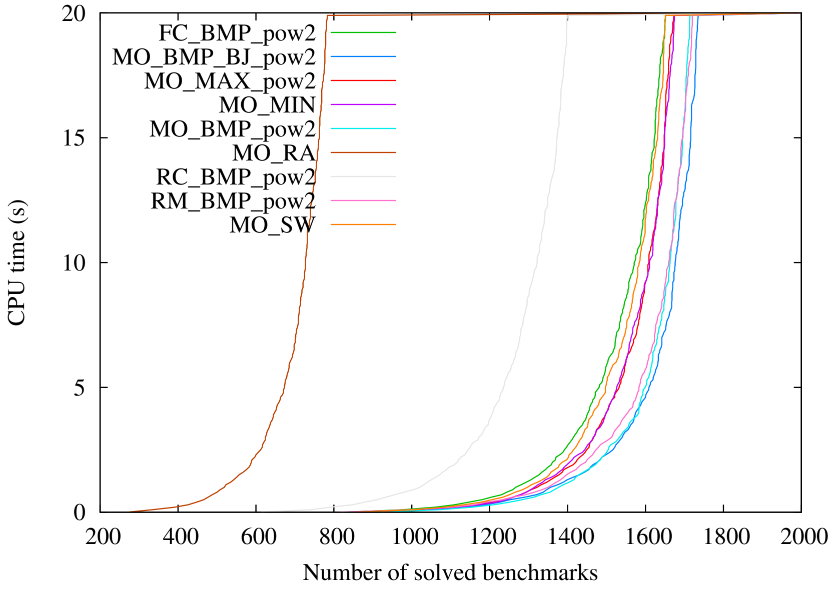


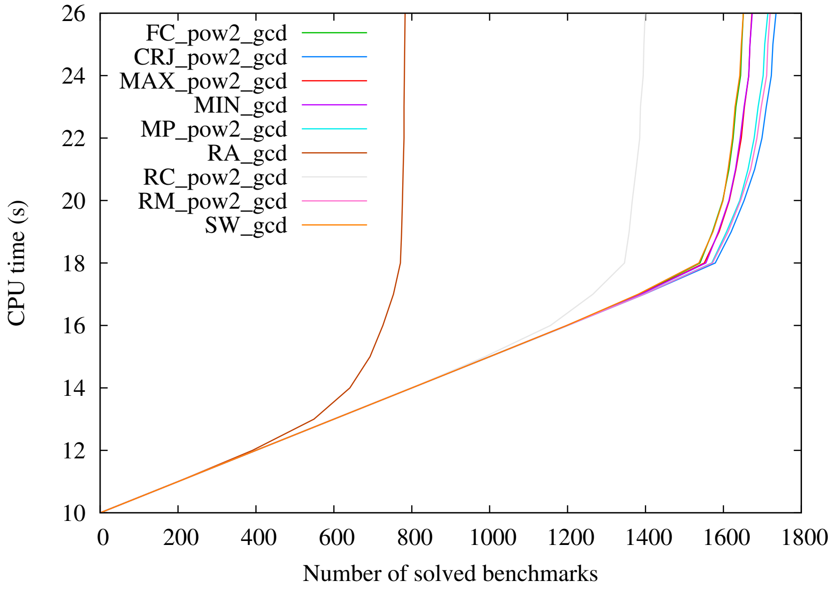

Rendomly Generated Benchmarks
SW
SW_gcd
SW_pow2
SW_pow2_gcd
CPU time (s)
0
5
10
15
20
800
1000
1200
1400
1600
1800
2000
Number of solved benchmarks

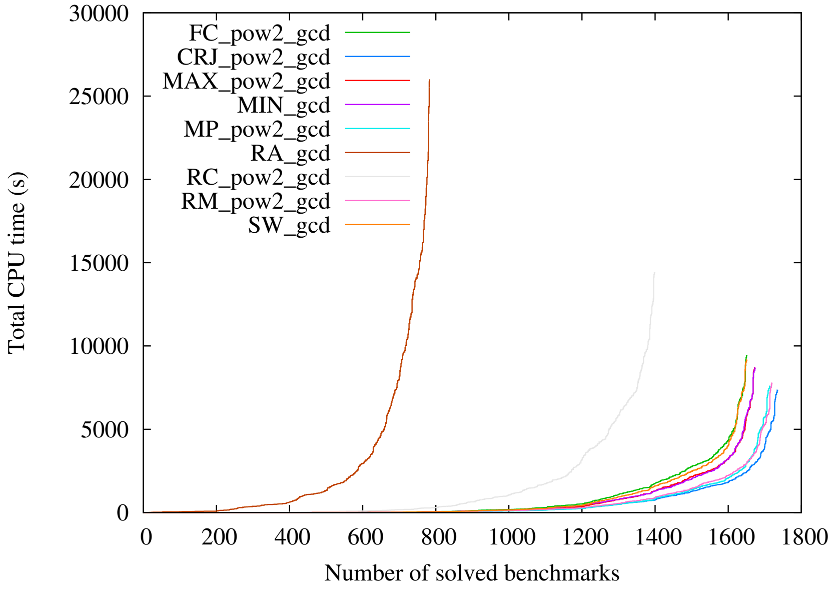

SW
SW_gcd
SW_pow2
SW_pow2_gcd
Number of variables
10
12
14
16
18
20
22
24
26
0
200
400
600
800
1000
1200
1400
1600
1800
Number of solved benchmarks


SW
SW_gcd
SW_pow2
SW_pow2_gcd
Total CPU time (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
200
400
600
800
1000
1200
1400
1600
1800
Number of solved benchmarks

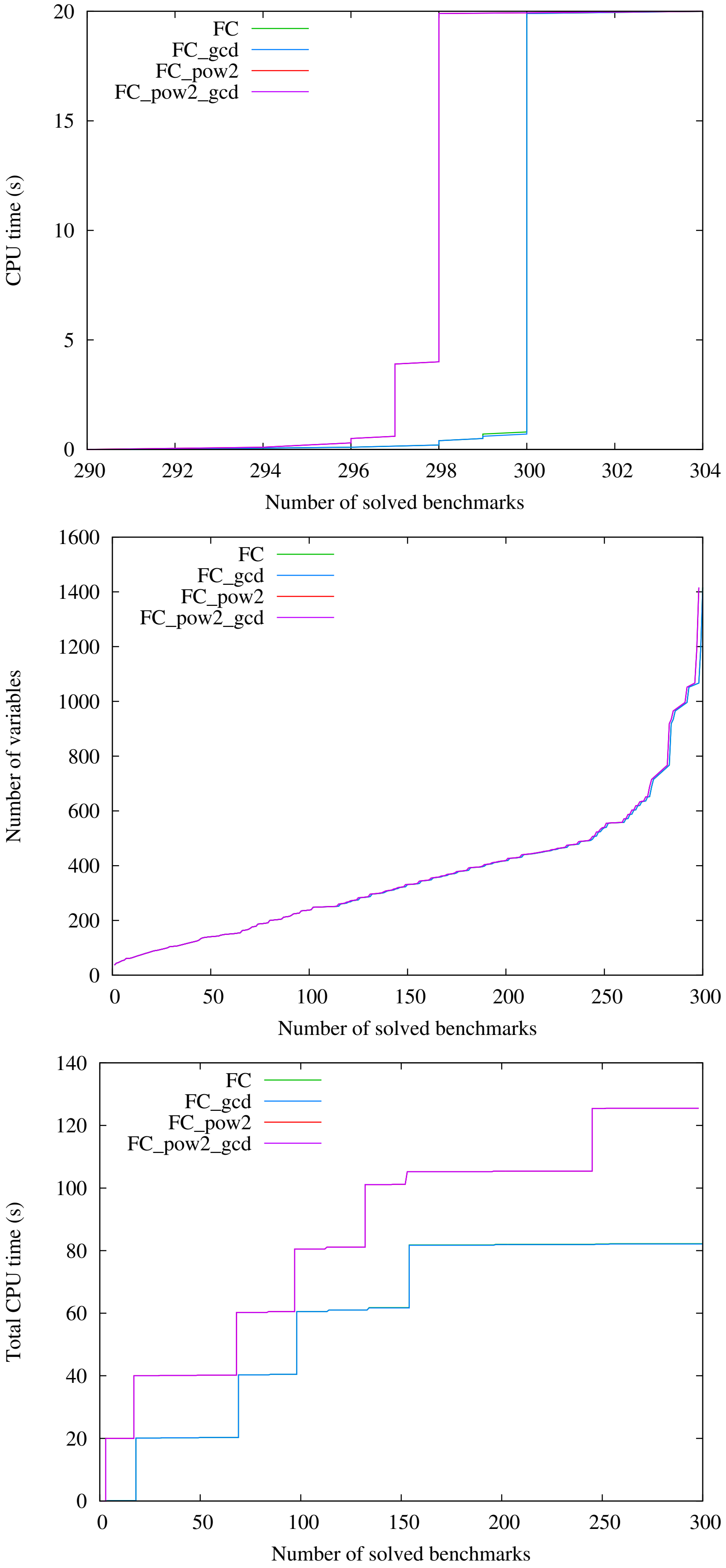
Rendomly Generated Benchmarks



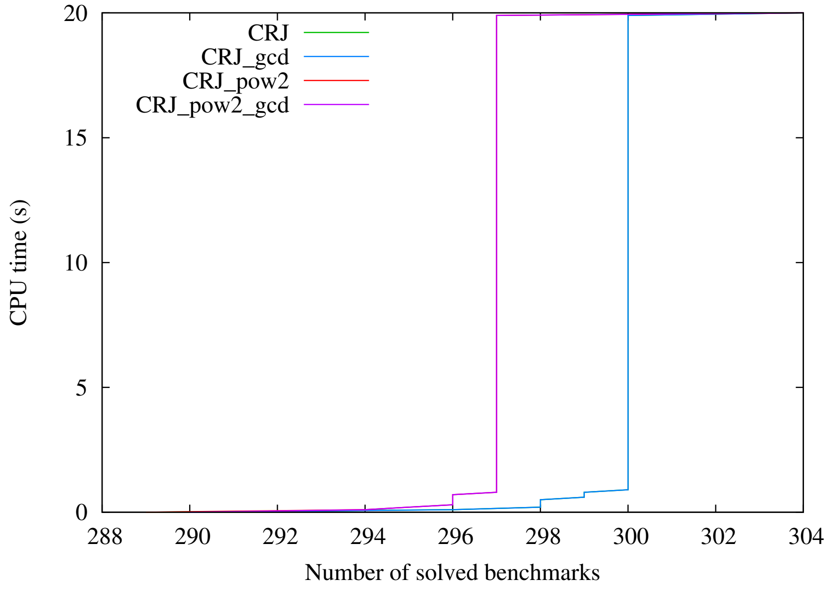


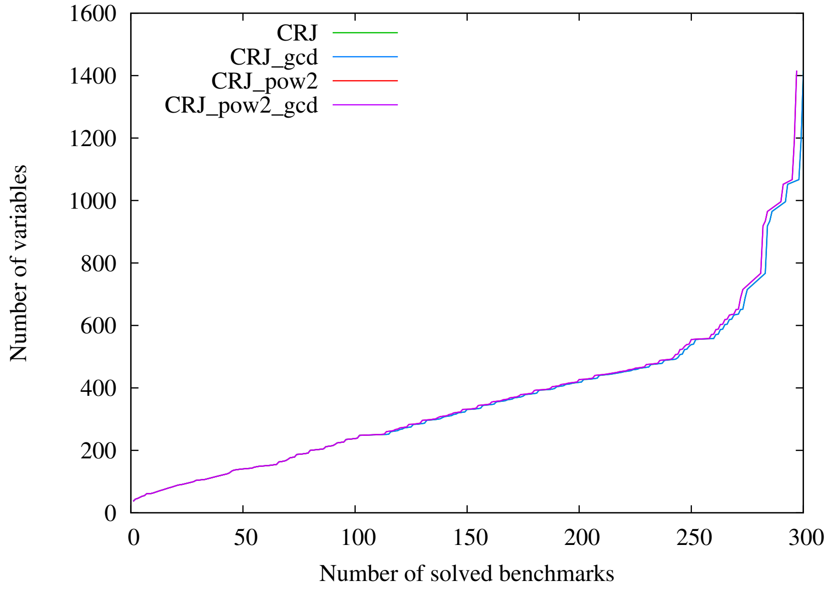


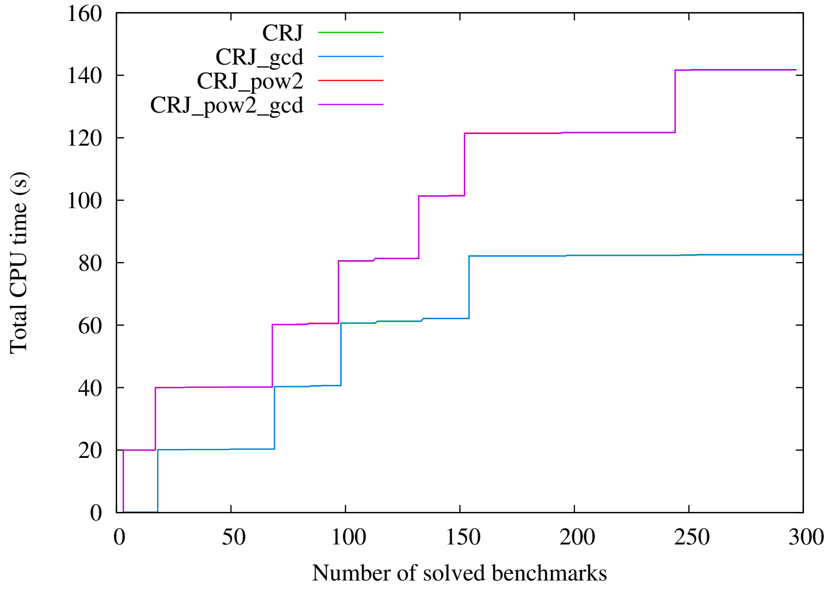
Real-Life Benchmarks (The First Set)



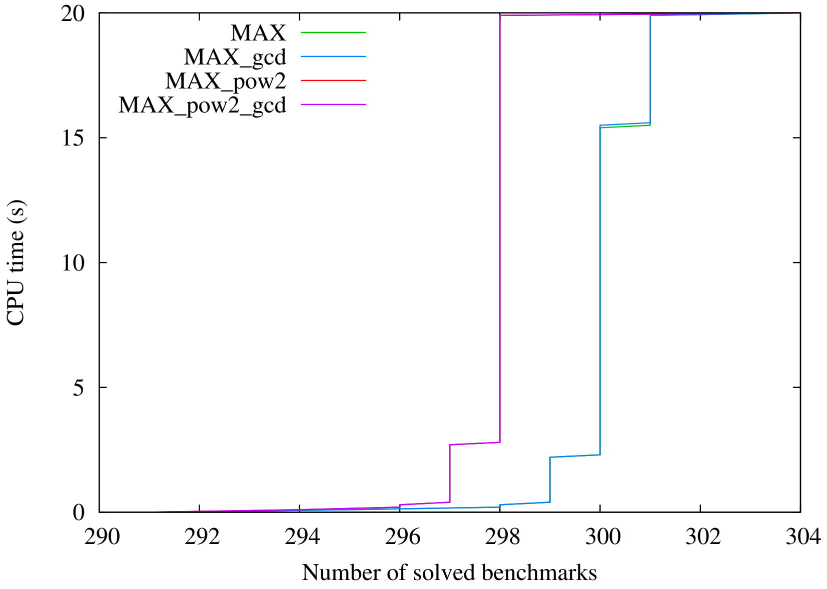
Real-Life Benchmarks (The First Set)

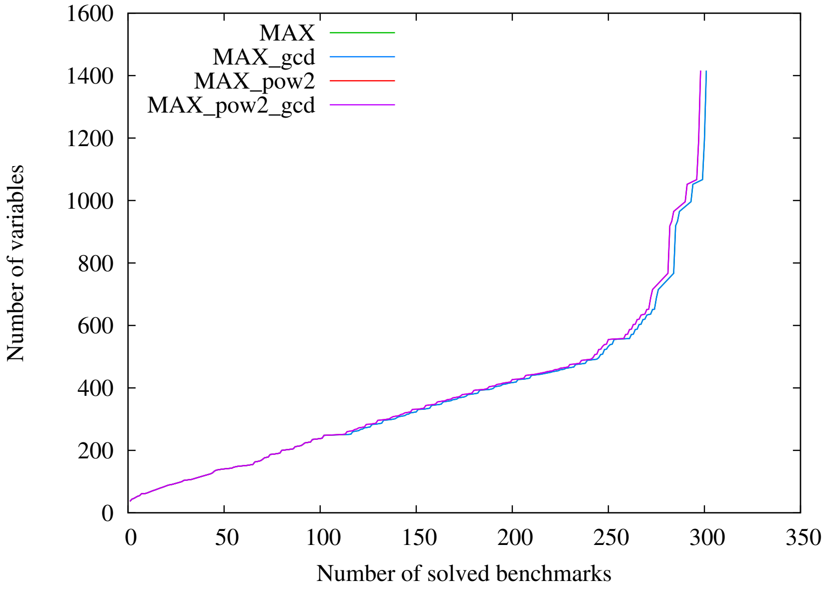


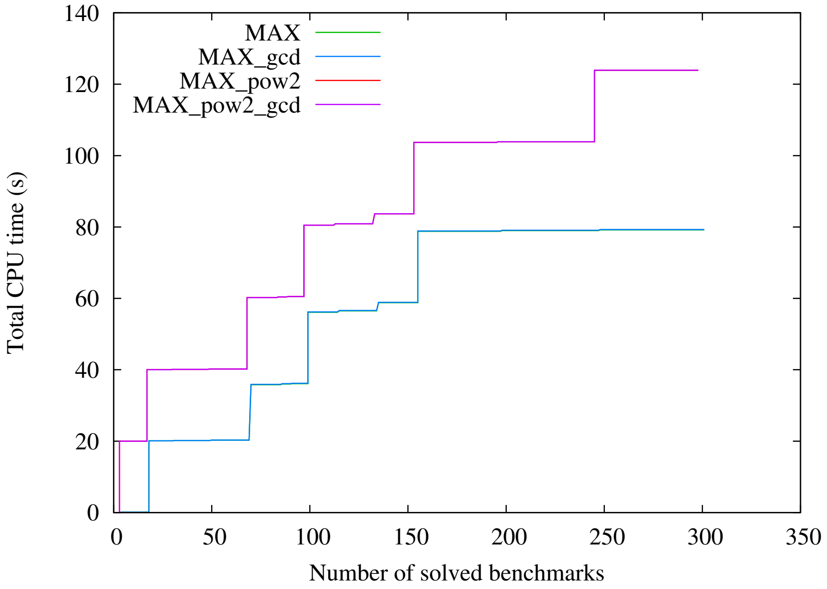




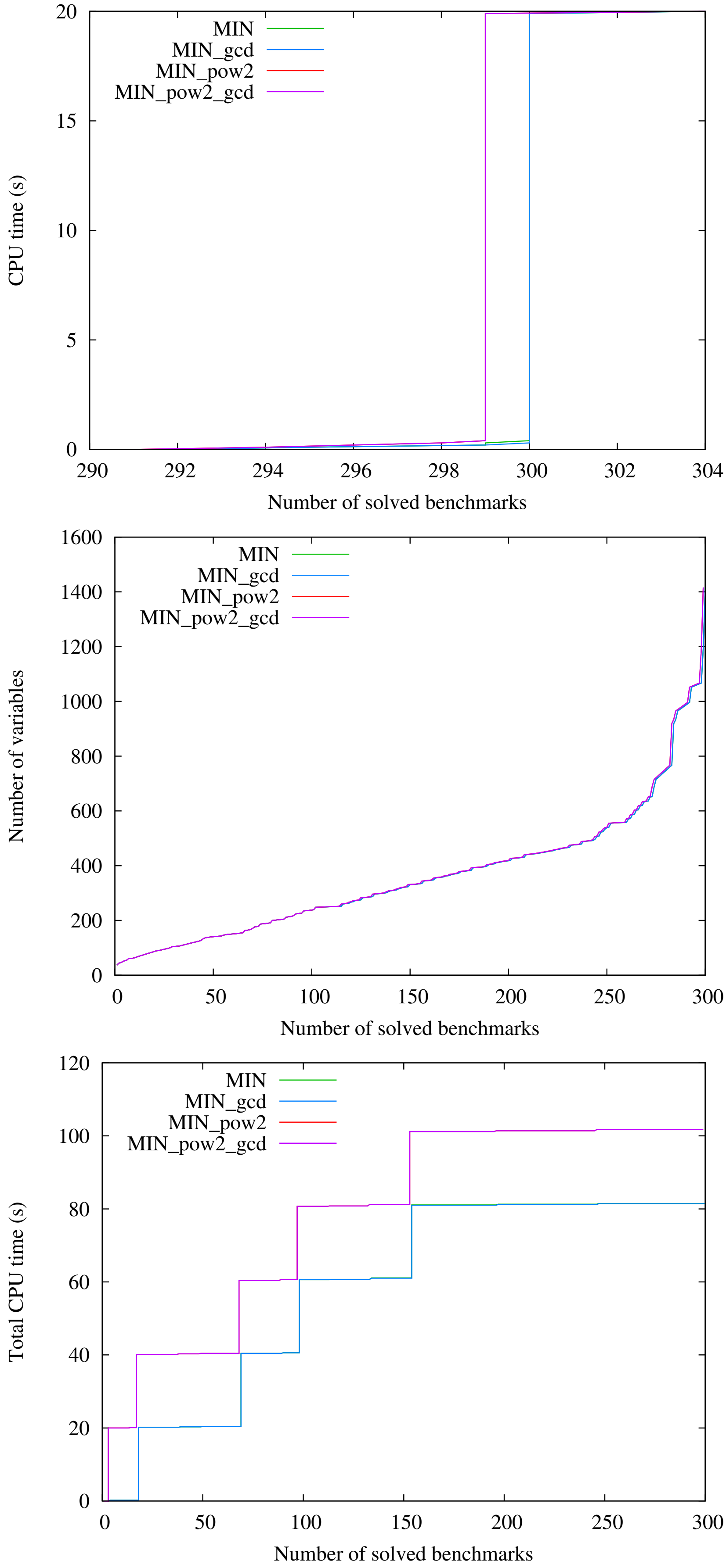
Real-Life Benchmarks (The First Set)



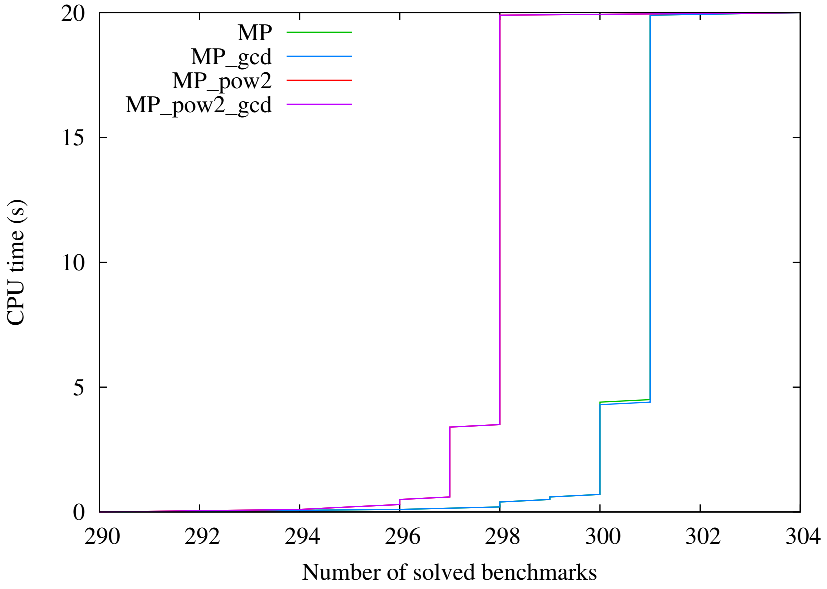


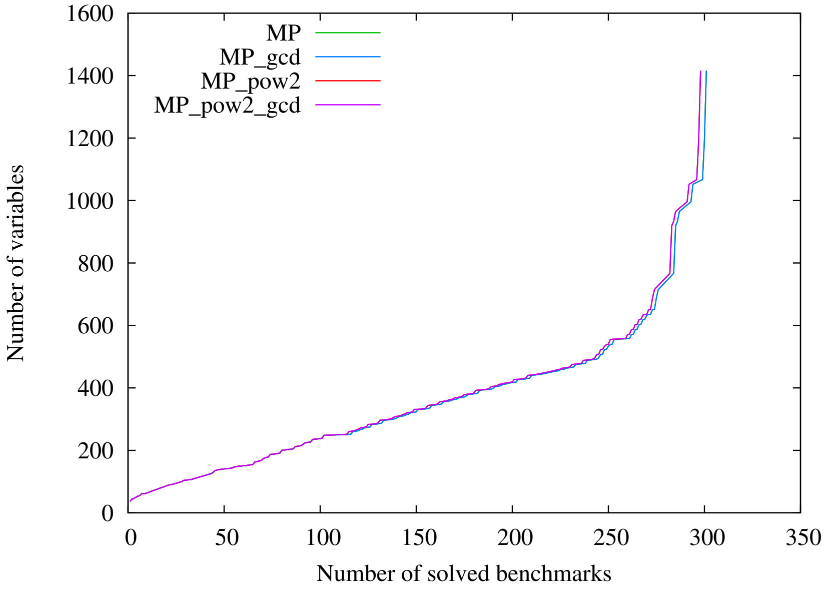


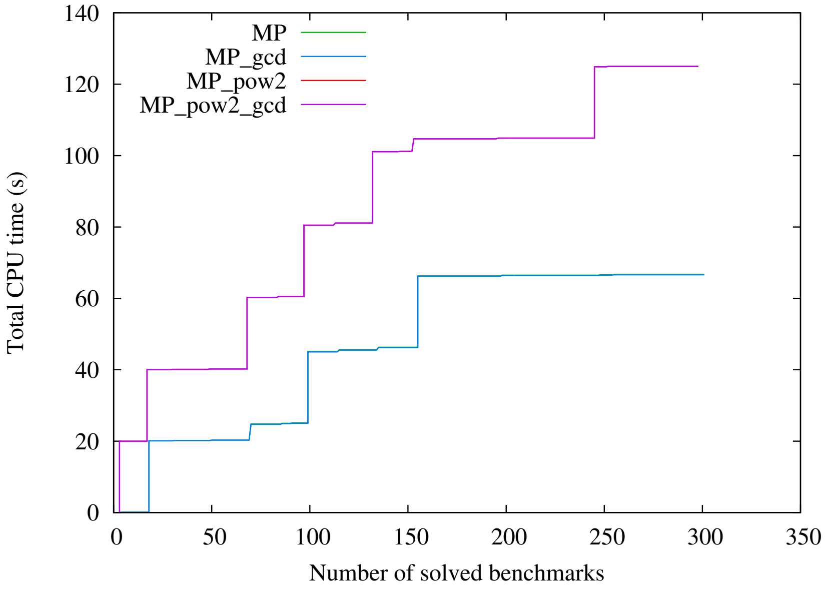
Real-Life Benchmarks (The First Set)



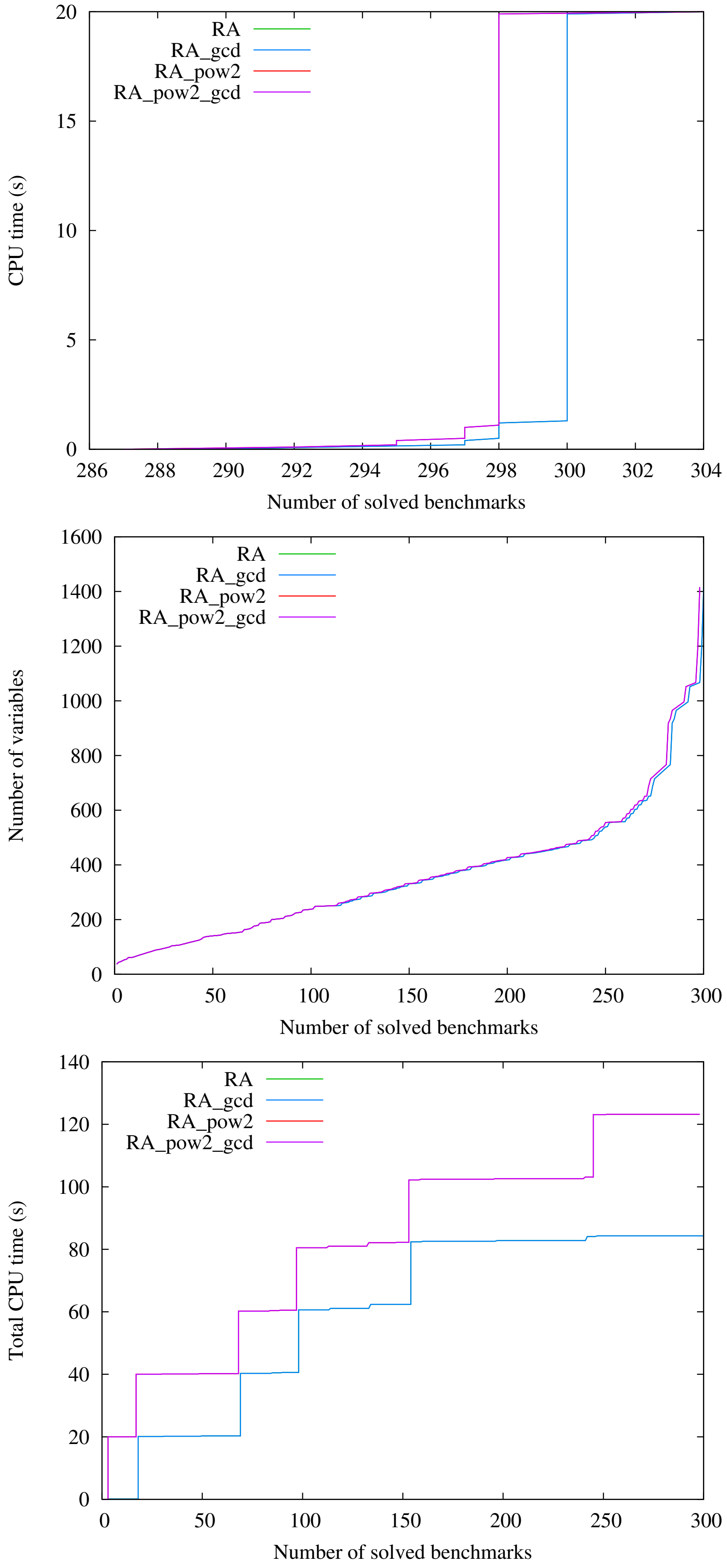
Real-Life Benchmarks (The First Set)



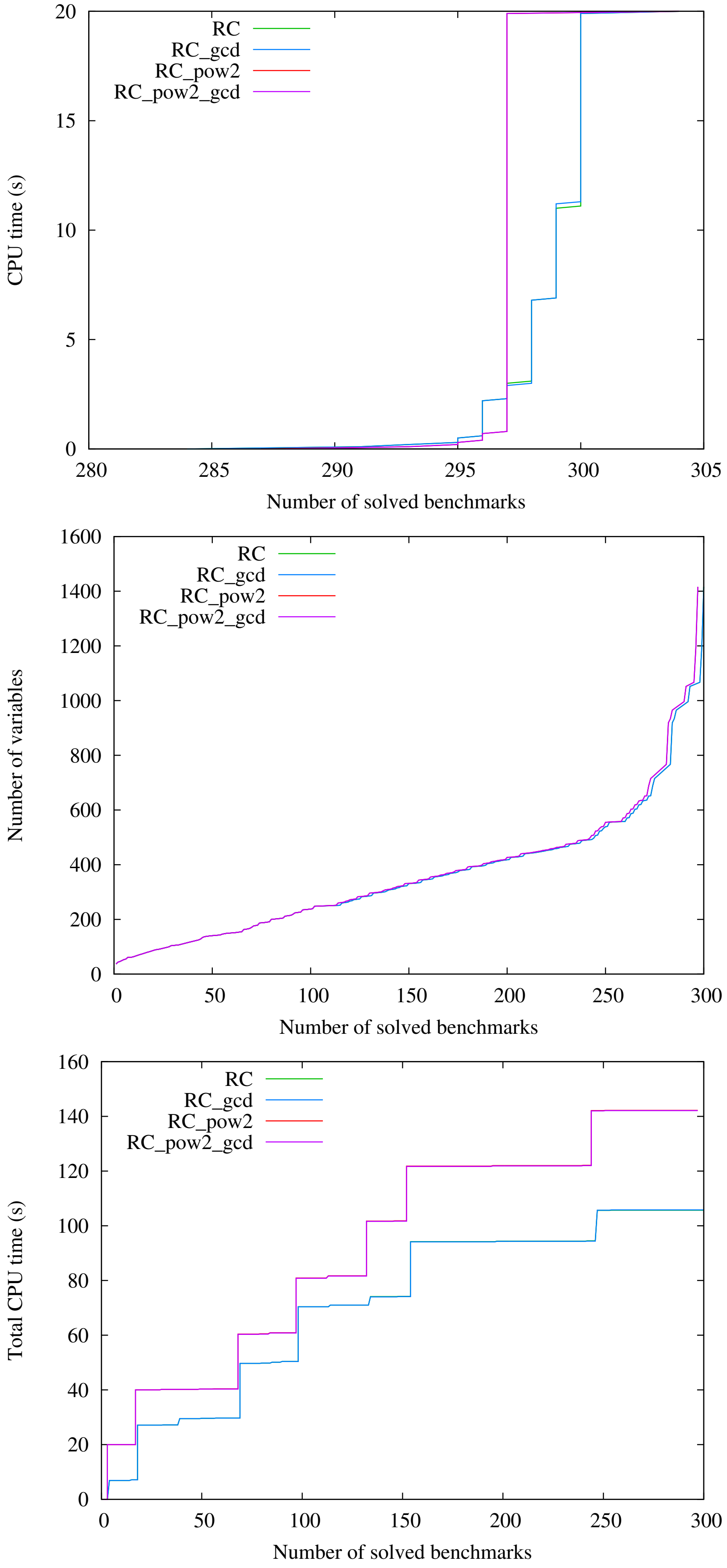




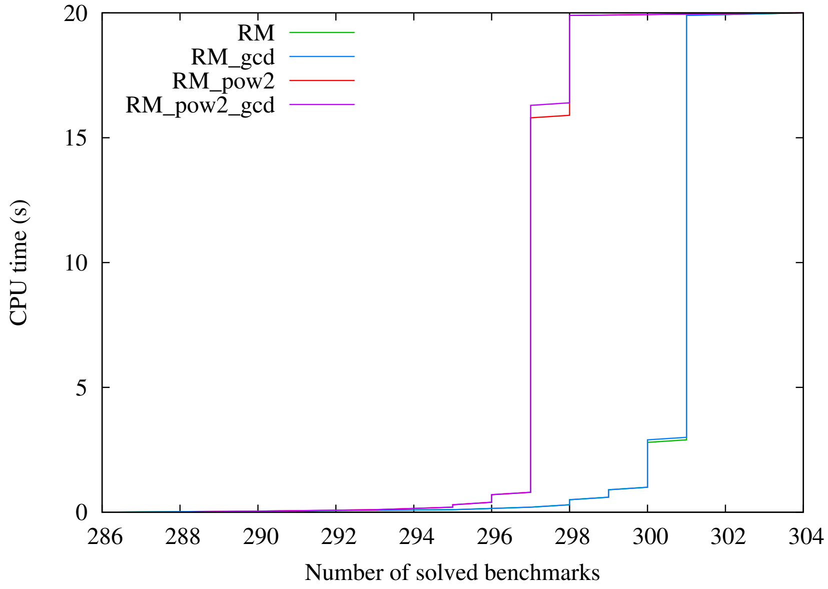
Real-Life Benchmarks (The First Set)

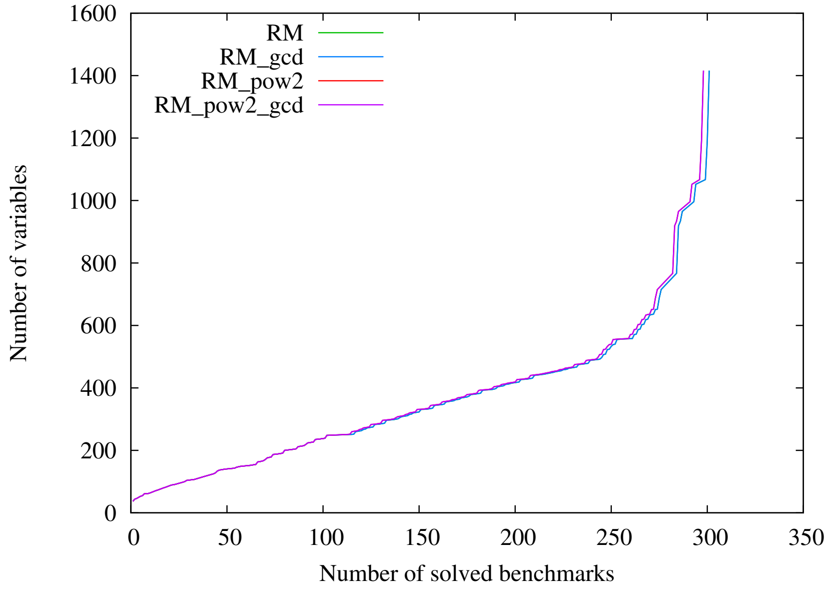


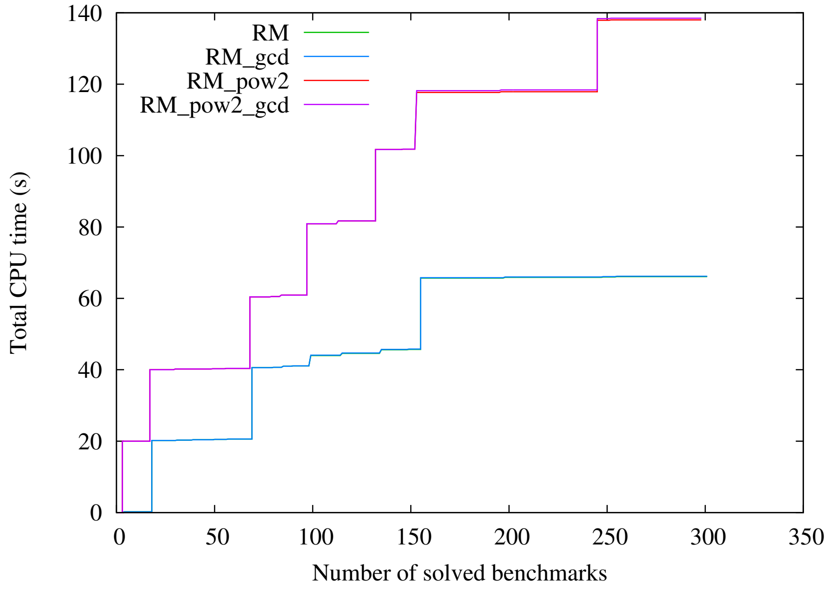
Real-Life Benchmarks (The First Set)



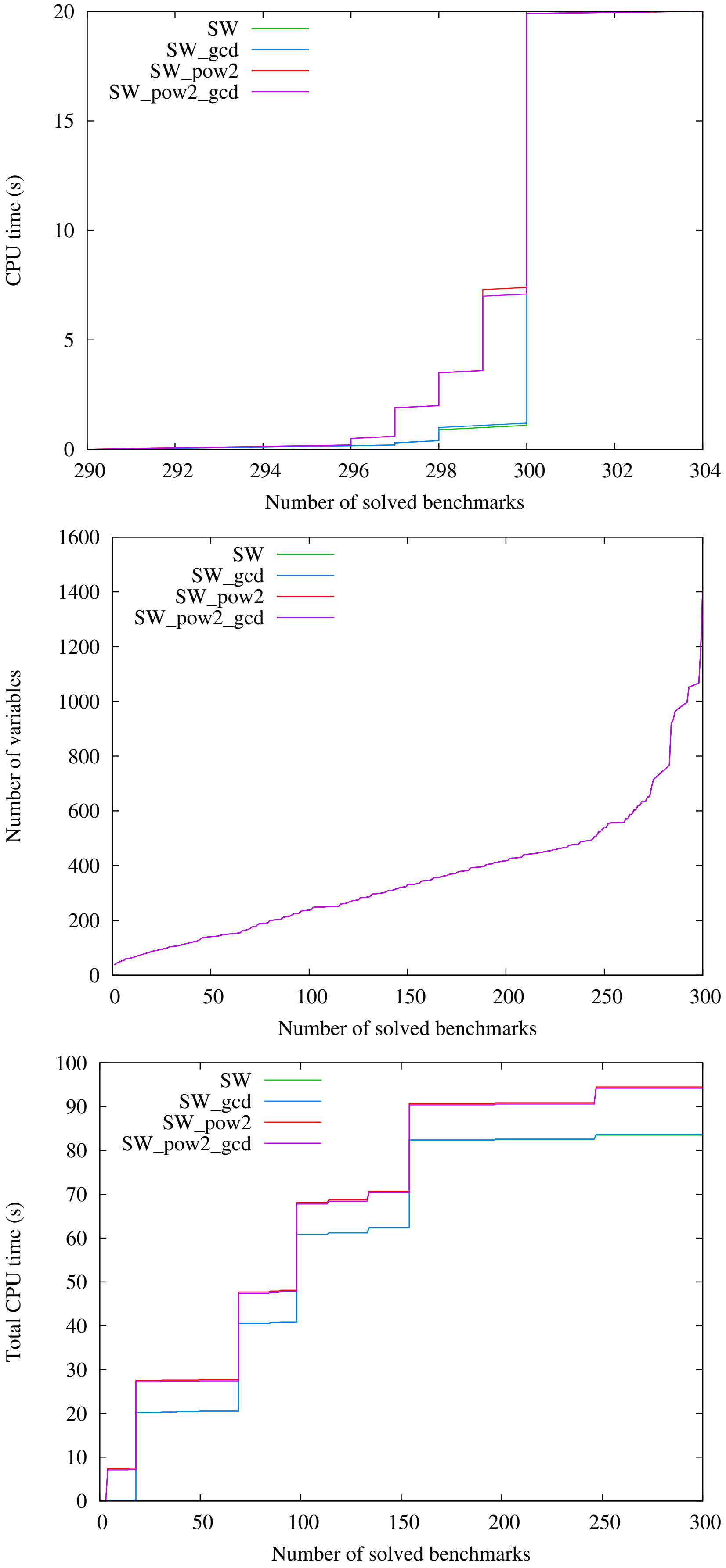
Real-Life Benchmarks (The First Set)

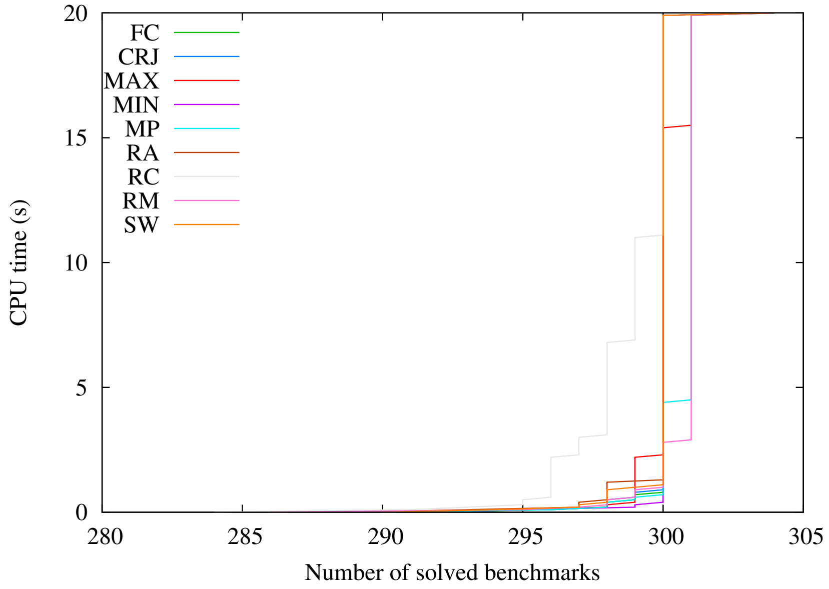




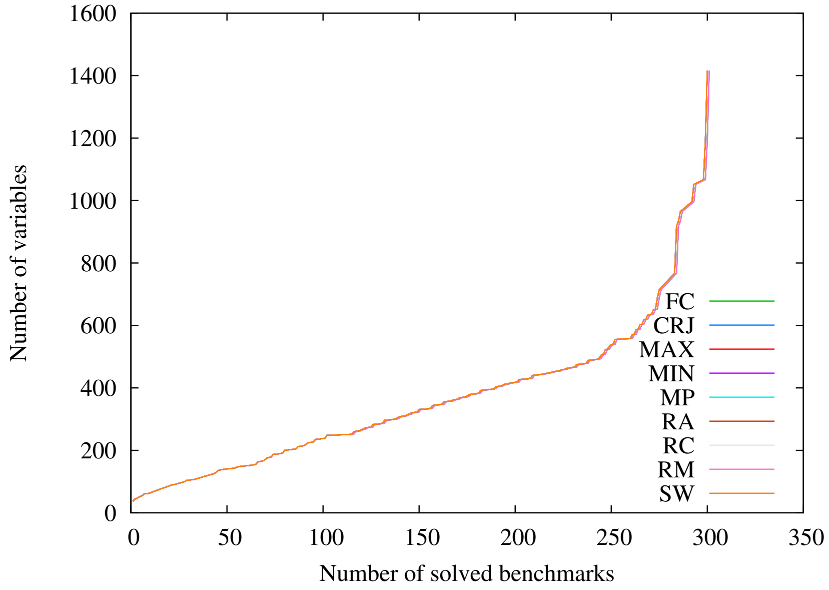


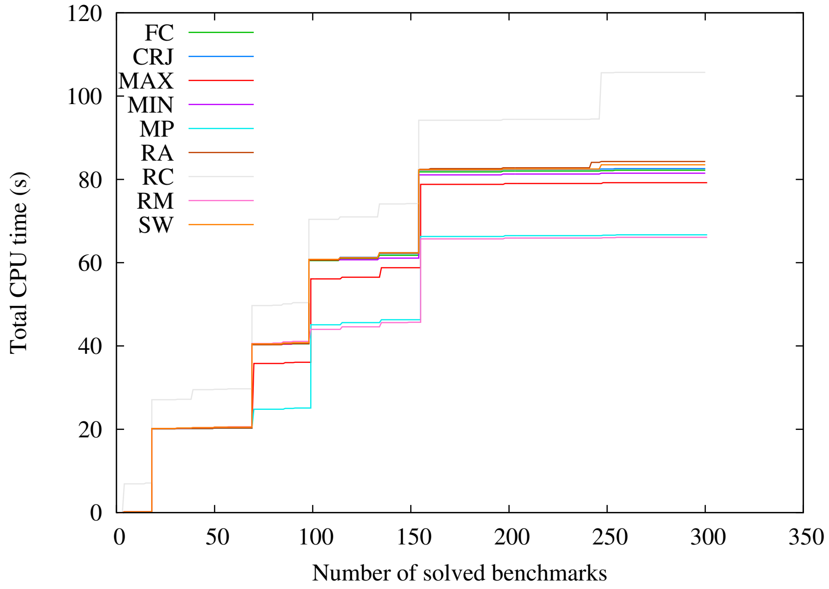
Real-Life Benchmarks (The First Set)



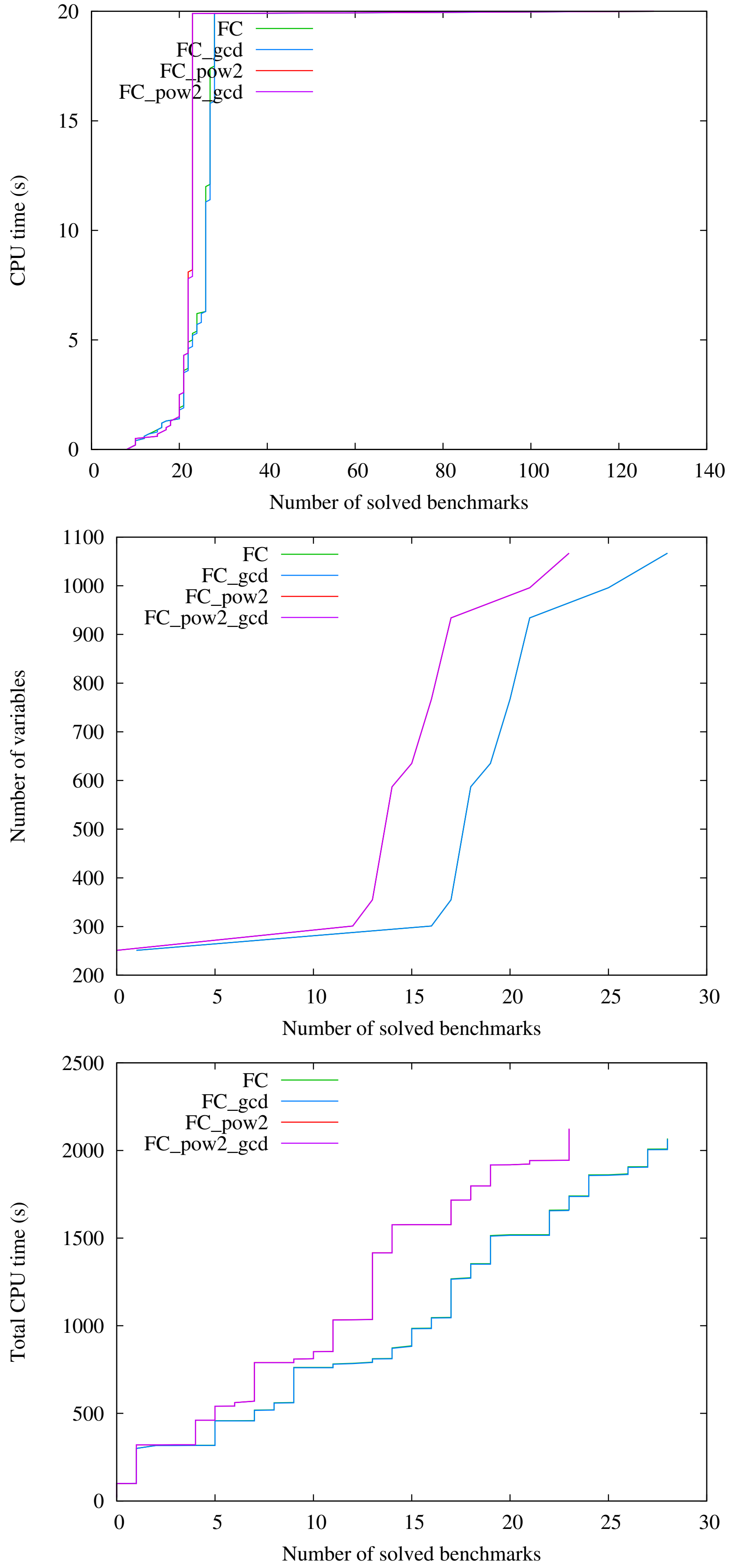


Real-Life Benchmarks (The First Set)

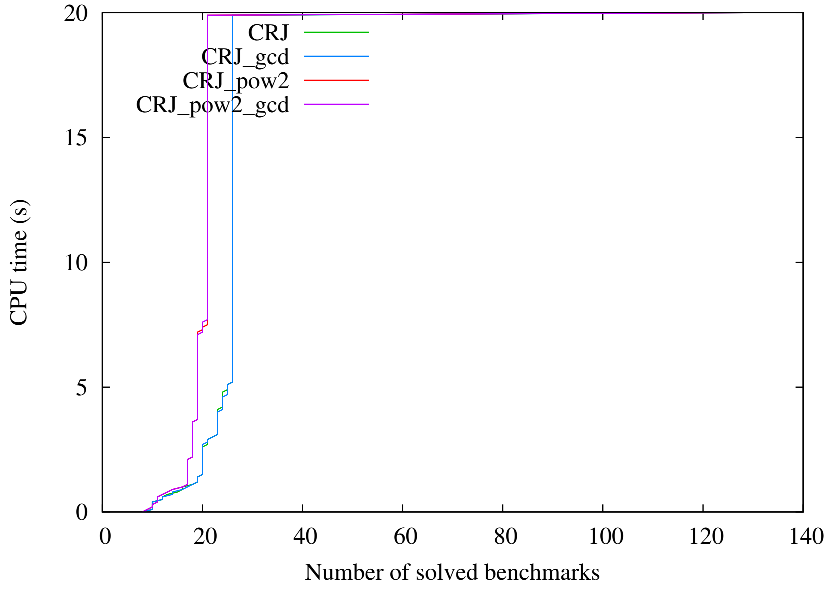


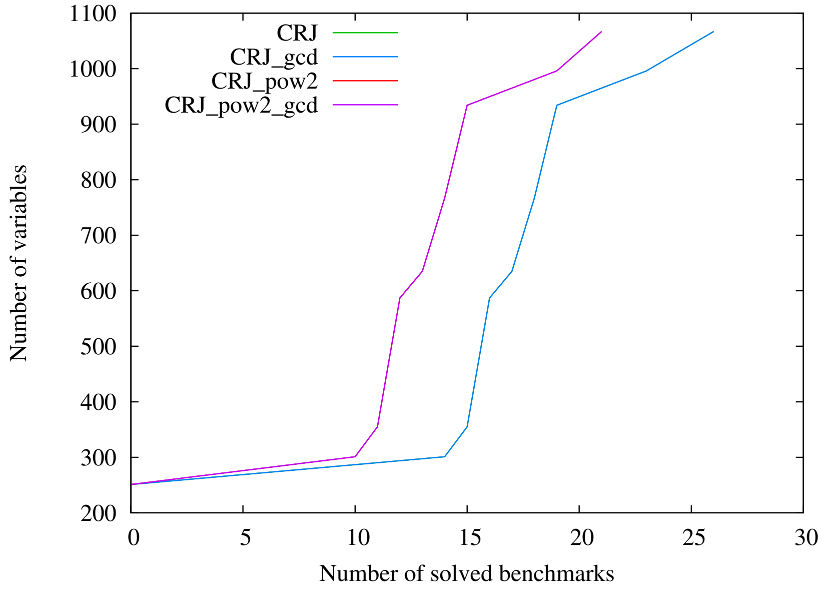


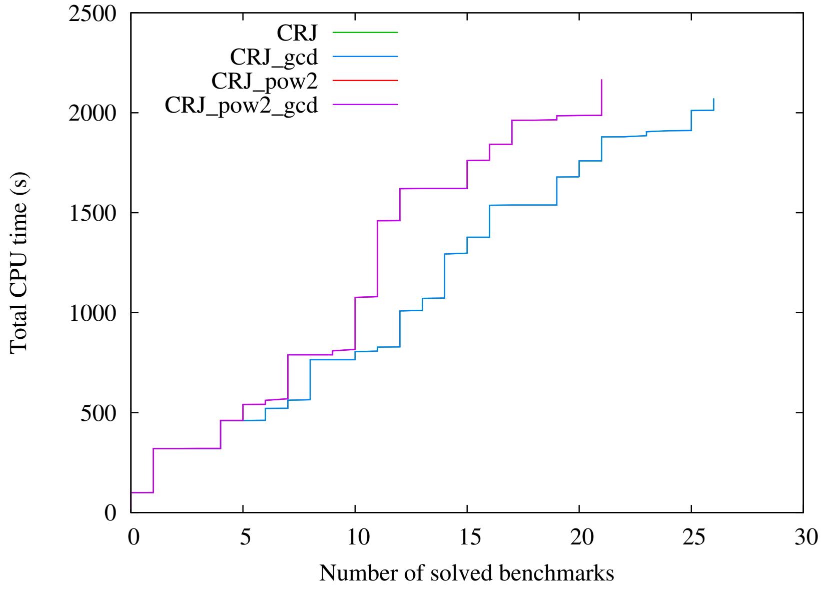
Real-Life Benchmarks (The Second Set)



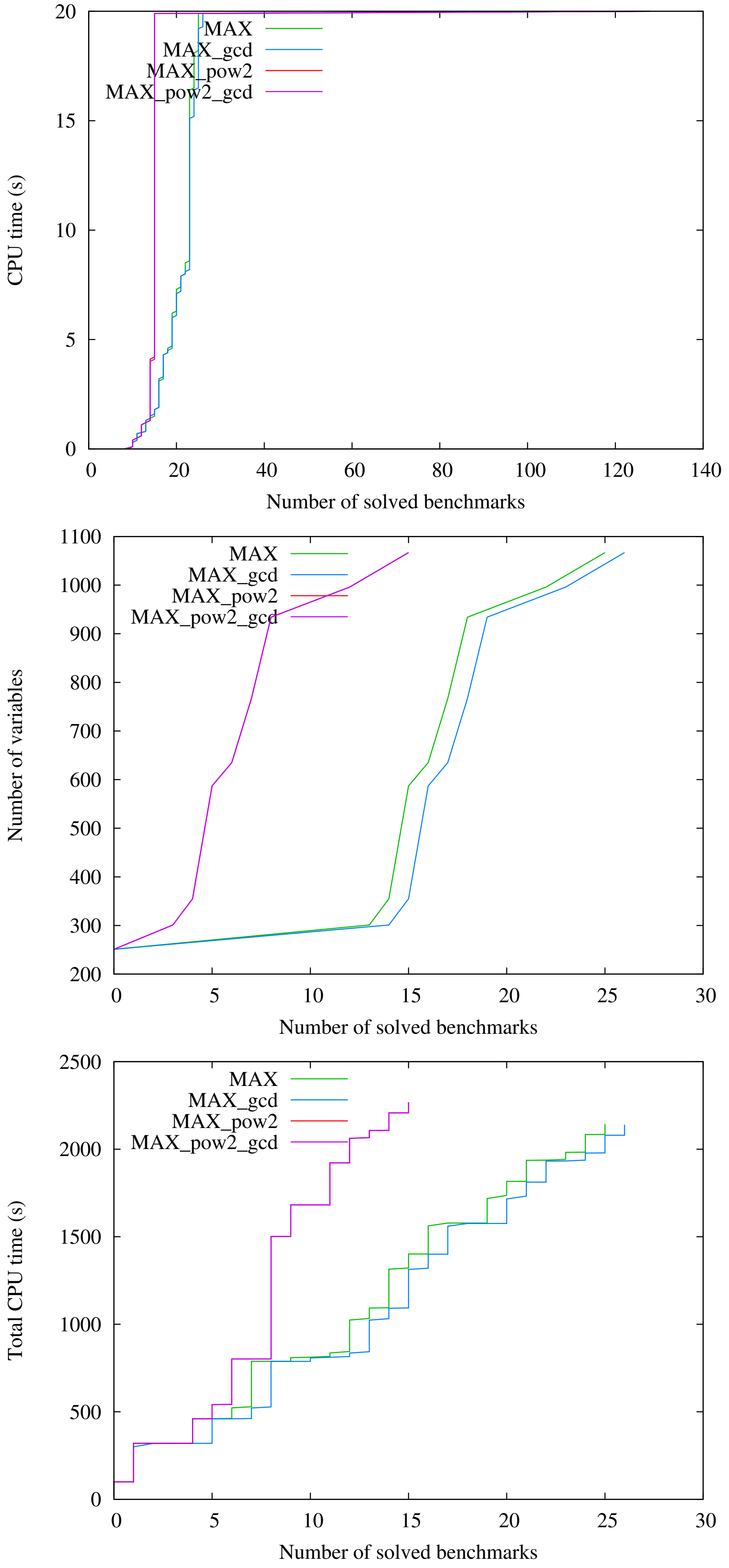
Real-Life Benchmarks (The Second Set)



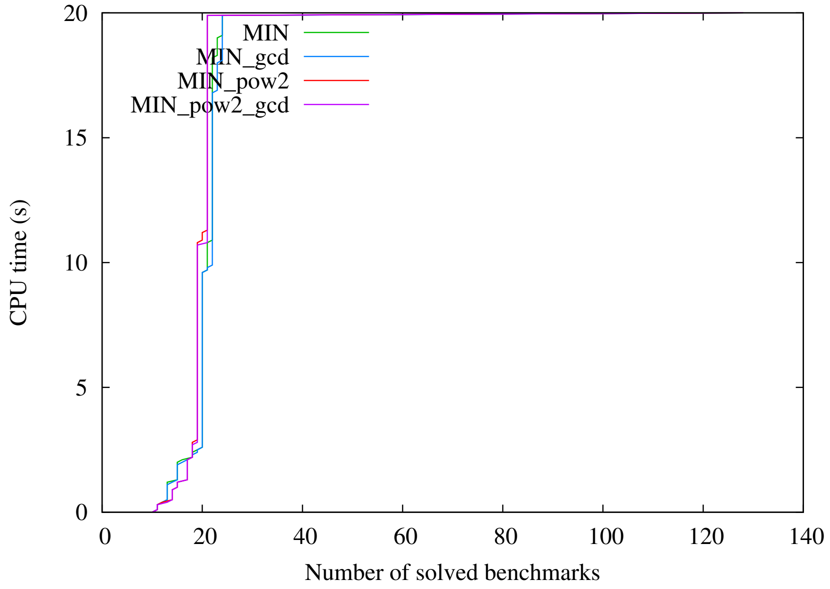


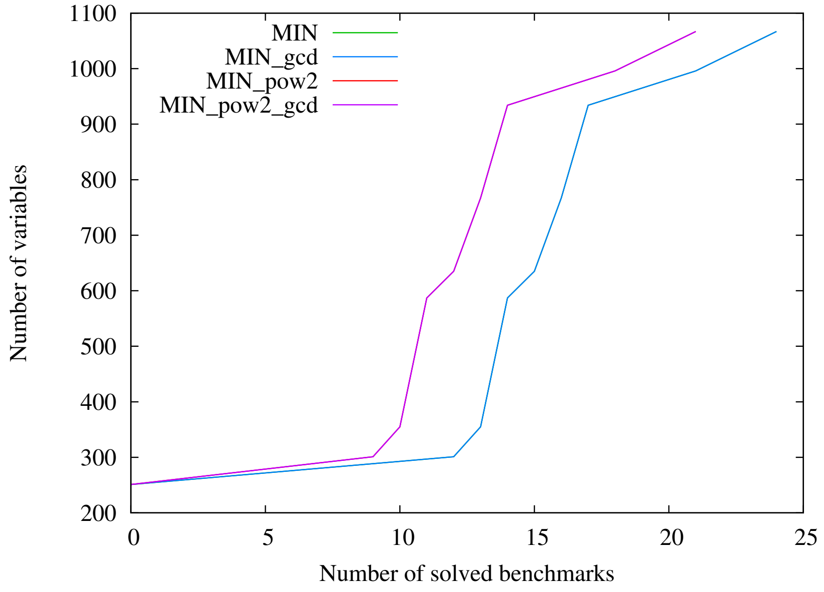


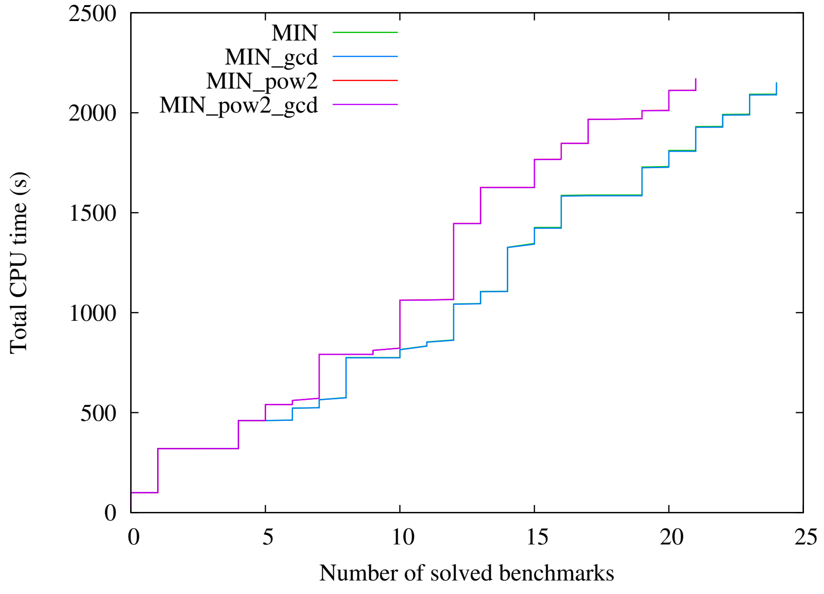
Real-Life Benchmarks (The Second Set)



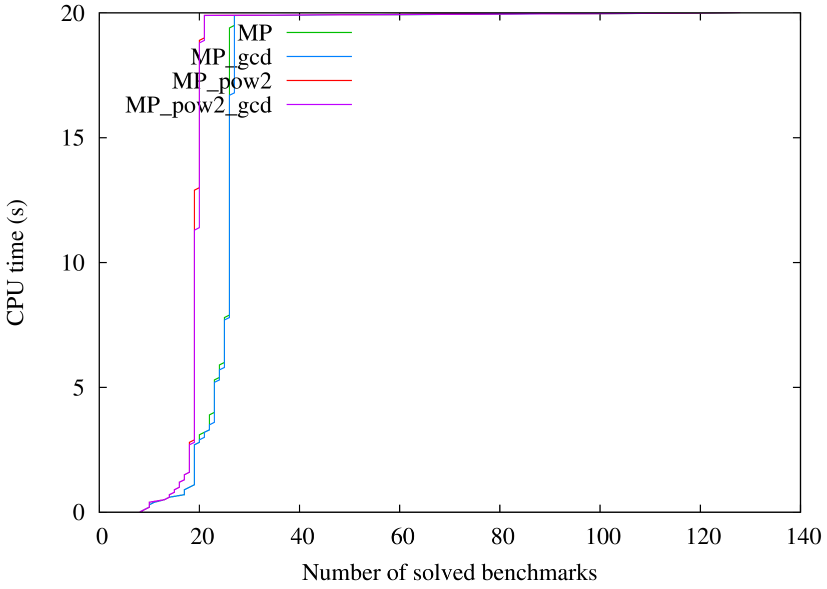
Real-Life Benchmarks (The Second Set)

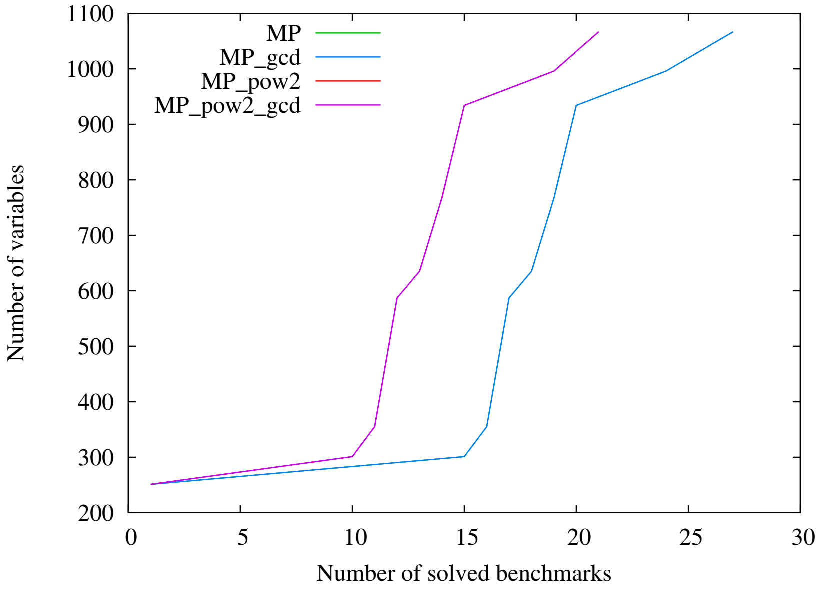


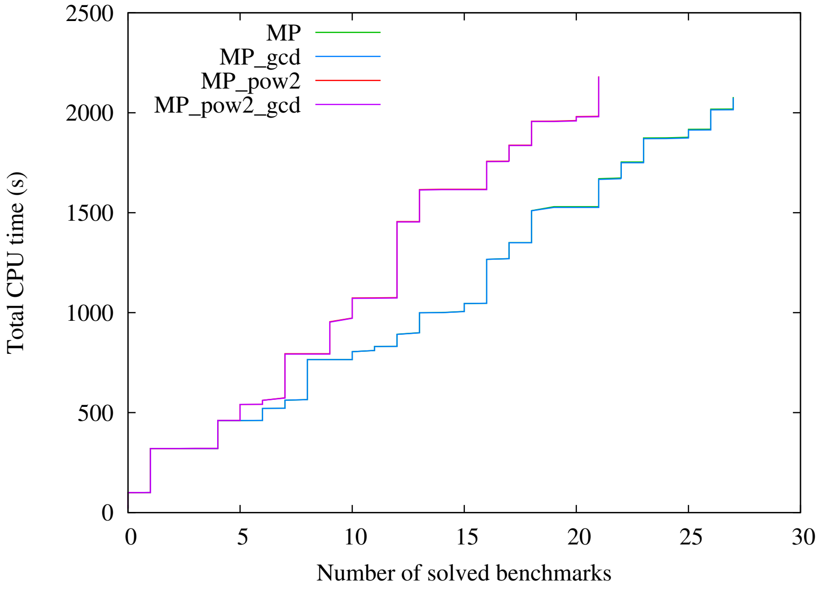




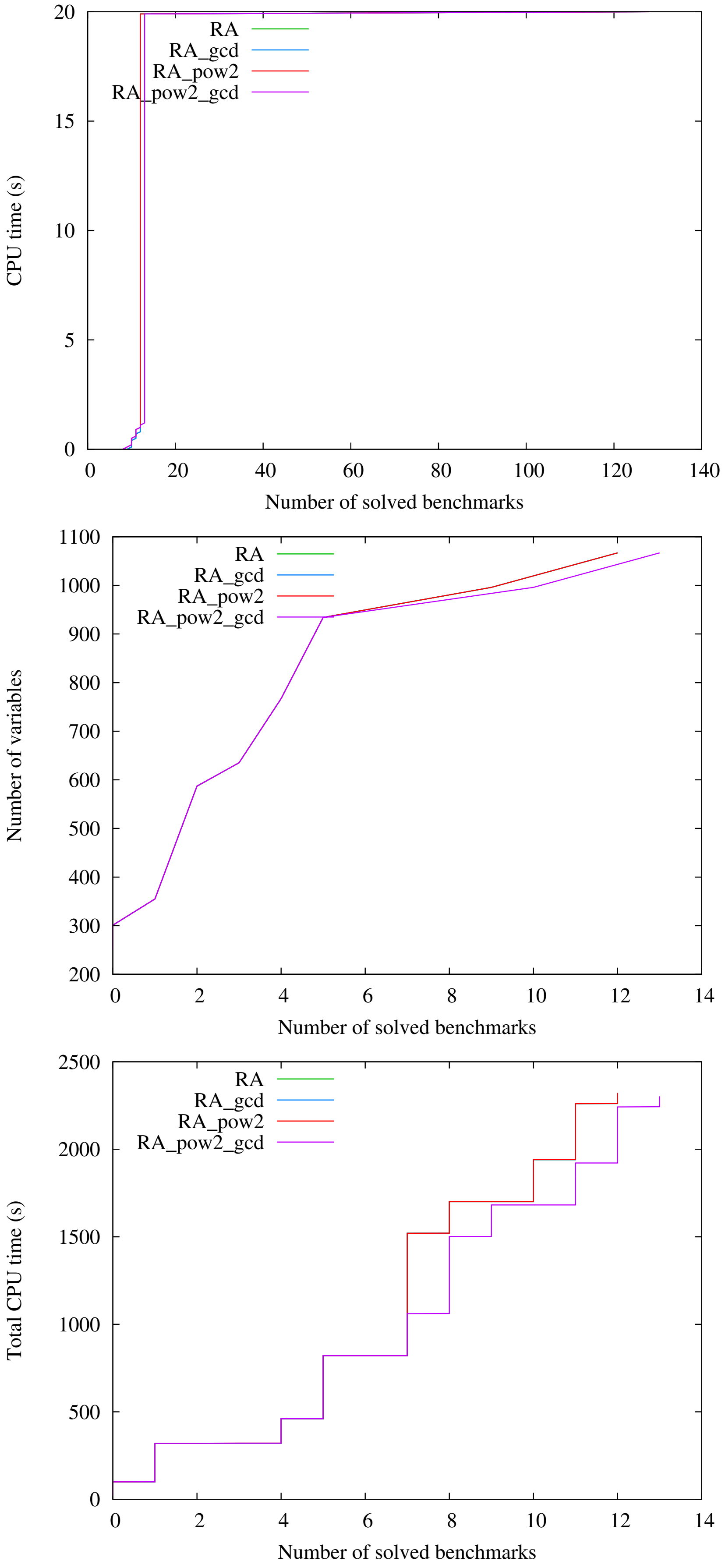
Real-Life Benchmarks (The Second Set)



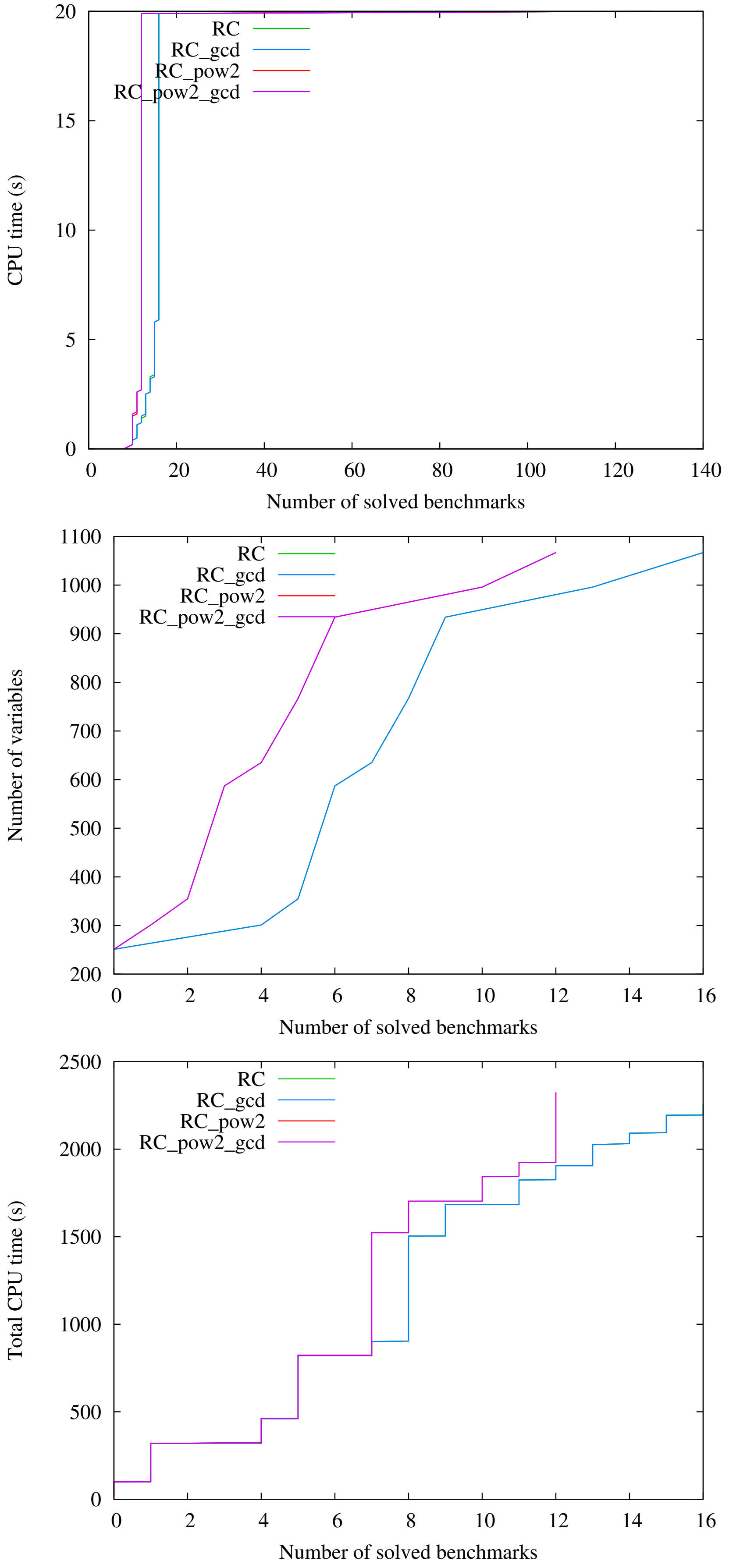




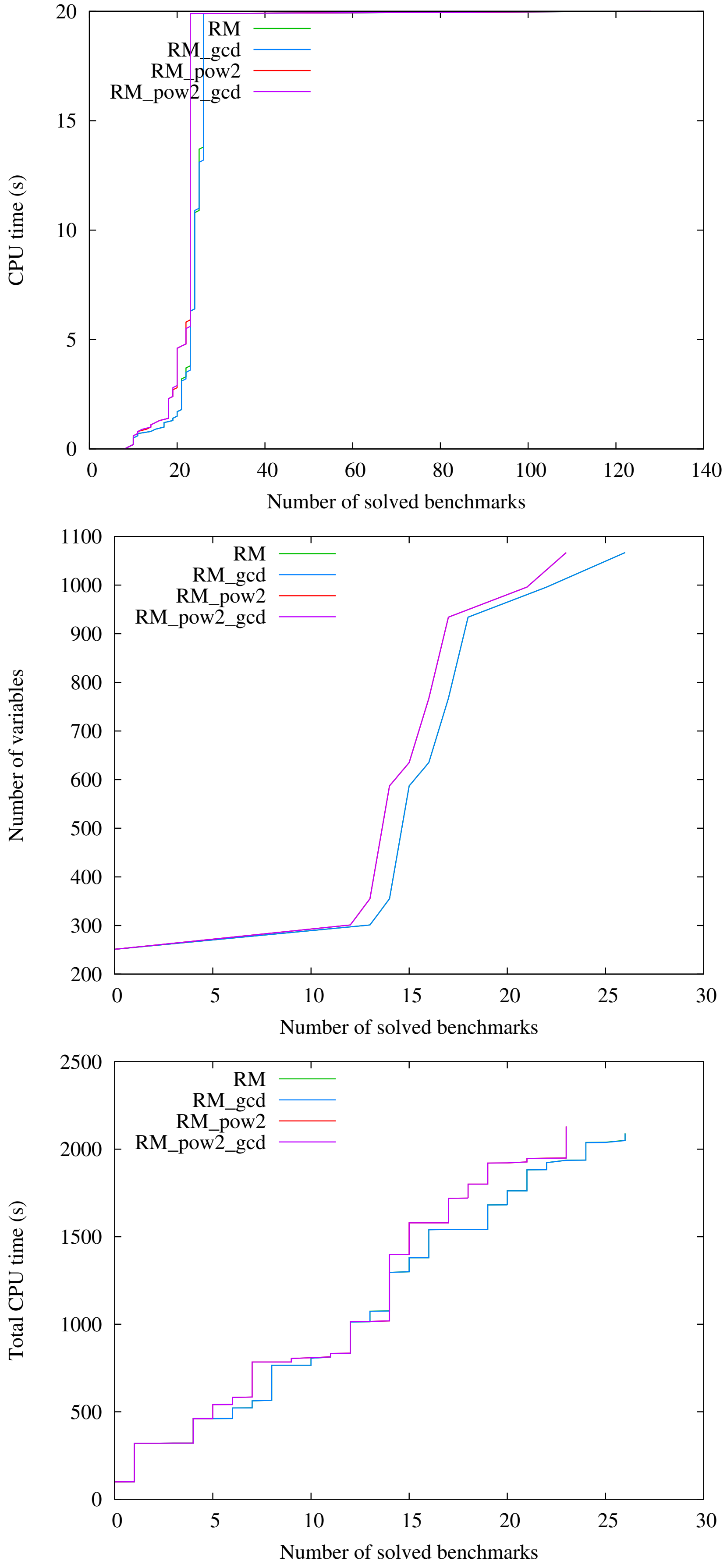
Real-Life Benchmarks (The Second Set)



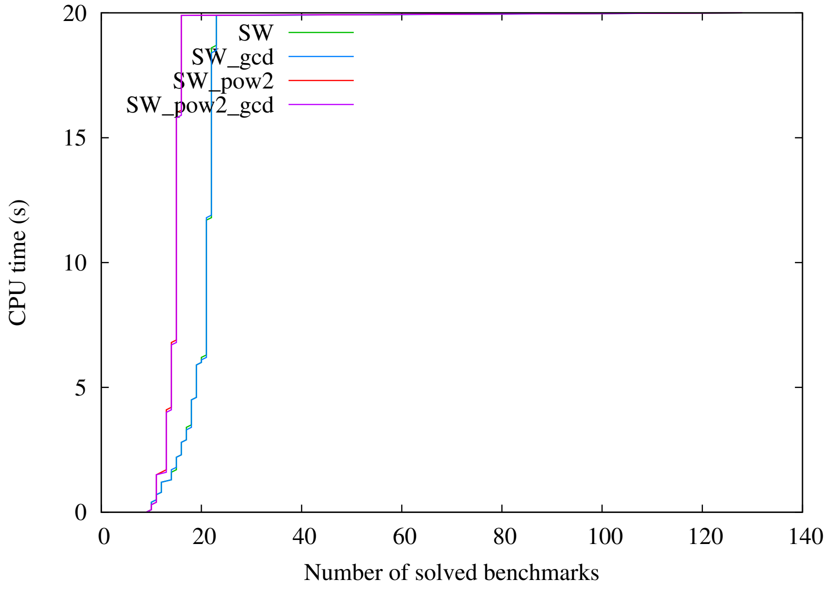
Real-Life Benchmarks (The Second Set)

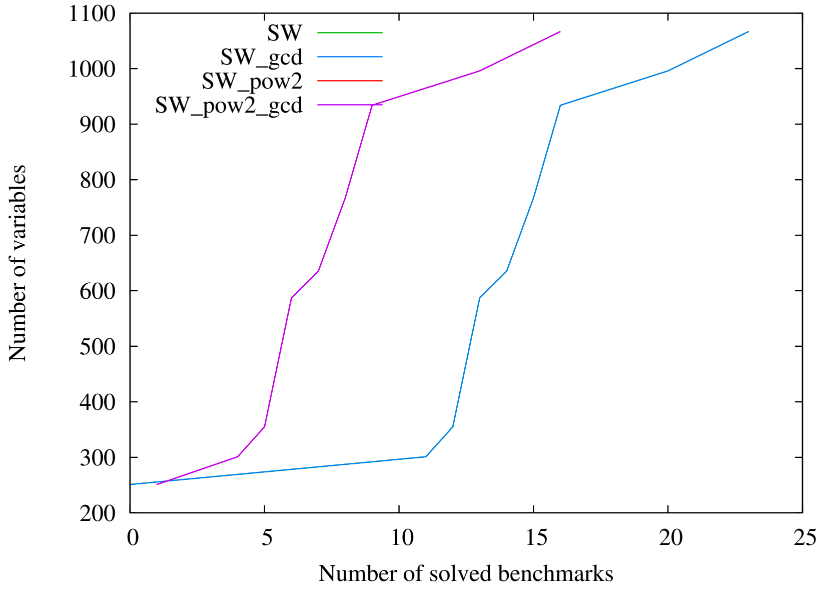


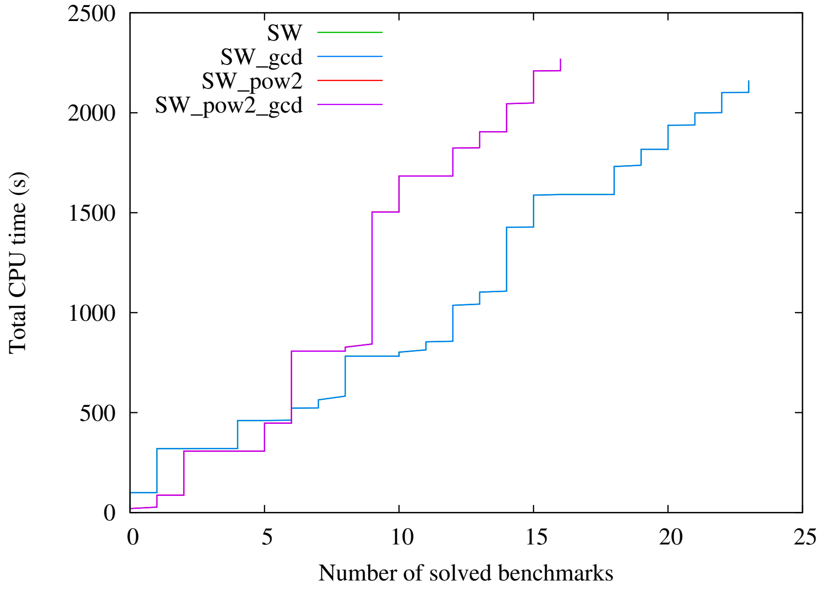
Real-Life Benchmarks (The Second Set)



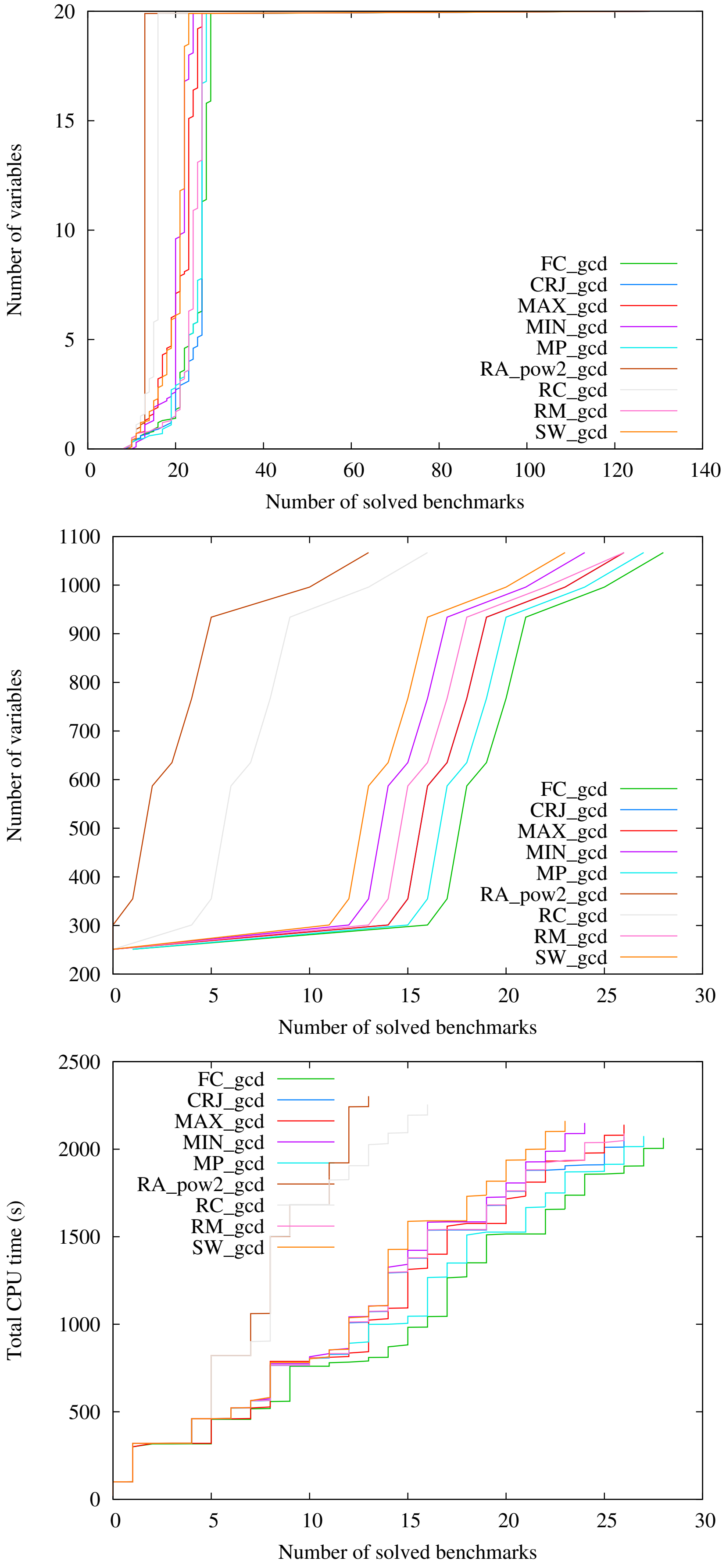
Real-Life Benchmarks (The Second Set)



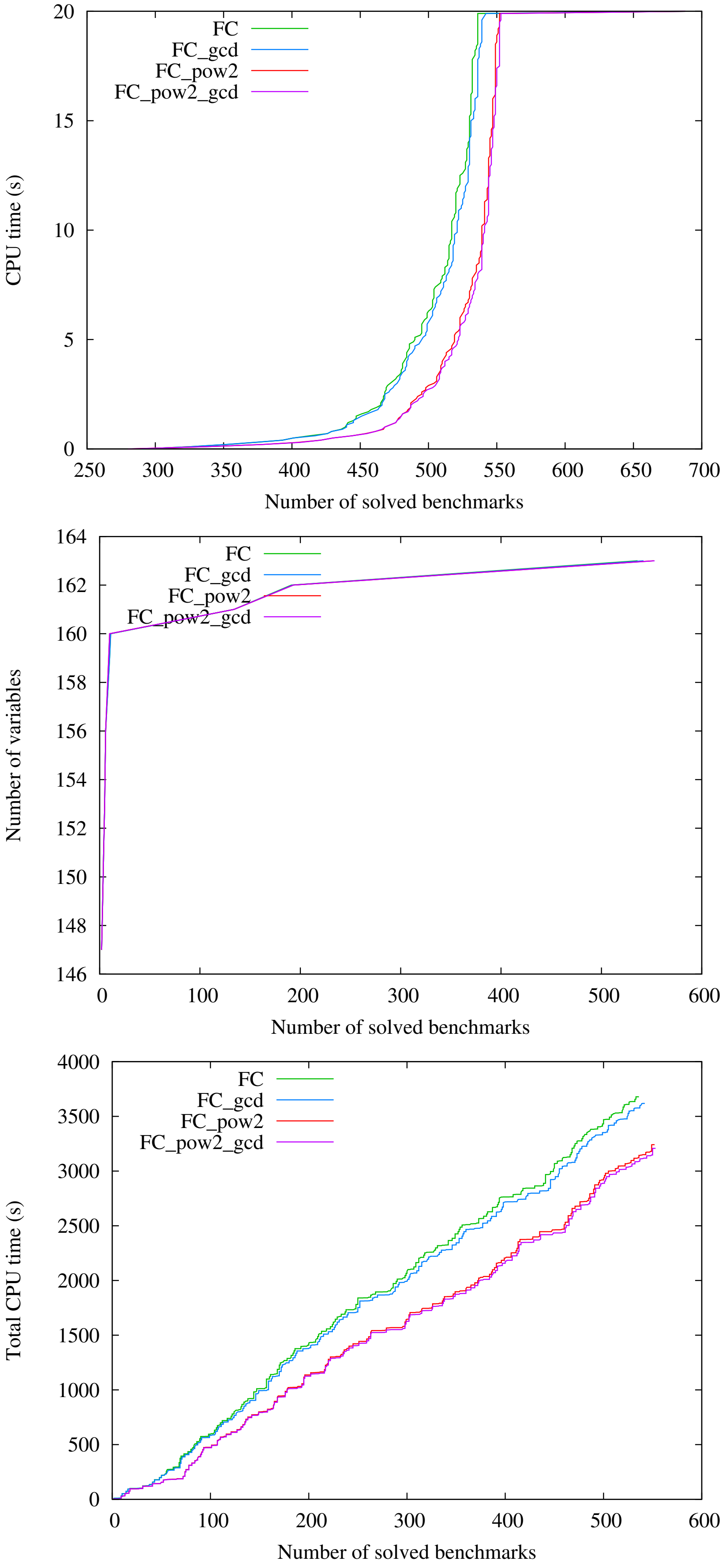




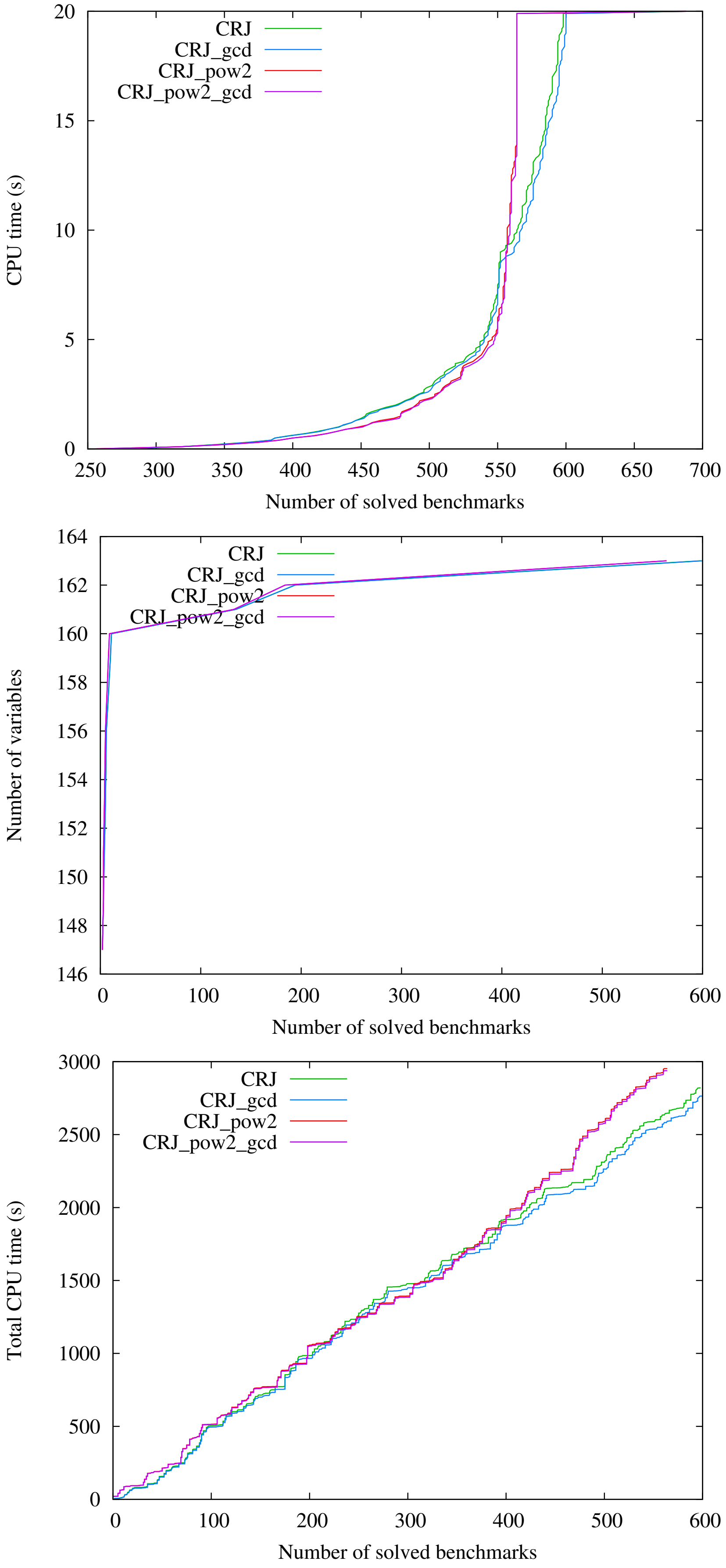
Real-Life Benchmarks (The Second Set)



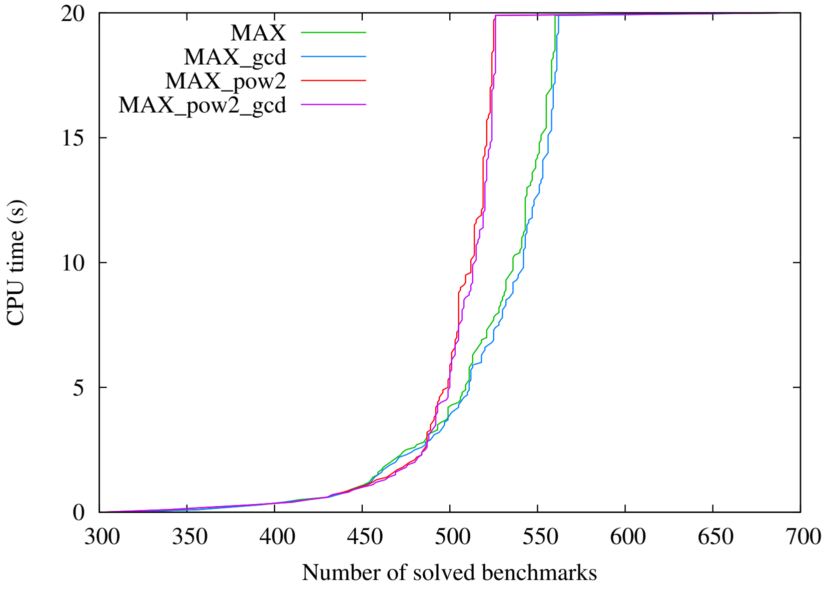
Real-Life Benchmarks (The Third Set By L. de Moura)

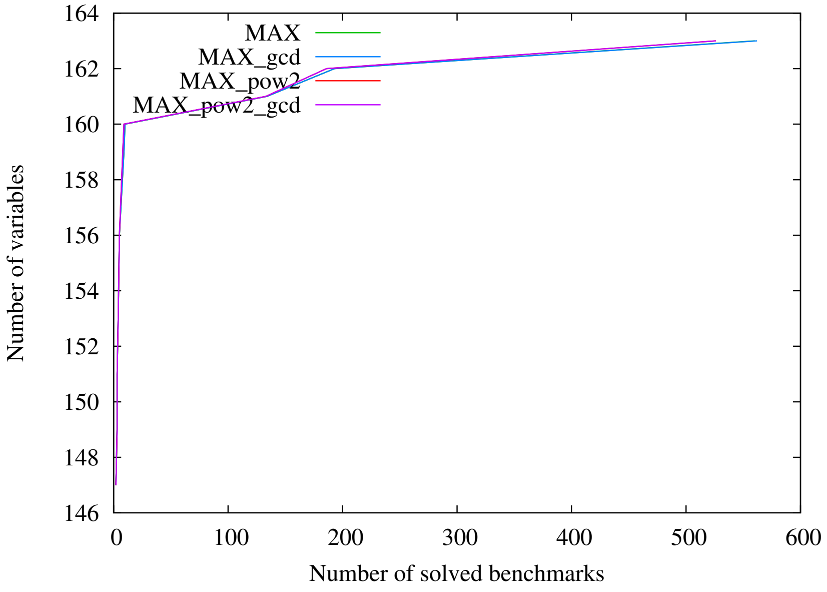


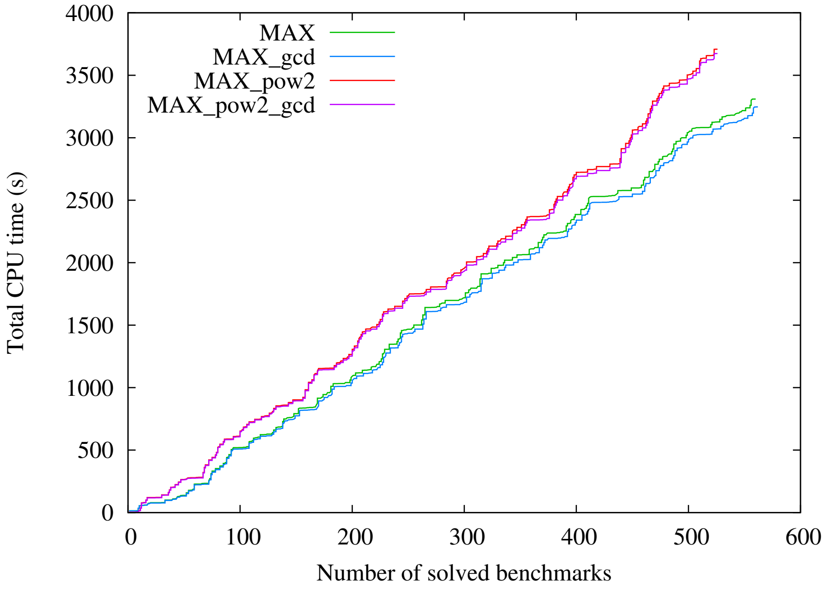
Real-Life Benchmarks (The Third Set By L. de Moura)



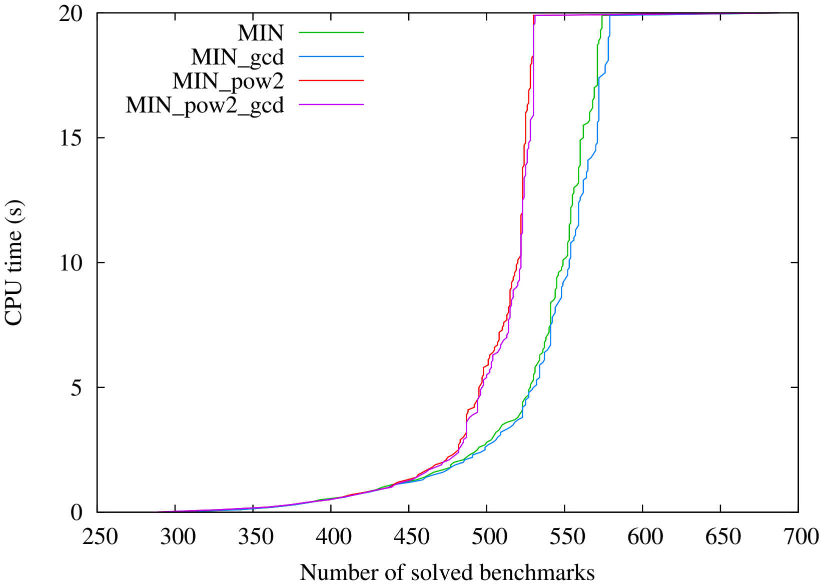
Real-Life Benchmarks (The Third Set By L. de Moura)

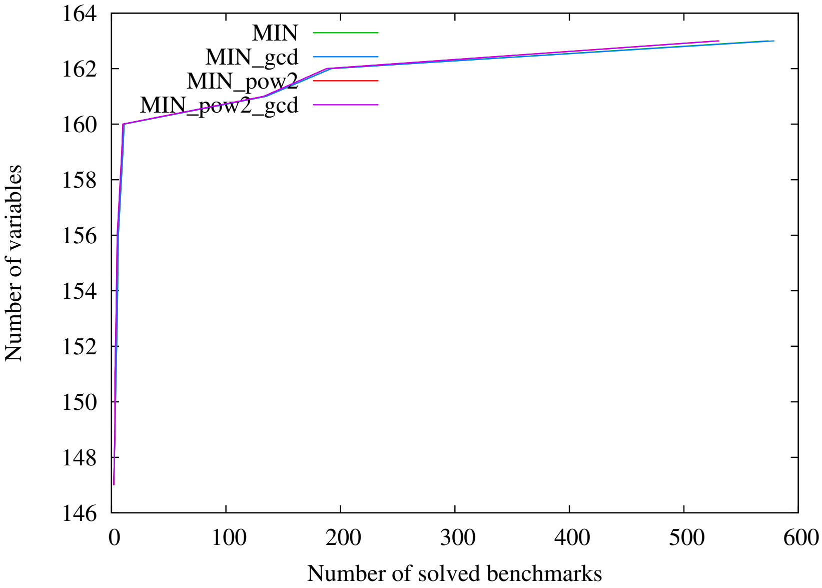


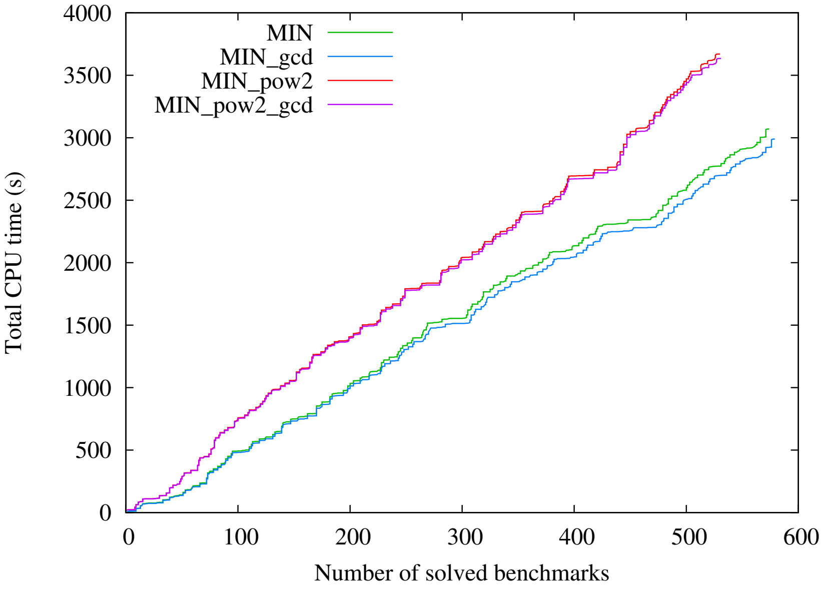




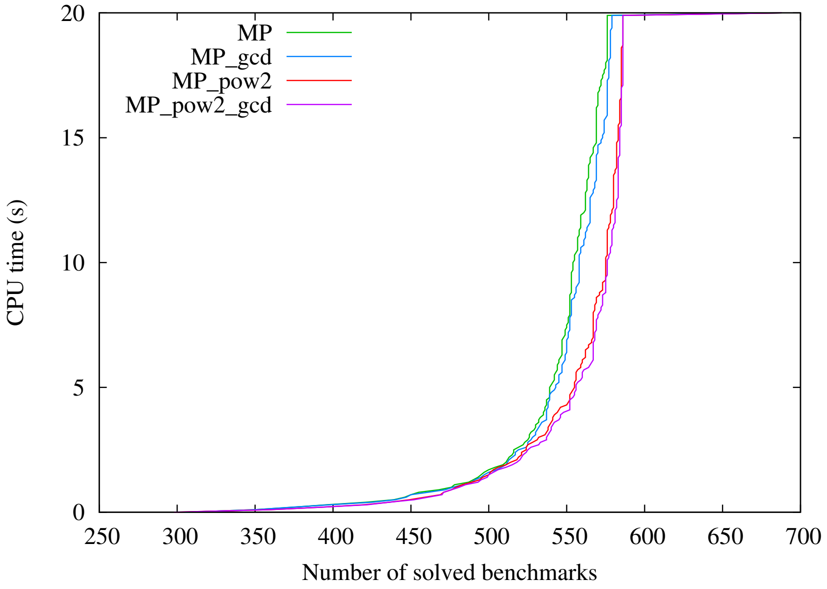
Real-Life Benchmarks (The Third Set By L. de Moura)

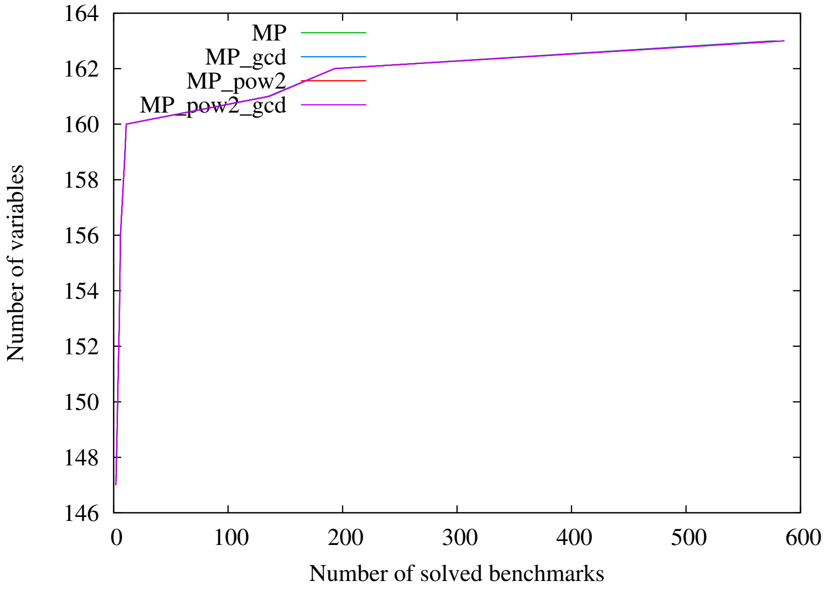


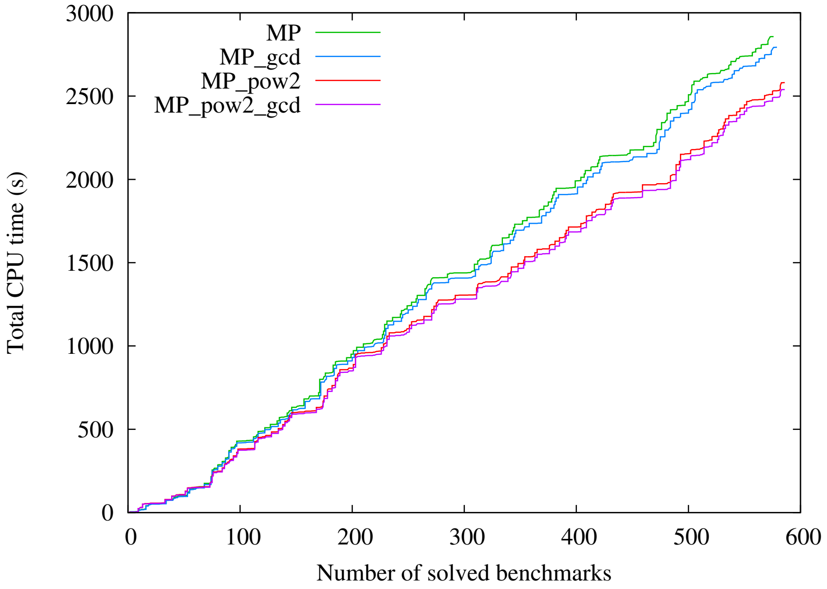




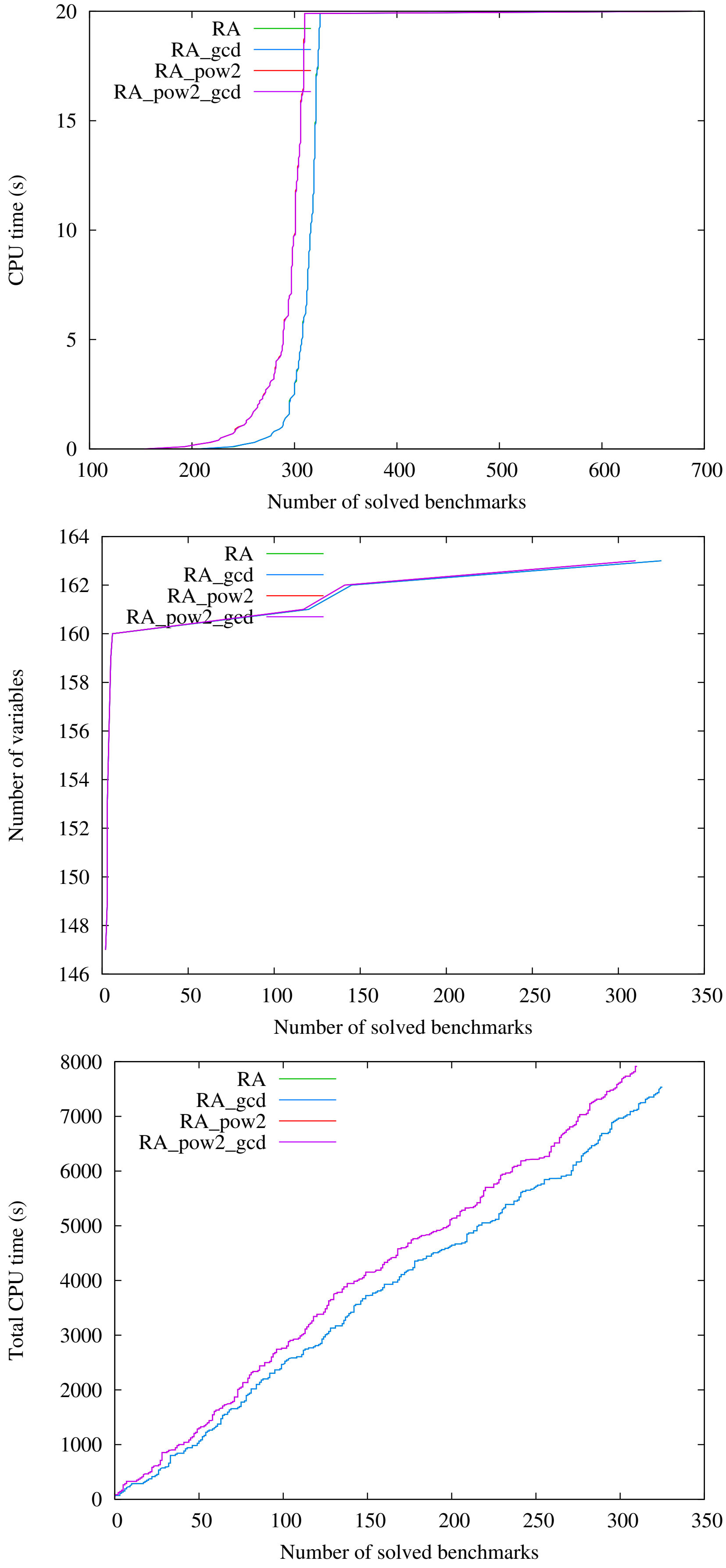
Real-Life Benchmarks (The Third Set By L. de Moura)



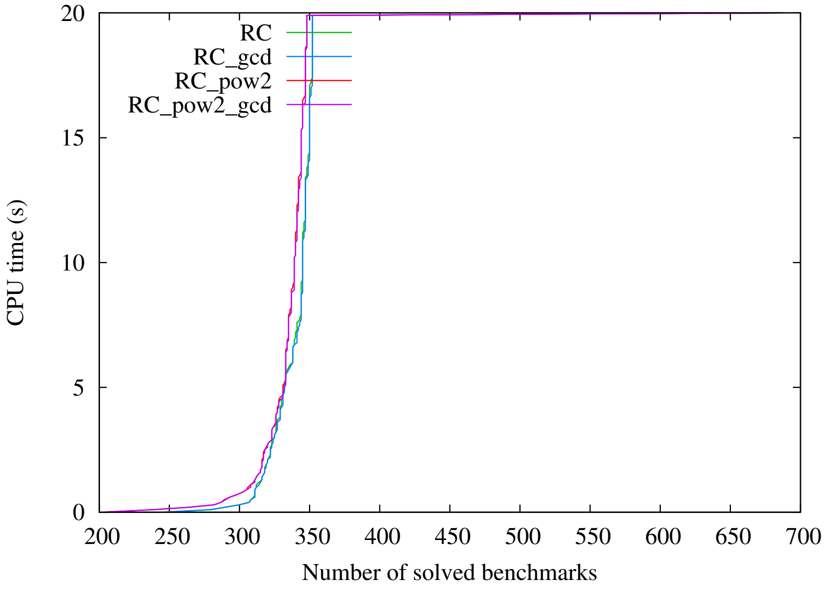


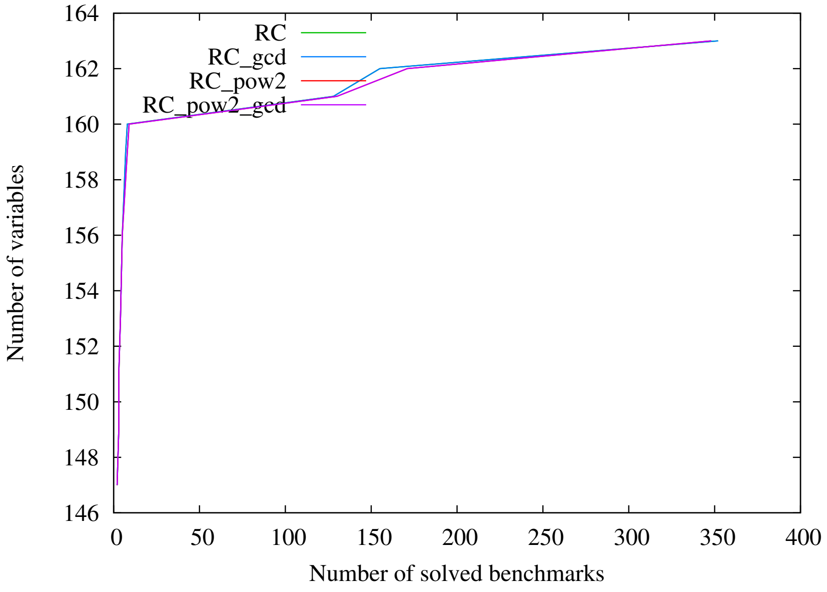


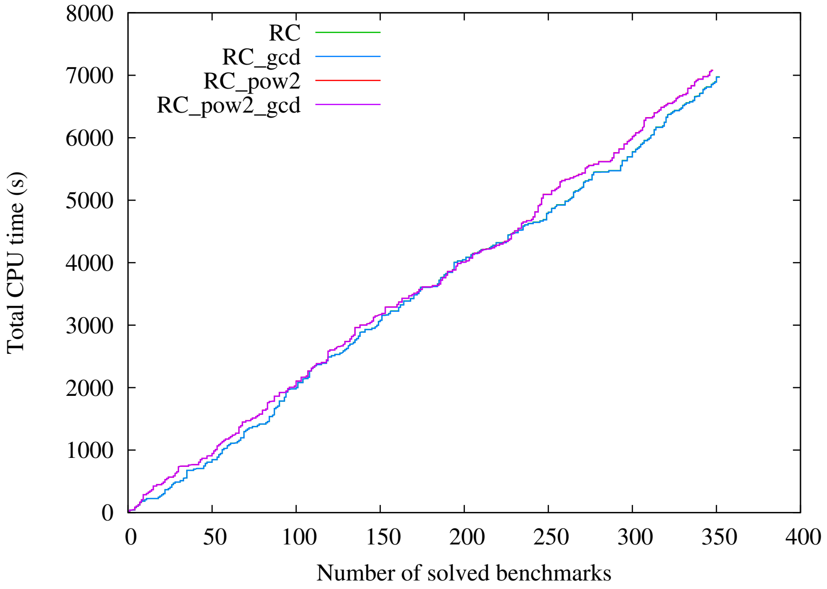
Real-Life Benchmarks (The Third Set By L. de Moura)



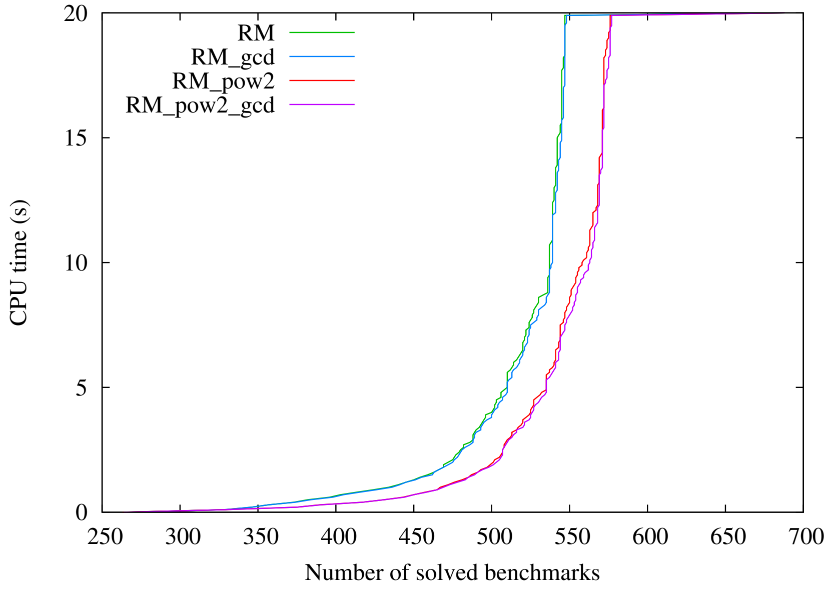
Real-Life Benchmarks (The Third Set By L. de Moura)

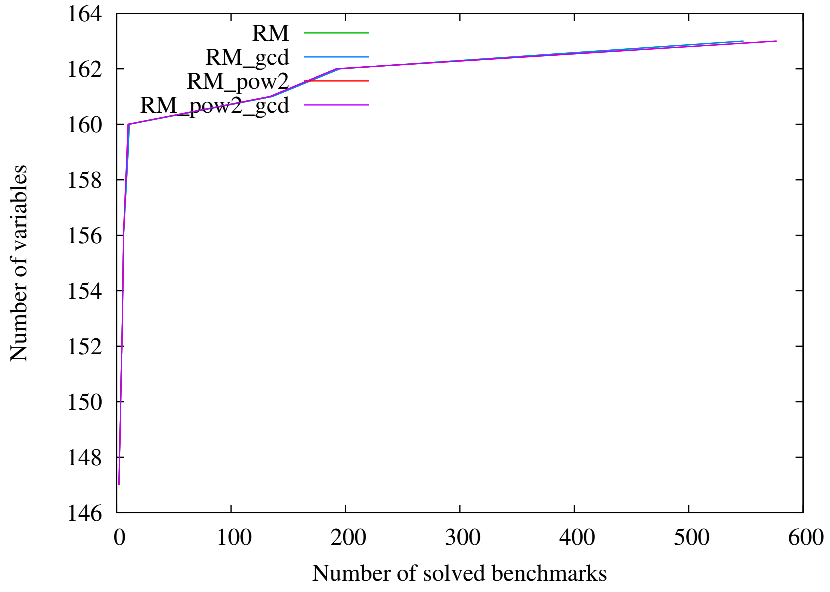


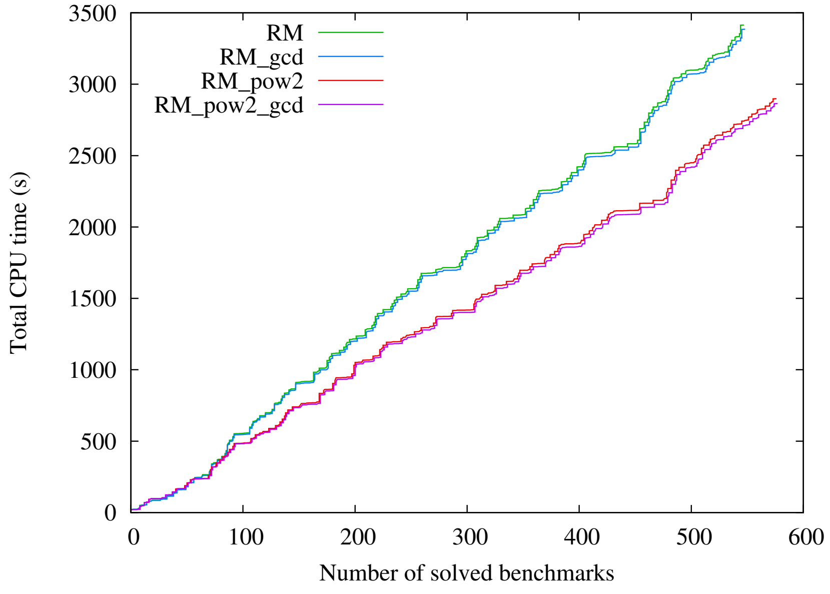




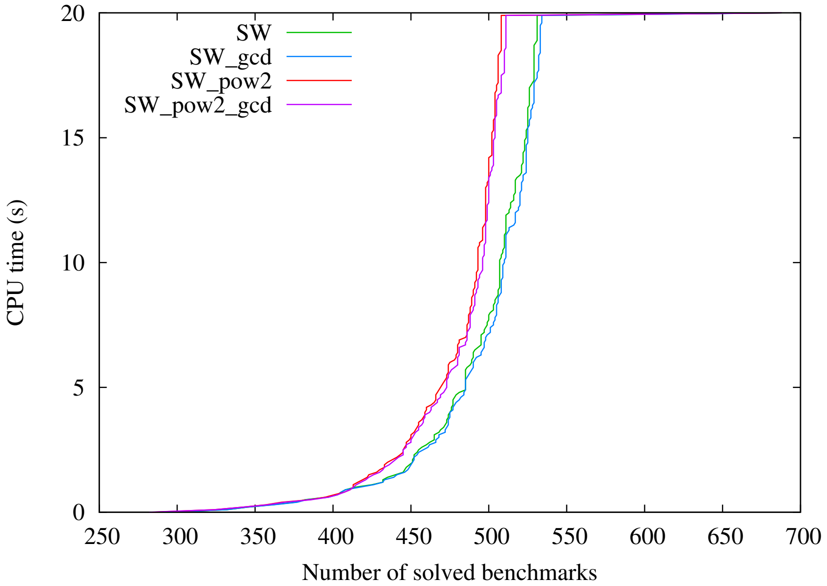
Real-Life Benchmarks (The Third Set By L. de Moura)

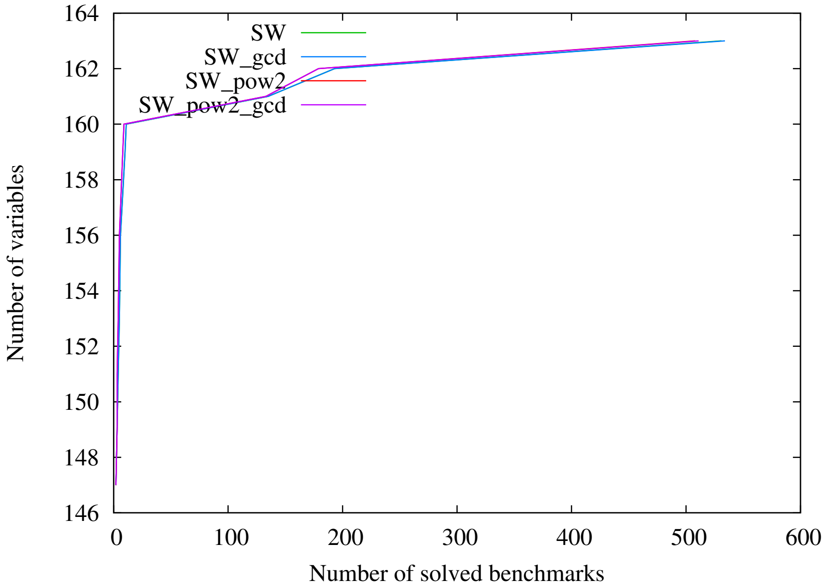


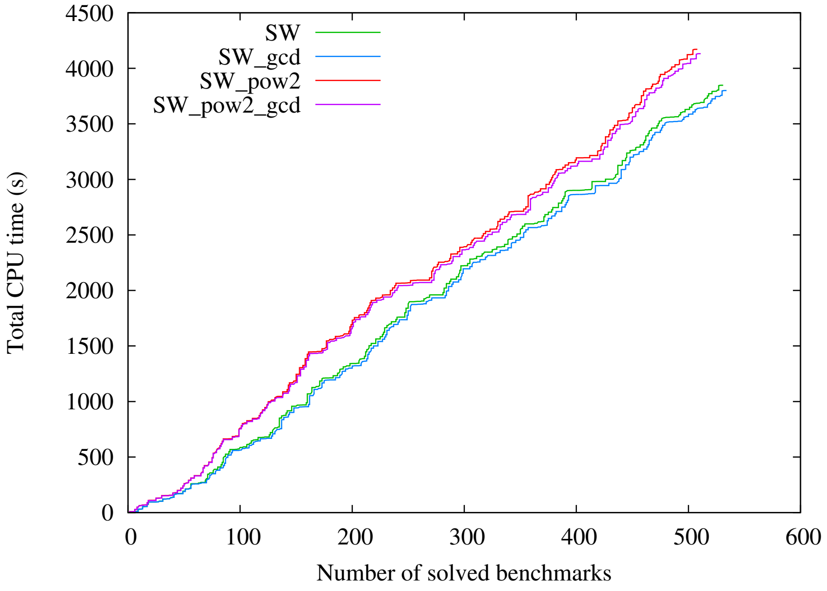




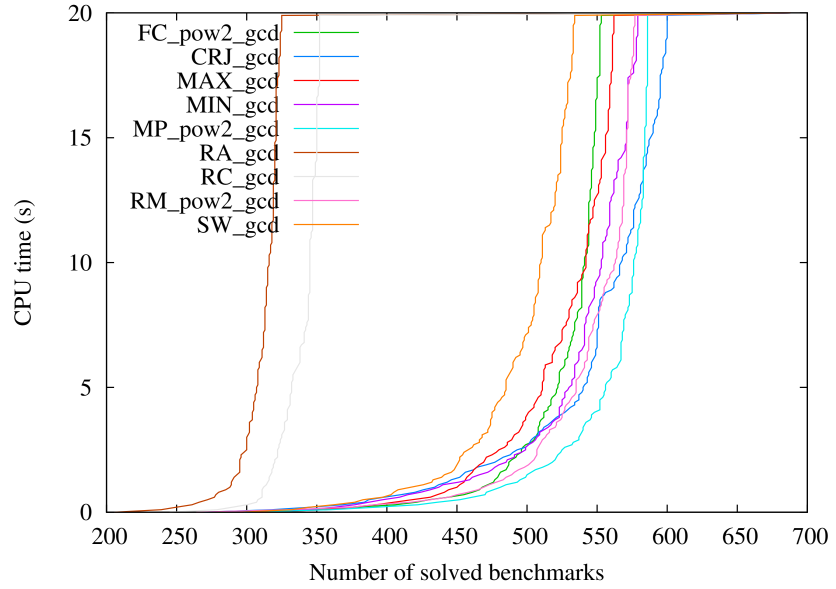
Real-Life Benchmarks (The Third Set By L. de Moura)

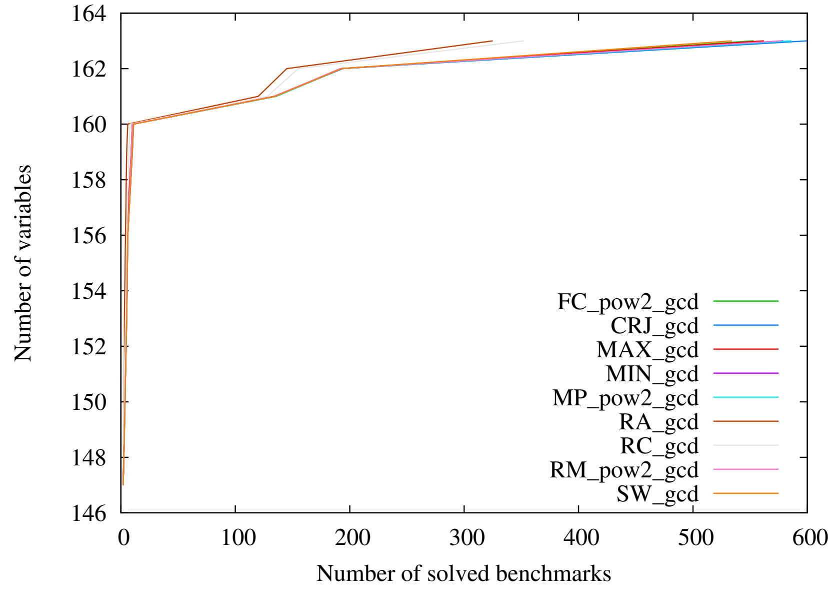


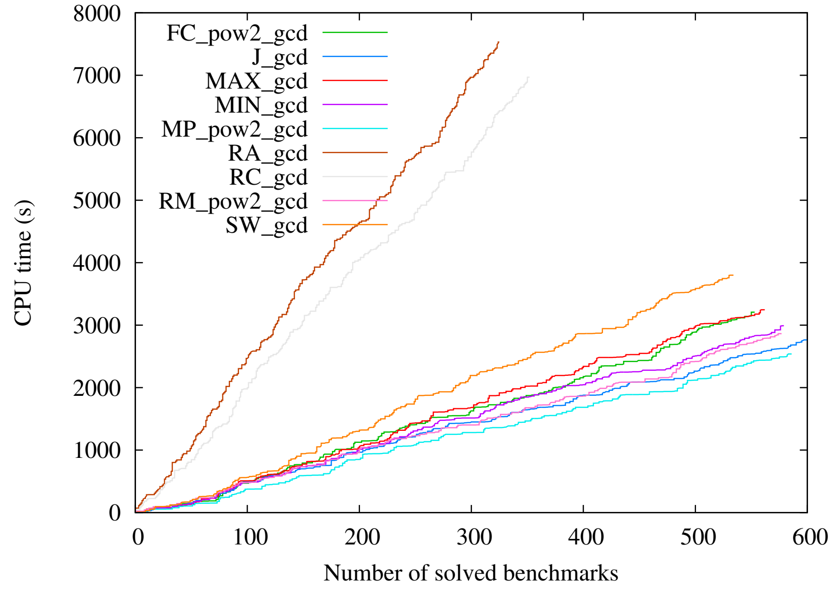




Real-Life Benchmarks (The Third Set By L. de Moura)







Index

$S_{<k}$ - subset of constraints, 35
$S_{=k}$ - subset of constraints, 35
$I(S, \sigma, k)$ - interval, 36
$L(S, \sigma, k)$ - lower bound, 36
σ_x^v - update of assignment, 24
$\mathbb{S}$ - state of the system, 36
$\succ$ - order on variables, **23**, 35
$U(S, \sigma, k)$ - upper bound, 36
k-conflict, *see* conflict

σ - assignment on variables, **24**

Agmon S., 33
AR rule, *see* assignment refinement rule (AR)
assignment refinement rule (AR), **37**, 38, 55, 63, 82

Barcelogic, 70, 73, 75, 93–95
benchmarks
  randomly generated benchmarks, 69, **70**, 73, 79, 89, 91
  real-life benchmarks, 69, **71**, 76, 85, 87, 89, 91, 92
bundles of heuristics, 79, 91

Chernikov algorithm, 21, 31, **31**, 70, 73, 74, 99
  redundancy criteria, 31
Chernikov S. N., 31
conflict, **36**
conflict resolution method, 20, 21, 34, **35**, 35–42, 48, 51, 65, 69, 94, 95, 97
  $I(S, \sigma, k)$ - interval, 36
  $L(S, \sigma, k)$ - lower bound, 36
  $U(S, \sigma, k)$ - upper bound, 36
  algorithm, 39
  conflict, **36**
  correctness and termination, 41
  extensions, 45–47
  half-bounded intervals, 36
  half-bounding levels, 36
  implementation, 59
  inference, 37
  level of constraints, **35**
  non-redundant constraints “almost”, 45
  order on variables, **55**, 55–59
  properties, **42**, 42–45
conflict resolution rule (CR), **37**, 55, 82
constraint
  implementation, 52
  most violated constraint, 61
  redundant, 31
CR rule, *see* conflict resolution rule
CVC3, 70, 73, 74, 93–95

Dantzig G. B., 26, 28, 32
de Moura L., 70
Dines L. L., 26

DPLL(T), 49

ellipsoid method, 27

Fourier-Motzkin method, 20, 21, 26, 28, 29, **29**, 29–31, 47, 48, 69, 70, 73, 74, 98
- geometric interpretation, 30
- modifications, 31

Fourier J. B. J., 26, **29**, 98

GMP library, 51
GoRRiLA - tool, 69, **71**

half-bounded intervals, 63, 85, 92
Hard Reality Tool (HRT), 70, **71**
heuristics
- general heuristics, 78
- major heuristics, 78

HRT
- see Hard Reality Tool, 70

hyperplane, **25**, 34

implementation, 51
- data structure, 51
- input constraints, 53
- levels, 54
- parser, 51
- Phase I, *see* Phase I
- Phase III, *see* Phase III
- preprocessing, 65
  - almost half-bounding levels, 66
  - half-bounding levels, 65
- quadruple, *see* quadruple
- sort input constraints, 51
- trace variables, 53

incrementality, 21, **49**, 50, 99
interior point method, 20, **27**, 28, 98

Kantorovich L. V., 26
Karmarkar N., 27
Khachiyan L. G., 27
Koopmans Tj. C., 26
Korovin K., 70

level
- half-bounding level, 36
- implementation, 54
- of a constraint, **35**

linear constraint, **23**
- level of constraints, **35**
- normalised linear constraint, 24
- solution of a linear constraint, 24

linear programming, 20, **25**, 26, 32, 47

Mehrota S., 27
method
- barrier function method, 27
- Chernikov algorithm, 21, 31, **31**, 70, 73, 74, 99
- conflict resolution method, 20, 21, 34, **35**, 35–42, 48, 65, 69, 94, 95, 97
- ellipsoid method, 27
- Fourier-Motzkin method, 20, 21, 26, 28, 29, **29**, 29–31, 47, 48, 69, 70, 73, 98
- interior point method, 20, **27**, 28, 98
- path-following method, 27
- relaxation method, 61
- simplex method, 20, 21, 26–28, **32**, 32–33, 69, 70, 75, 98, 99

Motzkin T. S., **26**, 33

Nemirovskii A. S., 27

Nocedal J., 27
non-redundant constraints "almost", 45

order on variables, **55**, 55–59

performance
  equal and nearly equal performances, 82, 85
  the best performance in quadruples, 84, 87
Phase I, 69
  experiments, 73
  summary of experiments, 76
Phase II, 69
  summary of experiments, 91
  experiments, 76
Phase III, 69
  experiments, 92
  summary of experiments, 94
polyhedron, **25**, 30, 32, 34
preprocessing
  almost half-bounding levels, 66
  experiments, 92
  half-bounding levels, 65

quadruple, **78**, 82, 85

real-life benchmarks, 75
redundancy
  see redundant constraints, 47
redundant constraints, 31, 47
relaxation method, 61
  geometric interpretation, 33

Satisfiability Modulo Theories (SMT), 19, **49**, 49–51, 69, 73, 99
  SMT-LIB, 51, 69, 71, 79
    QF_LRA, 72
Schoenberg I. J., 33
Schrijver A., 28, 30
Shor N. Z., 27
simplex method, 20, 21, 26–28, 32, **32**, 32–33, 69, 70, 75, 98, 99
  geometric interpretation, 32
strategies
  adding resolvents, 63
  closest binary, *see* middle point
  for adding resolvents, 77
  for choosing conflicts, 77
  for selecting assignment values, 62, 77
    interleaved points, 62
    maximal (minimal) point, 62
    middle point, 62, 93
    random choice, 62
  for selecting conflicts, 61
    algebraic approach, *see* maximal overlap strategy
    geometric approach, 61
    maximal overlap strategy, 61, 93
    random choice approach, 62
    take the first, 62
  for setting order on variables, 65
  half-bounded intervals, 64, 77
  maximal overlap strategy, 82
  reducing constraints by GCD, 64, 78
system of linear constraints, 20, **23**, 29, 35, 40
  application, 28
  geometric interpretation, 25
  solution of a system, 24
  solving, 20, **25**, 28, 32, 33, 97

unsatisfiability explanations, 21, **49**, 50, 99

value of the variable, 24
Vampire theorem prover, 51
von Neumann J., 26, **27**
Voronkov A., 51, 70

Wright S., 27

Yudin D. B., 27

Z3, 70, 93–95