Implementing Conflict Resolution

Konstantin Korovin *, Nestan Tsiskaridze, and Andrei Voronkov **

The University of Manchester
{korovin|tsiskarn|voronkov}@cs.man.ac.uk

Abstract. The *conflict resolution* method, introduced by the authors in [1] is a new method for solving systems of linear inequalities over the rational and real numbers. This paper investigates various heuristics for optimisation of the method and presents experimental evaluation. The method and heuristics are evaluated against various benchmarks and compared to other methods, such as the Fourier-Motzkin elimination method and the simplex method.

1 Introduction

In this paper we present an evaluation of our *conflict resolution* method [1] for checking solvability of systems of linear inequalities.

Conflict resolution is a solution driven method. Given a system of linear inequalities and an arbitrary initial assignment on variables, the method iteratively modifies the assignment aiming at obtaining a solution. During this process a conflict can arise when the current assignment cannot be directly modified to satisfy the system of inequalities. In this case there exists at least one pair of conflicting inequalities which impedes the refinement. Such a conflict is resolved by deriving a new inequality from the conflicting pair and adding it to the current system of inequalities. The process continues until either the assignment is refined into a solution, or a trivially unsatisfiable inequality is derived, showing unstaisfiability of the initial system of inequalities. In [1] the conflict resolution method is shown to be sound, complete and terminating.

The performance of the method can be improved by using various strategies for selecting conflicting pairs, refinement of assignments, and choosing the order on variables. In this paper we introduce a number of heuristics and strategies for the conflict resolution method. We evaluate them on various benchmarks and compare to other methods for solving systems of linear inequalities such as the Fourier-Motzkin elimination method and the simplex method.

This paper is structured as follows. In Section 2 we give some preliminary notations. Section 3 briefly overviews the conflict resolution method. In Section 4 we present heuristics whose performance is studied in the paper. Section 5 describes the set of benchmarks used for the experimental evaluation. In Section 6 we describe preprocessing methods used in the evaluation. . The results of experiments are discussed in Section 7. In Section 8 we summarise the presented work and discuss further research directions.

* Supported by a Royal Society University Research Fellowship
** Partially supported by an EPSRC grant

2 Preliminaries

Let $\mathbb{Q}$ denote the set of rationals, and X be a finite set of variables $\{x_1, \ldots, x_n\}$ where n is a positive integer. We call a *rational linear constraint over* X either a formula $a_n x_n + \ldots + a_1 x_1 + b \diamond 0$, where $\diamond \in \{\geq, >, =\}$ and $a_i \in \mathbb{Q}$ for $1 \leq i \leq n$, or one of the formulas $\bot, \top$. The formula $\bot$ is always false and $\top$ is always true. The constraints $\bot$ and $\top$ are called *trivial*.

We introduce an order on variables, without loss of generality we can assume: $x_n \succ x_{n-1} \succ \ldots \succ x_1$. For simplicity, we consider all constraints throughout the paper to be *normalised*, i.e. be of one of the forms: $\bot$, $\top$, $x_k + q \diamond 0$ or $-x_k + q \diamond 0$, where $\diamond \in \{\geq, >, =\}$, x_k is the maximal variable in the respective constraint, and q does not contain x_k. Evidently, every constraint can be effectively changed to an equivalent normalised constraint. We introduce a notion of the *level* of a constraint as follows: if the maximal variable in a constraint c is x_k, then we say that k is the *level* of c. If c contains no variables, then we define the level of c to be 0. Note that, since all constraints are assumed to be normalised, a constraint written in the form $x_k + p \geq 0$ or $-x_k + q \geq 0$ is of the level k.

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, a variable $x \in X$ and a value $v \in \mathbb{Q}$, we call the *update of* σ *at* x *by* v, denoted by σ_x^v, the assignment obtained from σ by changing the value of x by v and leaving the values of all other variables unchanged.

For a linear form q over X, denote by $q\sigma$ the value of q after replacing all variables $x \in X$ by the corresponding values $\sigma(x)$. An assignment σ is called a *solution of a linear constraint* $q \diamond 0$ if $q\sigma \diamond 0$ is true; it is a *solution of a system* of linear constraints if it is a solution of every constraint in the system. 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). A system of linear constraints is said to be *satisfiable* if it has a solution.

For simplicity, we consider only algorithms for solving systems of linear constraints of the form $q \geq 0$, $\bot$ and $\top$.

We define a *state* as a pair (S, σ), where S is a system of linear constraints and σ an assignment. Let $\mathbb{S} = (S, \sigma)$ be a state and k a positive integer. We say that $\mathbb{S}$ *contains a* k*-conflict* $(x_k + p \geq 0, -x_k + q \geq 0)$ if (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 simple say "conflict". Note that if σ is a solution of S, then $\mathbb{S}$ contains no conflicts.

3 The Conflict Resolution Algorithm

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 by either adding a new linear constraint to S or updating the assignment. We will formulate these rules 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 conflict resolution rule (CR) (at the level k) is the following rule:

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

Algorithm 1 The Conflict Resolution Algorithm CRA

Input: A set S of linear constraints.
Output: A solution of S or "unsatisfiable".

if $\perp \in S$ **then return** "unsatisfiable"
$\sigma :=$ arbitrary assignment;
$k := 1$
while $k \leq n$ **do**
  if $\sigma \not\models S_{=k}$ **then**
    while (S, σ) contains a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$ **do**
      $S := S \cup \{p + q \geq 0\}$; ▷ application of **CR**
      $k :=$ the level of $(p + q \geq 0)$;
      if $k = 0$ **then return** "unsatisfiable"
    end while
    $\sigma := \sigma^v_{x_k}$, where v is an arbitrary value in $I(S, \sigma, k)$ ▷ application of **AR**
  end if
  $k := k + 1$
end while
return σ

where (S, σ) contains a k-conflict $(x_k + p \geq 0, -x_k + q \geq 0)$.
The assignment refinement rule (AR) (at the level k) is the following rule:

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

where

1. σ satisfies all constraints in S of the levels $0, \ldots, k-1$.
2. σ violates at least one constraint in S of the level k.
3. $\sigma^v_{x_k}$ satisfies all constraints in S of the level k.

We will call any instance of an inference rule an *inference*. Thus, our algorithm will perform CR-inferences and AR-inferences. Note that the conflict resolution rule derives a linear constraint violated by σ.

In the description of the conflict resolution algorithm we use the following notation. For every set S of linear constraints and a positive integer k, denote by $S_{=k}$ (respectively, $S_{<k}$) the subset of S consisting of all constraints of the level k (respectively, of all levels strictly less than k). For any system S of linear constraints, a non-negative integer k and an assignment σ denote

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

Informally, the interval $I(S, \sigma, k)$ will be used in our main algorithm to define the range of admissible values of the variable x_k for the assignment refinement rule.

The *Conflict Resolution Algorithm CRA* is given as Algorithm 1. CRA is shown to be sound, complete and terminating in [1].

The 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 next section.

4 Strategies for Conflict Resolution

In this section we discuss strategies and heuristics used for fine-tuning the conflict resolution method. First we consider strategies based on the main parameters of the CRA algorithm:

1. strategies for selecting conflicts,
2. strategies for selecting values in the assignment refinement rule,
3. strategies for selecting the order on variables.

Then we discuss optimization-related strategies for: i) reducing the number of derived constraints, ii) dealing with half-bounded levels and iii) reducing coefficients in the constraints.

For each of the heuristics we introduce short namings and then combine them to address particular set of heuristics in our experiments and discussions.

4.1 Strategies for Selecting Conflicts

The issue of selecting a conflicting pair of constraints arises naturally when more than one conflicting pairs occur on a level. We implemented a number of various strategies. To illustrate these strategies, we will use the following example:

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

We consider order on variables $x_4 \succ x_3 \succ x_2 \succ x_1$ and initial assignment $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 0; x_2 \mapsto 0; x_1 \mapsto 0\}$. To illustrate the algorithm, we will split all inequalities into subsets corresponding to their levels. This initially gives two non-empty levels as shown bellow:

Level 4

$$\begin{array}{rrrrrrrrr}
(1) & 2x_3 & - x_1 - 5 \leq x_4 & & x_4 \leq & x_3 & + 2x_1 - 4 & (3)\\
(2) & x_3 - x_2 & - 2 \leq x_4 & & x_4 \leq -x_3 + & & + x_1 + 1 & (4)
\end{array}$$

Level 3

$$\begin{array}{rrrrrrrrr}
(5) & & - x_1 + 1 \leq x_3 & & x_3 \leq & & x_2 - 2x_1 + 5 & (6)
\end{array}$$

CRA starts with level 3. At this level $\sigma \not\models S_{=k}$ and the interval $I(S, \sigma, 3) = [1; 5]$ is non-empty, thus AR rule is applicable. AR refines the assignment $\sigma := \sigma^v_{x_3}$, updating the value of the variable x_3 by v, where v is an arbitrary value in $I(S, \sigma, 3)$. Let $v = 4$. CRA moves to level 4 with $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$ and $\sigma \models S_{<4}$. At level 4 the interval $I(S, \sigma, 4)$ is empty $[3; -3]$ and CRA detects 4 conflicts (indeed, all pairs of constraints of different signs are conflicting). To proceed, CRA selects one of the conflicts.

In the following we discuss our strategies for selecting conflicts.

Algebraic or Maximal Overlap approach (MO). One of the strategies we tried is based on maximal overlaps, defined as follows. 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 the notion of 'almost' non-redundant constraints, defined in [1]. For readers' convenience we recall this notion here.

Lemma 1. *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 $S_{<k} \cup (S^+ - S')$.* ❑

Based on this definition, choice of a conflict with maximal overlap guaranties that the constraints $x_k + p \geq 0$ and $-x_k + q \geq 0$ are 'almost' non-redundant in the above sense. In our example the maximal overlap $[3; -3]$ is obtained for the conflict: $((1), (4))$. Resolvent of the conflict $x_3 \leq \frac{2}{3}x_1 + 2$ is added to level 3. New bounds on the variable x_3 define a non-empty interval $I(S, \sigma, 3) = [1; 2]$ which does not contain the current value of $x_3 = 4$. The assignment refinement rule is applied. Let the new value of x_3 be 1. Moving to level 4, the algorithm detects the only conflict $((2), (3))$ giving the empty interval $I(S, \sigma, 4) = [-1; -3]$. This time resolvent of the conflict $-2x_1 + 2 \leq x_2$ is added to level 2. Having the only constraint at level 2 the interval $I(S, \sigma, 2)$ is half-bounded $[2; +\infty)$ and the current value of $x_2 = 0$ lies outside it. Again, the assignment refinement rule updates σ by assigning x_2 a new value from the interval, suppose this value is 2. One can easily check that following up to level 3 and level 4 no more conflicts are formed and no assignment refinement is needed. Thus, the system is satisfiable and $\sigma : \{x_4 \mapsto 0, x_3 \mapsto 1, x_2 \mapsto 2, x_1 \mapsto 0\}$ is a solution.

Geometric or Relaxation Method approach (RM). Another strategy for selecting a conflict comes from the geometrical ideas behind the *relaxation method*, (see, e.g., [8]). As we know, an assignment σ represents a point M in the n-dimensional space and the system of linear inequalities S defines a polyhedron in this space. The relaxation method iteratively changes the assignment trying to get inside the polyhedron defined by S. New assignment is chosen by reflecting M over a hyperplane that (i) is defined by a constraint in S that is 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 on the maximal distance from M. A constraint defining such a hyperplane is called *the most violated* constraint. The original relaxation method has a substantial drawback – each iteration leads to solving of a new problem. Moreover the relaxation method does not always terminate, producing approximations converging to a solution but never achieving it.

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, and moreover guarantees termination.

Let us show how CRA selects a conflict using the geometric approach. Let us return to level 4 with the refined assignment $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 4; x_2 \mapsto 0; x_1 \mapsto 0\}$. As mentioned above, all pairs are conflicting at this stage. Assignment σ corresponds to the point $M(0, 4, 0, 0)$ which is outside the solution space. To use the geometric approach, CRA searches for the hyperplane which is defined by one of the resolvents of these conflicts and is the furthermost to M. The distance from a point $P = (p_n, \ldots, p_1)$ to a hyperplane corresponding to the constraint $a_n x_n + \ldots a_1 x_1 + b \geq 0$ is calculated using the formula:

$$\frac{|a_n p_n + \ldots a_1 p_1 + b|}{\sqrt{a_n^2 + \ldots + a_1^2}}$$

Since we need the maximal distance we compared squares of the distances to avoid calculations with roots. It is easy to see, that the furthermost hyperplane to M is defined by the resolvent of the conflict $((2), (4))$: $x_3 \leq \frac{1}{2}x_2 + \frac{1}{2}x_1 + \frac{3}{2}$. This constraint is of level 3. The interval $I(S, \sigma, 3)$ becomes $[1; \frac{3}{2}]$ and the assignment can be refined by updating the value of x_3. Let $\sigma := \sigma^1_{x_3}$. Moving to level 4, we find that the interval is empty $I(S, \sigma, 4) = [-1; -3]$ and the only conflicting pair of constraints is $((2), (3))$. The resolvent of this conflict $-2x_1 + 2 \leq x_2$ is added to level 2. This constraint defines a half-bounded interval for the values of $I(S, \sigma, 2) = [2; +\infty)$ which does not contain the current value of x_2. Let us update the assignment by setting $x_2 := 2$. Following the algorithm it is easy to check that all constraints at level 3 and level 4 are satisfied, thus the system is also satisfied and $\sigma : \{x_4 \mapsto 0; x_3 \mapsto 1; x_2 \mapsto 2; x_1 \mapsto 0\}$ is a solution.

Take the first conflict (FC). The next strategy we tried simply takes the first detected conflict. It saves the calculation time needed for computing all conflicts at a level. The first conflict detected in our example is $((1), (3))$. Its resolvent $x_3 \leq 3x_1 + 1$ is of level 3 and narrows the interval $I(S, \sigma, 3)$ to $[1; 1]$. Thus $x_3 := 1$. Moving to level 4 with a refined assignment $\sigma : \{x_4 \mapsto 0, x_3 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 0\}$ CRA detects the only conflict $((2), (3))$. Obtained resolvent $-2x_1 + 2 \leq x_2$ sets bounds on the variable x_2 at level 2: $I(S, \sigma, 2) = [1; +\infty)$. By setting $x_2 := 2$ one can check that CRA passes all remaining levels without any conflicts and changes in the assignment.

Random choice of conflict approach (RC). The last conflict selection heuristics we implemented is randomly choosing a conflict with an equal probability. Assume that the randomly selected conflict is $((2), (3))$. The derived constraint $-2x_1 + 2 \leq x_2$ is of level 2 and the bound on x_2 forms a half-bounded interval $I(S, \sigma, 2) = [2; +\infty)$. If the value of x_2 is set to 2, we move to level 3 and pass it without refining an assignment. At level 4 CRA detects that only the selected conflict $((2), (3))$ is resolved, and three other conflicts are remained. Let us select the conflict randomly again, and assume it is $((1), (4))$. Its resolvent $x_3 \leq \frac{2}{3}x_1 + 2$ narrows the interval for x_3 to $[1; 2]$ and forces the value of x_3 to be updated. Let $x_3 := 2$. One can check that all constraints at level 4, and thus in the initial system, became satisfied.

As we see each of the conflict selection strategies results in a different behaviour of the algorithm. The running time of the algorithm depends significantly on the choice of

values for the variables in the assignment refinement rule. In the following, we discuss the strategies for assignment refinement used in our experiments.

4.2 Strategies for Assignment Refinement

We tried several strategies for selecting values in the assignment refinement rule, we list them bellow.

Minimal (Maximal) point (MIN/MAX). Select always the minimal (or always the maximal) endpoint of the interval $I(S, \sigma, k)$. As we saw from the examples of using strategies for selecting conflicts, the course of the algorithm and its performance can be significantly affected by the choice of the values of variables. For instance, if the value of the variable x_3 selected from the interval $[1; 5]$ for the first time is 1 and not 4 the algorithm avoids derivation of a conflict needed to adjust the value of x_3.

Interleaved (swapped minimal and maximal) points (SW). A natural extension of the previous strategy is to interleave the selection of the maximal and the minimal endpoints of the interval $I(S, \sigma, k)$ each time the interval is updated.

Random assignment value choice (RA). Another strategy is a random choice of the assignment value within the interval $I(S, \sigma, k)$.

Middle point. Select the middle point of the interval $I(S, \sigma, k)$.

Our experiments show that these strategies can result in a rapid growth of the sizes of numerators and denominators of rational values in the assignment, which in turn leads to heavy calculations. The next strategy is aimed at reducing sizes of rational numbers used in the assignment values.

Closest binary to the middle point (BMP). If the endpoints of $I(S, \sigma, k)$ coincide, we select this point. 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 can be shown 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.

4.3 Strategies for Selecting the Order on Variables

We implemented the following strategies for selecting the order on variables.

Random order (RO). The first strategy we tried sets the order on variables randomly before running the CRA algorithm.

Length-based order (LO). The second strategy orders variables giving preference to variables occurring in short constraints. For simplicity, we formalise the second strategy as follows. To each variable x we associate a pair of integers $(l(x), t(x))$, where $l(x)$ is the length of the shortest constraint containing variable x and $t(x)$ is a number of such constraints. We define the ordering on variables $\succ$ as follows: for two variables x, y we have $x \succ y$ if (i) $l(x) > l(y)$ or (ii) $l(x) = l(y)$ and $t(x) > t(y)$. That is, we try to put variables occurring in shortest constraints on lower levels.

4.4 Optimization-related strategies

There are also other heuristics that we considered interesting to study. One of them concerns adding resolvents to the current system at run-time.

Adding resolvents. In general, adding a derived constraint to its level may result in new conflicts at this level. Thus resolving one conflict may result in a cascade of conflicts in lower levels. Adding all such consecutive resolvents may result in a quick expansion of the system. Based on this we studied the following heuristics:

1. add all resolvents derived during the run-time of the algorithm;
2. do not add a resolvent if it results in a new conflict at its level, rather keep resolving conflicts without adding them until a resolvent is derived which results in no conflicts (backjumping – **BJ**).

The latter heuristic describes a process of 'backjumping' to the lowest level (the first non-conflicting level) and adding only the last resolvent to the system.

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.

Dealing with half-bounded intervals in the AR rule. In our experiments half-bounded intervals in the assignment refinement rule occur very frequently. We deal with such cases by introducing an artificial bound on the intervals. If during the run-time CRA returns to a level with a half-bounded interval considerably often, we considered increasing the size of the interval consecutively. We tried two heuristics for dealing with half-bounded intervals:

1. increase the size of an artificial interval exponentially each time CRA returns to the corresponding level; in the experiments we tried this strategy with increasing by powers of 2 (**pow2**);
2. keep the size of artificial intervals constant, in the experiments fixed to 10 (**const10**).

Reducing constraints by gcd. The last implemented heuristic corresponds to the problem of decreasing size of numerators and denominators of rational numbers during the calculations. Namely, if all coefficients in a constraint have the greatest common divisor different from 1 than the constraint can be reduced by dividing its coefficients by their greatest common divisor. This simple idea yields the heuristics:

1. always reduce constraints by gcd (**gcd**);
2. never reduce constraints by gcd.

We studied various combinations of all presented heuristics and integrated them into the CRA algorithm. We call *major heuristics* the ones (i) for selecting conflicts, and (ii) for selecting assignment values. The heuristics (a) for dealing with half-bounded intervals in the assignment refinement rule and (b) for reducing constraints by gcd of their coefficients are *general* in their nature and can be combined with any major heuristic mentioned above.

5 Benchmarks

This section describes the benchmarks used in our experiments. We evaluated our solver on two types of benchmarks: randomly generated benchmarks and benchmarks extracted from real-life problems. The real-life benchmarks consist of systems of linear constraints extracted from SMT-LIB problems [3].

Real-life problems are substantially different from randomly generated ones. They differ not only by their size but also by their structure. In real-life problems the number of variables and constraints is considerably higher and most of the problems contain hundreds of variables and constraints. In addition, real-life problems often have sparse matrices and relatively simple coefficients.

Benchmarks with randomly generated problems. We used random benchmarks with integer coefficients generated by the GoRRiLA tool [5]. GoRRiLa is a generator of random problems for propositional logic and for systems of linear constraints over the rational or integer numbers. GoRRiLA can generate random problems of a given number of variables, so that each constraint has the number of variables in a certain range (for example, between 3 and 5 variables).

We evaluated CRA on two sets of random benchmarks.

1. The first set consists of 1600 problems with a number of variables ranging from 11 to 18;
2. The second set consists of 400 problems with a number of variables ranging from 19 to 26.

Benchmarks Extracted From SMT-LIB. 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 library. We used the benchmarks from the QF_LRA division of the SMT-LIB: these benchmarks contain quantifier-free SMT problems in the theory of linear real arithmetic.

We obtained real-life benchmarks using the Hard Reality Tool (HRT) [6]. 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.

We used two sets of real-life benchmarks generated by us with the Hard Reality Tool (HRT) [6]. The difference between these sets is in their difficulty levels which reflect the time needed to solve the problem by the best solver. The sets contain both satisfiable and unsatisfiable problems.

1. The first set consists of 305 problems with a number of variables ranging from 37 to 1416.
2. The second set consists of 128 problems of considerable higher difficulty level with a number of variables ranging between 251 and 1067.

6 Preprocessing

Our real-life benchmarks contained several hundreds of variables and constraints. On many instances we observed that our solver was continuously passing a considerable

number of levels, adding new constraints to them and expanding the system without any contribution to the solving process. We used the following preprocessing of the input system which considerably improved the performance on such problems.

Eliminating Half-Bounded Variables. Let us note, that if all occurrences of a variable x in the system are of the same sign, than x will never be eliminated with the CR rule. Moreover, if a derived constraint contains a variable x, its coefficient in this constraint will be of the same sign too. This means, the levels of half-bounded variables may expand during the run-time of the CRA algorithm but never give a bounded interval for the corresponding variables.

In such instances the conflict resolution algorithm would run along half-bounding levels never obtaining both bounds for the variables. To avoid this we can remove from the system all constraints containing such a half-bounded variables. Solve the system of remained constraints, if it has no solutions, than the original system has no solution either. Otherwise, we assign a value to all such removed variable based on the values of the variables in the obtained solution. Since the intervals for the removed variables were half-bounded such an assignment always exists.

Consecutively removing all such variables from the system reduces the number of variables in the system, the number of initial constraints, and the number of constraints derived at the run-time.

Eliminating unit-half-bounded variables. We extend the above preprocessing by considering *unit-half-bounded variables*. The difference between unit-half-bounded variables and half-bounded variables is in allowing unit constraints to bound the variable. A constraint is called *unit constraint*, if it contains only one variable. Consider a unit constraint such that either (i) an equality $x = a$ (where $a \in \mathbb{Q}$) or (ii) an inequality that has the coefficient of the variable of the sign opposite to the sign of the coefficients of this variable in the rest of the constraints.

If (i) holds, then the unit constraint, of the form $x = a$, explicitly assigns a value to the variable x. We can directly eliminate x from the system by simply substituting all occurrences of x by a. Obviously, this brings us to the equivalent system with one variable less. If the modified system has a solution, the solution of the initial system can be easily obtained by expanding the assignment with the value a for the variable x.

Note, that we can eliminate from the system all such variables one after another, thus reduce the dimension of the system.

Case (ii) also allows elimination of the variable x from the system. We can eliminate the variable x from the system by simply summing the unit constraint with the rest of the constraints containing x. Obviously, this operation also results in an equivalent system of constraints by solving which we can easily obtain a solution to the initial system – we build it by extending the solution of the derived system for the variable x. The value for the variable x is obtained from the interval defined by the initial constraints containing x, by simply substituting the values of the other variables. If the derived system has no solutions, the initial system has no solutions either. Similarly to the previous cases, it is possible to eliminate all such variables one after another, reducing the dimension of the system this way. Let us note that after eliminating a variable new variables may become eligible for the preprocessing, We apply this preprocessing exhaustively, i.e., until no eligible variables remains.

List of Strategies					
Selecting Conflict		**Selecting Assignment**		**Optimisation-related**	
Abbreviation	**Strategy**	**Abbreviation**	**Strategy**	**Abbreviation**	**Strategy**
FC	First Conflict	**BMP**	Binary Middle Point	**BJ**	Back Jumping
MO	Maximal Overlap	**MAX**	Maximal Endpoint	**RO**	Random Order
RC	Random Conflict	**MIN**	Minimal Endpoint	**LO**	Length-based Order
RM	Relaxation Method	**SW**	Swap Endpoints	**pow2**	Exp. Half-bounded
		RA	Random Assignment	**const10**	Const. Half-bounded
				gcd	Reduction by gcd
				prep	Preprocessing

Table 1. Abbreviation of Strategies

7 Experimental Evaluation

We run our experiments on Intel Xeon Quad Core machines with 2.33 GHz and 12 GB of memory.

For readers' convenience we name heuristics based on the abbreviation of major strategies for selecting conflicts (see Section 4.1) and assignment refinement (see Section 4.2) together with other general heuristics (see Section 4.4). For example in a heuristic MO_MAX_pow2 we use the maximal overlap strategy for conflict selection, the maximal point strategy for assignment refinement and increase half bounded intervals by powers of two. We present a full list of abbreviations used throughout the paper in Table 1.

We determine the best choice of combination of heuristics for both random and real-life benchmarks.

First we select the best set of general heuristics for each major heuristic. Then we compare selected combinations of heuristics between each other.

7.1 Randomly Generated Benchmarks

On randomly generated problems all implemented strategies had certain similarities in performance and behaviour.

- In the problems with the number of variables ranging between 11 and 18 all 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 and 26.

Experiments showed that the reduction of constraints by gcd is almost always beneficial. A better performance was also observed when we increased half bounded intervals by powers of two.

We plotted all major heuristics combined with the best choices of general heuristics for them on Fig. 1. A point (x, y) on the chart indicates that x problems were solved in y time or less.

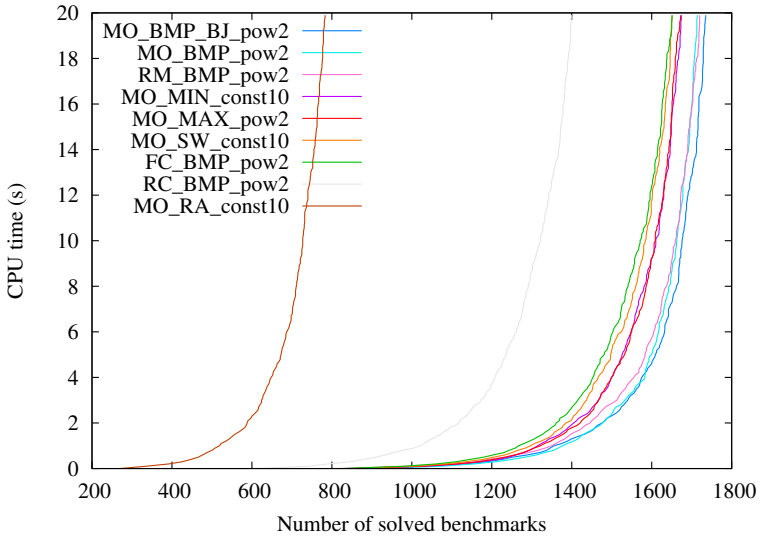

Fig. 1. CRA: major heuristics on randomly generated problems (all using reduction by gcd). Heuristics are labelled from right to left.

As wee see, the best heuristic appeared to be MO_BMP_BJ_pow2, followed by MO_BMP_pow2 and then by RM_BMP_pow2, all close to each other. This chart shows that our non-random heuristics considerably outperform the random ones.

7.2 Real-Life Benchmarks

For real-life benchmarks we added preprocessing, as discussed in Section 6, and plotted results of our experiments for the combinations of heuristics that were more representative, see Fig. 2. As we see, the top performances were shown by MO_BMP_pow2, FC_pow2, and RM_BMP_pow2.

In our experiments, the performance of various bundles of heuristics of the CRA algorithm was different on randomly generated benchmarks and real-life benchmarks. On both types of problems we have large difference in performance based on different choices of heuristics. This indicates that the good choice of heuristics is crucial for the performance of CRA.

Regarding the best performances, for both randomly generated and real-life problems, almost always combinations of the following major heuristics were showing the top three performances: maximal overlap (MO) and relaxation method (RM) strategies for conflict selection, combined with the binary middle point strategy (BMP) for the assignment refinement and also with the back jumping (BJ).

In combination with most major heuristics reduction by gcd as well as exponential increase of half-bounded intervals was effective, for many problems, but with some it

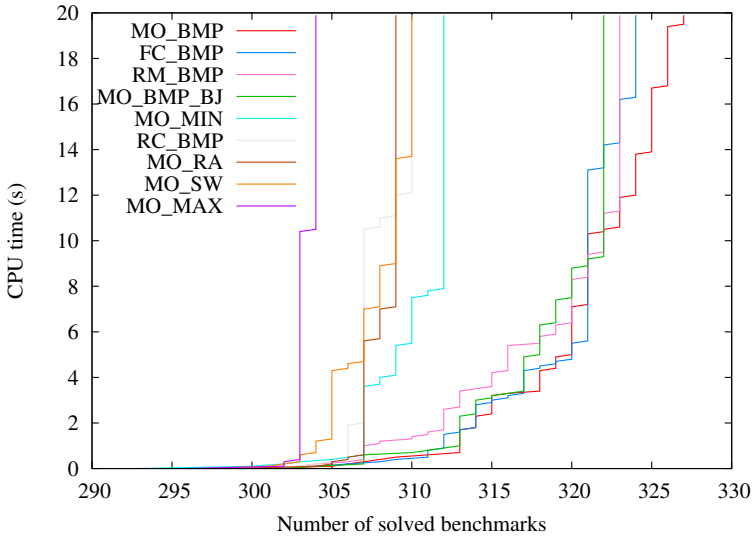

Fig. 2. CRA: major heuristics on real-life problems (all using preprocessing and gcd_pow2 strategy). Heuristics are labelled from right to left.

had insignificant improvement. Overall, the use of reduction by gcd and exponential increase of artificial bound of intervals were almost always beneficial.

7.3 CRA vs state-of-the-art SMT solvers

For further experiments we used the implementation of the CRA algorithm that uses one of the best combinations of the heuristics listed in the previous sections, and incorporates the preprocessing discussed earlier in Section 6 and the length-based variable ordering. Namely, we used a strategy MO_BMP_pow2_gcd_LO_prep which incorporates preprocessing, the length-based order on variables and following strategies: maximal overlap strategy for selecting the conflict, closest binary middle point strategy for the assignment refinement, reduction of constraints by gcd and exponential increase of half-bounded intervals.

In the following we present results of comparison of this implementation of CRA to other linear arithmetic solvers incorporated in the state-of-the-art SMT solvers. On Fig. 3 we compared on real-life problems different solvers: Barcelogic, CVC3, Z3, and CRA incorporating preprocessing and the length-based order on variables.

On studied real-life benchmarks CRA considerably outperforms the Fourier-Motskin elimination based CVC3 solver, and shows a competitive behaviour compared to the simplex based Barcelogic and Z3. In our experiments with random problems CRA outperforms CVC3 on almost all problems and in some cases CRA is about twice as faster as Barcelogic and Z3.

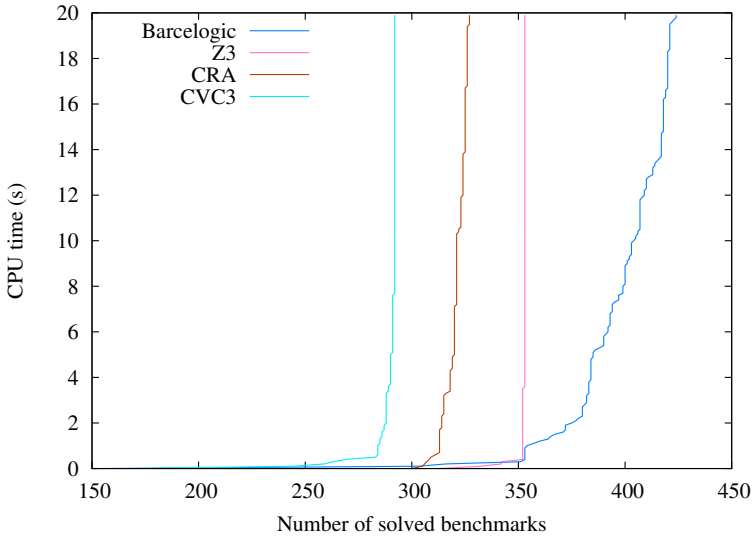

Fig. 3. CRA (MO_BMP_pow2_gcd_LO_prep) vs CVC3, Z3 and Barcelogic on real-life problems. Heuristics are labelled from right to left.

CRA	**Faster**	**Same**	**Slower**
Barcelogic	165	119	148
CVC3	157	274	1
Z3	18	48	366

Table 2. CRA (MO_BMP_pow2_LO_prep) vs Barcelogic, CVC3, Z3 on real-life benchmarks.

The results of the experiments with real-life benchmarks are presented in the Table 2.

CRA with implemented 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 85% of the problems it showed similar performance, outperformed Z3 on about 4% of the problems, and was slower than Z3 on about 11%. As we see, the CRA algorithm not only outperforms the Fourier-Motzkin algorithm but is also highly competitive with the simplex method.

8 Summary

Our experiments showed that choosing various parameters has a significant impact on the performance of the CRA solver. Also, depending on the nature of the problem different heuristics may appear preferable. Let us outline the strategies behaving the best in general.

In selecting a conflicting pair the best two choices turned out to be the maximal overlap strategy and the relaxation approach. For selecting the assignment one could overall recommend taking the middle point (based on the binary approximation). Interestingly, using the boundary value assignments may also appear successful in some special cases.

Among the general heuristics, the reduction by gcd and exponential increase in the case of of half-bounded intervals turned out to improve the performance nearly in all cases.

The algorithm appeared to be sensitive to the implemented combination of heuristics and preprocessing. Integrating effective preprocessing to the CRA algorithm and more strategies for selecting appropriate ordering on variables may also result in a significant benefit. This problem needs further detailed studies.

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.

References

1. K. Korovin, N. Tsiskaridze, A. Voronkov. Conflict Resolution *CP 2009*, volume 5732 of *LNCS*, pages 509–523, Springer Verlag 2009.
2. C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV '07*, volume 4590 of *LNCS*, pages 298–302. Springer-Verlag, 2007. Berlin, Germany.
3. C. Barrett, S. Ranise, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.
4. V. Chandru. Variable elimination in linear constraints. *Comput. J*, 36(5):463–472, 1993.
5. K. Korovin and A. Voronkov. GoRRiLA. Available at: http://www.cs.man.ac.uk/~korovink/rpg, 2009.
6. K. Korovin and A. Voronkov. Hard Reality Tool. Submitted, available at http://www.cs.man.ac.uk/~korovink/hr, 2009.
7. R. Nieuwenhuis and A. Oliveras. Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. (Invited Paper). In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.
8. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.
9. L. de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*. *TACAS*, 2008, pages 337–340.