Journal of Universal Computer Science, vol. 4, no. 2(1998), 125-146
submitted: 25/9/97, accepted: 1/11/97, appeared: 28/2/98 © Springer Pub. Co.

A Symbolic-Numerical Branch and Prune Algorithm for Solving Non-linear Polynomial Systems

Laurent Granvilliers

LIFO – Univ. Orléans – IIIA – Rue L. de Vinci
BP 6759 – 45067 ORLÉANS Cedex 2 – FRANCE
Laurent.Granvilliers@lifo.univ-orleans.fr

Abstract: This paper discusses the processing of non-linear polynomial systems using a branch and prune algorithm within the framework of constraint programming. We propose a formalism for a kind of branch and prune algorithm implementing symbolic and numerical methods to reduce the systems with respect to a relation defined from both inclusion of variable domains and inclusion of sets of constraints. The second part of the paper presents an instantiation of this general scheme. The pruning step is implemented as a cooperation of factorizations, substitutions and partial computations of Gröbner bases to simplify the systems, and interval Newton methods address the numerical, approximate solving. The branching step creates a partition of domains or generates disjunctive constraints from equations in factorized form. Experimental results from a prototype show that interval methods generally benefit from the symbolic processing of the initial constraints.

Key Words: Branch and prune algorithm, non-linear constraint solving, cooperative constraint solvers, symbolic simplification, interval Newton methods, Gröbner basis, polynomial system.

1 Introduction

Simplification methods are foundations of computer algebra, used to normalize formulae or to preprocess it before applying an appropriate resolution algorithm. Among others, Gröbner bases computations [10] reduce sets of non-linear polynomials by deriving a particular basis of the ideal defined by the initial polynomials, and cylindrical algebraic decomposition [11, 12] simplifies quantified formulae over the reals.

Interval analysis was introduced in the early sixties by Moore [32] to handle round-off errors of numerical computations using machine numbers, and to quantify precisely the accuracy of the results. The key idea is to extend each real operation on intervals taking rounding of the bounds into account, and to evaluate real functions on intervals. Doing so, the range of a real function is ensured to be included in its interval evaluation, which guarantees correctness of computations. Based on interval arithmetic, various methods for solving systems of non-linear equations were defined in the interval community [32, 2, 19, 27, 35]. Most of these are derived from Taylor series approximations, the best known being the interval version of Newton's method, whose purpose is to search zeros of univariate interval functions. Being intrinsically incomplete, these methods have to be combined with enumeration techniques for deriving each solution, which defines a *branch and prune algorithm*. It iterates a pruning step which reduces the search space of the initial system and a branching step which generally consists in splitting the search space.

Moore [32] showed that interval computations strongly depend on the forms of real functions and bad expressions of functions may lead to very inefficient interval evaluations. For example, the interval evaluation of $f(x) = x - x$ over $[0, 1]$ is $[-1, 1]$ while its range is $[0, 0]$. This comes from a weakness of interval operations computing with bounds of intervals[1] and then the two occurrences of x are considered as different variables. To handle such situations arising when solving systems of equations over intervals, various methods were proposed. Among others, *centered forms* [32, 37, 36, 3] and *preconditionning* [26] using diagonally dominant Jacobian matrices guarantee good convergences of interval solving.

Recently, the cooperation of constraint solvers was discussed in the cc framework [38] and in the CLP context [20, 4, 31]. In particular, several works [29, 6] propose to combine symbolic and interval methods for solving systems of non-linear equations. Essentially, symbolic methods generate redundancies speeding-up interval methods. However, more work can be done during the symbolic process. Let us consider the system of equations $\{xy - x = 0, x^{25} + 2xy + 1 = 0\}$. The factorization of $xy - x = 0$ allows to instantiate the second equation with either $x = 0$ or $y = 1$. This generates the disjunction $\{x = 0, 1 = 0\} \cup \{y = 1, x^{25} + 2x + 1 = 0\}$. Both systems can be easily solved using numerical methods. Moreover, factorization followed by generation of disjunctive equations reduces the *dependency* between variables, which is pointed out by Moore as a main problem for interval methods.

This led us to extend the heterogeneous constraint solving model described in [4] and classical branch and prune algorithms to take into account the generation of disjunctive constraint systems during the symbolic process. Constraint systems are viewed as conjunctions of disjunctions of real constraints and are processed by iterating two steps: the application of closure operators, called *constraint narrowing operators*, reduces the variable domains or adds some redundancies; enumeration is implemented by *constraint branching operators* to split the variable domains or the equations having a factorized form. The second part of the paper is devoted to an instantiation of this general framework. We propose an algorithm for solving non-linear polynomial equations over the reals, combining partial computations of Gröbner bases [6], factorizations and substitutions over rational equations, and interval Newton methods over arbitrary equations. Algebraic methods are used to reduce the dependency between variables for speeding-up interval computations. Experimental results from a prototype are given and show that symbolic transformations of systems greatly improve the computational behaviour of interval methods for most of the presented benchmarks.

With respect to the related papers mentioned above, we believe that this paper proposes a novel cooperation of methods from computer algebra and numerical analysis for solving non-linear polynomial equations. An important collection of benchmarks from various domains are tested with a prototype, and show significant speed-ups due to the symbolic transformations of constraints. Then, we identify different behaviours of the software for different constraint systems, in particular when the cooperation involves a slow down. A discussion ends the section relating the experimental results and focuses on the needs for developing tight cooperation strategies between solvers together with parallelism.

The rest of the paper is organized as follows. Section 2 recalls the basics

[1] $[a, b] - [c, d]$ is $[a - d, b - c]$ with outward rounding of the bounds.

of approximations, interval arithmetic and Gröbner bases. The motivations of symbolic transformations of real constraints for improving interval computations are introduced in section 3. We define a generic branch and prune algorithm in section 4 for solving real constraints. Section 5 describes an instantiation of this algorithm for solving polynomial systems. Section 6 presents the experimental results and we conclude in section 7.

2 General framework

Let Σ be a structure $< \mathbb{R}, \mathcal{O}, \mathcal{R} >$ where $\mathbb{R}$ is the set of real numbers, called the universe, $\mathcal{O}$ a set of function symbols and $\mathcal{R}$ a set of relation symbols. Let $V = \{x_1, \ldots, x_n\}$ be a set of variables taking their values over $\mathbb{R}$. This set is assumed to be large enough to built the constraints described here. Terms in the constraint language are syntactic expressions built in the usual way from constants, variables and operations. Constraints are syntactic expressions representing relations made from terms and relation symbols. In this paper, terms are multivariate polynomials and constraints are polynomial equations over $\mathbb{R}$. The power set of $\mathbb{R}$ is denoted $2^{\mathbb{R}}$.

In the following, a constraint and the relation it represents are denoted by the same symbol.

2.1 Approximations

Solving real constraints being an intractable problem when using machine numbers, approximations of the set of solutions are computed over *approximate domains.*

Definition 1 (Approximate domain) *An* approximate domain A *over* $\mathbb{R}$ *is a subset of* $2^{\mathbb{R}}$*, closed under intersection, such that* $\mathbb{R} \in A$ *and for which the inclusion is a well-founded ordering.*

Given an approximate domain A, the *approximation* over A of a unary relation ρ, denoted $\underline{apx}_A(\rho)$, is defined as the intersection of all the elements of A containing ρ. The approximation of an n-ary relation $\rho = (\rho_1, \ldots, \rho_n)$ is the Cartesian product of the approximations of each of its projections $\underline{apx}_A(\rho_1) \times \cdots \times \underline{apx}_A(\rho_n)$. A tuple $\vec{X} = (X_1, \ldots, X_n)$ of elements of A, called a *box*, denotes the Cartesian product $X_1 \times \cdots \times X_n$ (see [7] for a complete discussion about approximations).

2.2 Interval arithmetic

Interval computations restore correctness of numerical computations, enclosing solutions with *intervals* being bounded compact sets of reals. Termination is obtained by taking a finite number of possible values for the bounds of intervals (in practice the floating point numbers), together with some convergence properties. Correctness is addressed providing a mechanism of *outward rounding* of computations at bounds of intervals. We recall in this section the essentials which can be found in [32, 2].

Definition 2 (Interval) *Let R be a finite subset of $\mathbb{R}$ and l, r be two reals such that $l \in R \cup \{-\infty\}$ and $r \in R \cup \{+\infty\}$. An* interval $I = [l, r]$ *is the compact set of reals $\{x \in \mathbb{R} \mid \mathrm{l} \leq x \leq \mathrm{r}\}$. In particular $[-\infty, +\infty]$ corresponds to $\mathbb{R}$, $[-\infty, r]$ represents $\{x \in \mathbb{R} \mid x \leq \mathrm{r}\}$ and $[l, +\infty]$ denotes $\{x \in \mathbb{R} \mid \mathrm{l} \leq x\}$. All the intervals $[l, r]$ such that $l > r$ are mapped to the empty interval $\emptyset$.*
The set of all the intervals over R is denoted $\mathbb{I}(R)$.
An interval in $\mathbb{I}(R)$ is said to be canonical *if it contains at most two elements of R (then $\emptyset$ is canonical).*

$\mathbb{I}(R)$ is an approximate domain with respect to the set inclusion ordering. A real r is approximated by the smallest interval containing r. More generally, a relation ρ over $\mathbb{R}$ is approximated by the smallest interval containing all the elements in ρ, using the approximation function over $\mathbb{I}(R)$. To simplify the notations, $\mathbb{I}(R)$ is denoted $\mathbb{I}$ considering the implicit existence of a set R; in practice, it is the set of floating point numbers in double precision [23], in which case an element of $\mathbb{I}$ is said to be a floating point interval.

Interval arithmetic operations are set extensions of real operations. Given $\cdot$ a real operation, two intervals I, J, and $\odot$ the interval operation corresponding to $\cdot$, $I \odot J$ is $\underline{\mathrm{apx}}_{\mathbb{I}}(\{x = y \cdot z \mid y \in I, z \in J\})$. An n-ary *interval function* is a mapping $\mathbb{I}^{\mathrm{n}} \rightarrow \mathbb{I}$ involving interval constants, interval operations and interval variables. Van Hentenryck et als. [40] introduced various forms of *interval extensions* to approximate over intervals a real function. They are defined as interval functions obtained from its by a syntactic operation. In particular, two different expressions of a same real function may lead to different interval extensions. Our definition of interval extension corresponds to the natural interval extension of [40].

Definition 3 (Interval extension) *Let f be an n-ary real function. The* interval extension *of f is a mapping $\mathbb{I}^{\mathrm{n}} \rightarrow \mathbb{I}$ obtained from f by replacing in the expression of f each constant r by $\underline{apx}_{\mathbb{I}}(\{r\})$, each operation by its corresponding interval operation and each variable by an interval variable.*

Interval extensions are correct, i.e. enclosing the range of variation of real functions, and monotonic. We recall the property of *subdistributivity* of interval arithmetic.

Property 1 (Subdistributivity) *Let $I, J, K \in \mathbb{I}$. Then*
$$I \otimes (J \oplus K) \subseteq (I \otimes J) \oplus (I \otimes K).$$

2.3 Gröbner bases

The theory of Gröbner bases can be found in [10, 13]. A polynomial is a sum of monomials which are ordered using *a monomial ordering*. A monomial ordering is a total, well-founded ordering on monomials which is compatible with the multiplication on monomials. In what follows, we fix the monomial ordering and consider that all the polynomials are ordered with respect to this ordering. For every polynomial p, the *leading term* of p, denoted $LT(p)$, is the monomial of p which is maximal with respect to the monomial ordering.

Example 1 (Lexicographic ordering) *Let $x_1, \ldots, x_n$ be n variables alphabetically ordered ($x_1 > \ldots > x_n$). A monomial $x_1^{d_1} \ldots x_n^{d_n}$ is greater than a* *monomial* $x_1^{d'_1} \dots x_n^{d'_n}$ [2] *iff it exists* i *such that* $d_i > d'_i$ *and for all* j *smaller than* i, $d_j = d'_j$.

A S-polynomial is a combination of two polynomials producing the cancellation of their leading terms.

Definition 4 (S-polynomial) *Let* p *and* q *be two nonzero polynomials and* x^γ *denotes the least common multiple of the power products appearing in* $LT(p)$ *and* $LT(q)$. *The S-polynomial of* p *and* q *is the polynomial*

$$S(p,q) = \frac{x^\gamma}{LT(p)}p - \frac{x^\gamma}{LT(q)}q$$

Example 2 *Let* $p = xy^2 - x + y + 1$ *and* $q = 3x^2y + 3x + 2$ *be two polynomials ordered by the lexicographic ordering.* $LT(p) = xy^2$, $LT(q) = 3x^2y$ *and* $x^\gamma = x^2y^2$. *The S-polynomial of* p *and* q *is* $3x.p - y.q = -3x^2 - 2y + 1$.

Given a nonzero polynomial p and $S = \{p_1, \dots, p_n\}$ a set of polynomials, p can be written as $p = \sum_{i=1}^n a_i p_i + r$ such that either r is the zero polynomial or r is a linear combination of monomials such that none of them is divisible by any of $LT(p_1), \dots, LT(p_n)$. The polynomial r is obtained as the remainder of the division of p with respect to S. It is called the *reduction of* p *with respect to* S and is denoted $\overline{p}^S$.

Example 3 *Let* S *be the set* $\{p_1 = xy - 2, p_2 = 2x + 1\}$ *and* $p = 4x^2y^2 + x^2y - y^2 + 1$. *Then,* $p = (4xy + x + 8)p_1 + p_2 - y^2 + 16$. *The remainder of the division of* p *with respect to* S *is* $-y^2 + 16$.

The following definition for Gröbner bases gives their algorithmic characterization and is due to Buchberger [10]:

Definition 5 (Gröbner basis) *Let* $S = \{p_1, \dots, p_n\}$ *be a set of polynomials.* S *is a Gröbner basis iff for all pairs* $(i,j) \in \{1, \dots, n\}$ *the reduction with respect to* S *of the S-polynomial* $S(p_i, p_j)$ *is equal to* 0.

The basic algorithm computing Gröbner bases consists in successive computations of reduced S-polynomials over the initial set of polynomials augmented by the computed S-polynomials different from zero. The algorithm stops, after a finite number of steps, when no S-polynomial different from 0 can be computed. The resulting set of polynomials is a Gröbner basis and is denoted GB(S).

3 Dependency between variables and constraints

It is well known that the efficiency of numerical methods for enclosing zeros of real functions is strongly influenced by their syntactic forms. In particular, interval computations are as efficient as the constraints contain few variables or occurrences of a same variable. This is known as the *dependency problem* [32]. Thus, it is important to simplify the constraints before the interval solving

[2] The degrees can take the 0 value.

process. For this purpose, we identify different methods from computer algebra which are local to a constraint or global to a system of constraints. Since the inclusion of interval functions generally depends on the variable domains, some simplifications may have a heuristics nature. However, interval computations are necessarily more precise if the simplified constraints are added as redundancies.

As shown by Moore, interval arithmetic considers occurrences of a same variable as different variables, which widens widths of computed intervals. The following example also illustrates the needs for factorizing real functions.

Example 4 *Let $f(x) = x^2 - x$ and $g(x) = x(x-1)$ represent two different expressions of the same real function and $F(X) = X^2 \ominus X$, $G(X) = X \otimes (X \ominus [1,1])$ be their interval extensions. Then,*

$$\begin{aligned} &\{f(x) \mid x \in [0,1]\} = \{g(x) \mid x \in [0,1]\} = [-0.25, 0] \\ \subset\ & G([0,1]) = [-1, 0] \\ \subset\ & F([0,1]) = [-1, 1] \end{aligned}$$

In this example, G gives a more precise interval than F, while it does not approximate exactly the range of the real function. Then, it can be asked if the evaluation of G is included in the evaluation of F for every interval, in which case G could always replace F. And second, are different interval extensions of a same real function always comparable? The answer is no, noticing that $G([0,1]) = [-1,0] \subset F([0,1]) = [-1,1]$ while $F([-1,0.5]) = [-0.25,2] \subset G([-1,0.5]) = [-1,2]$. However, it can be remarked that the interval evaluation of $x(x-1)$ is always included in the interval evaluation of $x*x-x$, which is due to the subdistributivity property of interval arithmetic. The difference between $x*x$ and x^2 appears in the removal of the negative part when evaluating x^2 over intervals.

The generation of redundancies may reduce the dependency between variables in the constraints. It is a global method since redundancies are inferred from other, existing, constraints.

Example 5 (Redundancy) *Let $S = \{x^2+y^2-5=0, x^2-y^2-3=0\}$ be a set of real constraints. The equation $x^2-4=0$ is a redundant constraint which can be obtained as a simple linear combination of the constraints in S. The dependency is reduced because the variable x is not dependent on y in this constraint.*

The following example illustrates the substitution of terms by less dependent terms in expressions.

Example 6 (Substitution) *Let $S = \{xy = 1, x + xy = 0\}$ be a set of real constraints. From S, the equations $x+1=0$ and $y+1=0$ are obtained successively by applying some trivial substitutions.*

The main problem remains to prove that less dependent terms give tighter intervals, which is not always the case. Consider for example two variables, x with domain $[-0.5, 0.5]$ and y with domain $[-1,1]$. Though y is less dependent than $x*y$, the evaluation of y, being $[-1,1]$, is larger than the evaluation of $x*y$, being $[-0.5,0.5]$. As a consequence, the comparison between terms also depends on the variables domains. However, if the resulting constraints are added as redundancies, the computed intervals are always tighter, while this may slow down the whole process due to an amount of computations for these constraints.

Classically, branching is done by partitioning the domains of variables. Moreover, it can be applied on constraints, which reduces the dependency.

Example 7 (Disjunction) *Let $S = \{xy = 0, x^2 = y\}$ be a set of real constraints. From S, it can be generated the disjunction of $S' = \{x = 0, x^2 = y\}$ and $S'' = \{y = 0, x^2 = y\}$. The resolution of S is not immediate, which is due to the weak reductions computed with $x * y = 0$, while solving S' and S'' is trivial.*

Each presented method seems to be useful for reducing the dependency between variables. However, as seen in the experimental results, they cannot always be applied on the initial constraints. Actually, we think that what makes powerful the symbolic transformations of constraints is the combination of these methods. A particularly efficient strategy is to factorize the redundancies[3], obtained for example by Gröbner bases computations, and to generate disjunctions of constraints from factorized equations.

4 Modelling branch and prune algorithms

Branch and prune algorithms aim at searching solutions of sets of formulae. They are described as an iteration of two steps: from a set of formulae, a pruning step enforces some reductions of the variable domains using local methods, *i.e.* restricted to a subset of the formulae; these local methods generally being incomplete, a branching step enumerates the domains of variables. In the case of finite domains, the problem in the worst case is NP-hard (when the pruning does not remove any value from the domains, each element of the search space has to be enumerated).

We describe now a generic branch and prune algorithm for processing *real constraint systems* in terms of *constraint narrowing operators* and *constraint branching operators*. Intuitively, pruning is applying closure operators on constraint systems and branching is generating disjunctions of constraint systems.

Due to practical considerations, *i.e.* the handling of disjunctive constraints (see section 5), a constraint system is viewed as a conjunction of disjunctions of constraints.

Definition 6 (Constraint system) *Let A be an approximate domain over $\mathbb{R}$. A* constraint system *over A is a pair $(C, \vec{X})$ where C is the conjunction $C_1, \ldots, C_m$ and each C_i is the disjunction of real constraints $c_{i,1} \cup \ldots \cup c_{i,l_i}$. $\vec{X}$ is a finite Cartesian product over A denoting the domains of the variables appearing in C.*

Given an approximate domain A over $\mathbb{R}$, $\mathcal{C}_{(\Sigma,A)}$ denotes the set of all the constraint systems made from Σ and A. The set of solutions of a constraint system $S = (\{c_{1,1} \cup \ldots \cup c_{1,l_1}, \ldots, c_{m,1} \cup \ldots \cup c_{m,l_m}\}, \vec{X})$, denoted $S^\star$, is obtained by making the union of the relations in each disjunction and intersecting the results for all the disjunctions, that is $\cap_{i=1}^{m} \cup_{j=1}^{l_i} c_{i,j}$.

[3] When it is possible.

Constraint systems are generally ordered with respect to the inclusion of domains. However, we claim that it is important to take the constraint part into account, considering that numerical methods are dependent on the forms of constraints. For example, adding a redundant constraint to a constraint system reduces the search space. We define the relation $\subseteq$ over $\mathcal{C}_{(\Sigma,A)}$ to order constraint systems, as an extension of the relation proposed in [4]. Intuitively, c and c' being two relations, $c \cap c'$ is included in c (or c') while $c \cup c'$ contains c (or c'). More formally,

$$\begin{array}{c}(\{C_1,\dots,C_m\},\vec{X}) \subseteq (\{C'_1,\dots,C'_p\},\vec{X}') \iff \\ \vec{X} \subseteq \vec{X}' \quad \wedge \quad \forall i \in \{1,\dots,m\}\ \exists j \in \{1,\dots,p\} \\ (\forall k \in \{(j,1),\dots,(j,l_j)\}\ \exists h \in \{(i,1),\dots,(i,l_i)\})\ \ c_k = c_h\end{array}$$

The partially ordered set $(\mathcal{C}_{(\Sigma,A)}, \subseteq)$ is a complete lattice. Given two constraint systems $S = (C, \vec{X})$ and $S' = (C', \vec{X}')$ over the approximate domain A, the meet of S and S' is $(C \cup C', \vec{X} \cap \vec{X}')$ and the join is $(C \cap C', \underline{apx}_A(\vec{X} \cup \vec{X}'))$.

A pruning operation is modeled by *constraint narrowing operators*, which are closure operators over $(\mathcal{C}_{(\Sigma,A)}, \subseteq)$. A branching step is modeled by *constraint branching operators* which aim at generating disjunctions of constraint systems.

Definition 7 (Constraint narrowing / branching operators) *Given a structure Σ and an approximate domain A over* $\mathbb{R}$, *a* constraint narrowing operator *is a mapping $N : \mathcal{C}_{(\Sigma,A)} \to \mathcal{C}_{(\Sigma,A)}$ such that*

$$\begin{array}{lll}\forall S \in \mathcal{C}_{(\Sigma,A)} & : N(S) \subseteq S & (\mathit{Contractance}) \\ \forall S, S' \in \mathcal{C}_{(\Sigma,A)} & : S \subseteq S' \Rightarrow N(S) \subseteq N(S') & (\mathit{Monotonicity}) \\ \forall S \in \mathcal{C}_{(\Sigma,A)} & : N(S)^\star = S^\star & (\mathit{Correctness})\end{array}$$

A constraint branching operator *is a mapping $B : \mathcal{C}_{(\Sigma,A)} \to 2^{\mathcal{C}_{(\Sigma,A)}}$ such that for every S, $B(S) = \{S_1,\dots,S_n\}$, the following properties hold:*

$$\begin{array}{rl}\forall i \in \{1,\dots,n\} : S_i \subseteq S & (\mathit{Contractance}) \\ \cup_{i=1}^{n} S_i^\star = S^\star & (\mathit{Correctness})\end{array}$$

In this model, the resolution of constraint systems is defined as an iteration of applications of constraint narrowing operators and constraint branching operators, until none domain of the generated constraint systems can be further reduced. The result is a union of irreducible domains representing the solutions of the initial constraint system. The implemented methods generally being incomplete, the real solutions are guaranteed to be included in the computed domains while a computed non empty domain may contain no solution. However, the consideration of constraints in the definition of the relation $\subseteq$ disables the termination of classical algorithms. Then, this problem has to be handled for each practical application. For example, constraint narrowing operators adding an infinity of redundant constraints without ever reducing the domains must be excluded.

A constraint narrowing operator is an abstract description of a constraint solver. Furthermore, a combination of constraint narrowing operators over different approximate domains over $\mathbb{R}$ is still a constraint narrowing operator over the union of the approximate domains (see [4] for more details). This allows to describe constraint solving systems in which several solvers cooperate.

5 Solving polynomial systems

We propose to combine partial Gröbner bases (section 5.2), factorizations and substitutions (section 5.5), constructive disjunction (section 5.4), branching over domains and constraints (section 5.3) and interval Newton methods (section 5.1) for solving systems of polynomial equations over the reals, embedded in a branch and prune algorithm described in section 5.6.

Since the domains of computations are different, rationals *vs.* intervals, we need to merge these domains. Let $\mathbb{D}$ be the set $\{\mathbb{R}, \emptyset, \mathrm{q}_1, \mathrm{q}_2, \ldots\}$ where the q_i's are the rationals. An immediate result is that $\mathbb{D}$ is an approximate domain. It follows that $\mathbb{D} \cup \mathbb{I}$ is an approximate domain which will be the computational domain of the cooperative system. The inclusion and intersection over $\mathbb{D} \cup \mathbb{I}$ are the set operations over $\mathbb{R}$.

5.1 Interval Newton methods

Interval Newton methods [32, 2, 35, 7, 14, 18, 39, 22, 27] are numerical algorithms enclosing zeros of functions by intervals. We present here the methods combined in `Newton` [39]. They are embedded in an iterative algorithm derived from AC-3 [28, 5], computing an approximation over intervals of arc-consistency which is a local consistency notion from Artificial Intelligence. It stops when the domains cannot be further reduced, when they are canonical or smaller than the desired precision.

The first one is based on an extension over intervals of the well-known Newton root finding method over the real numbers. This method has been extended to interval functions (see [32, 19, 2, 18, 27, 35]). Let f be a real function. Let X be an interval and suppose that F is the natural interval extension of f, F' the natural interval extension of f', and $m(X)$ the approximation of the center of X. The Newton interval function is the function:

$$N(X) = m(X) \ominus (F(m(X)) \oslash F^{'}(X))$$

From this definition, one can design an *interval Newton method* enclosing roots of interval functions. Given an initial interval X_0 and an interval function F, a sequence of intervals $X_1, X_2, \ldots, X_n$ is computed using the iteration step $X_{i+1} = N(X_i) \cap X_i$. X_n is either empty, which means that X_0 contains no zero of F, or is a fixed point of N. If the function f contains several variables then this method is applied on each projection (by replacing each variable except the one considered by its domain) of F and F'.

The second method is derived from the mean value form [32] where the function is approximated using a Taylor expansion of order 1 around the center of the domains of the variables. Given $f(x_1, \cdots, x_n)$ a real function, $F(X_1, \cdots, X_n)$ its natural interval extension, J a set of intervals $\{J_1, \cdots, J_n\}$ for $X_1, \cdots, X_n$ and $m_i(J)$ the projection on i of $m(J)$, the following equality holds:

$$F(m(J)) \oplus \sum_{i=1}^{n} \frac{\partial F}{\partial x_i}(J) \otimes (X_i \ominus m_i(J)) = 0$$

If it is projected on the variable x_i, then the interval for the variable can be expressed as:

$$X_i = m_i(J) \ominus \frac{1}{\frac{\partial F}{\partial x_i}}[F(m(J)) \oplus \sum_{k=1,k\neq i}^{n} \frac{\partial F}{\partial x_k}(J) \otimes (J_k \ominus m_k(J))]$$

This expression gives a method to narrow down the interval for x_i using the Taylor narrowing operator, computing a new interval for x_i as $J_i \cap X_i$.

5.2 Partial Gröbner bases

Gröbner bases computations [10] are successive computations of S-polynomials (see section 2.3). Benhamou et als [6] proposed to preprocess constraint systems using Gröbner bases to reduce the dependency between variables. They introduced a notion of hierarchy between S-polynomials based on the notion of depth of a S-polynomial.

Definition 8 *Let S be an ordered set of polynomials. The* S-sets *of S are ordered sets of S-polynomials computed from S, and defined as follows:*

$$\begin{cases} S_0 = S \\ S_i = S_{i-1} \cup \{\overline{S(p,q)}^{S'} \neq 0 \mid p,q \in S_{i-1}\}, \quad i \geq 1 \end{cases}$$

where $\overline{S(p,q)}^{S'}$ is the S-polynomial of p and q reduced with respect to S'. S' is the set of all the polynomials appearing in S_i before $S(p,q)$. S_i is called the S-set of S of depth i.

The computation of S-sets stops when two consecutive S-sets are equals, i.e. none nonzero S-polynomial can be computed, which is guaranteed by the classical termination proofs of Gröbner bases computations using the idea that only a finite number of nonzero S-polynomials can be computed. If S is a set of polynomials and n is a natural such that $S_{n+1} = S_n$, then S_n is a Gröbner basis for S.

We give now a basic example of Gröbner bases computations structured by S-sets.

Example 8 *Let $S = \{x^3 - 2xy, x^2y - 2y^2 + x\}$ be a set of polynomials. The S-sets of S are computed using the reverse lexicographic ordering:*

$$\begin{array}{l} S_0 = \{x^3 - 2xy, x^2y - 2y^2 + x\} \\ S_1 = S_0 \cup \{x^2\} \\ S_2 = S_1 \cup \{2xy, 2y^2 - x\} \end{array}$$

S_2 is a Gröbner basis for S. The computation of S_1 adds the trivial equation $x^2 = 0$. Then, it is not useful to compute the whole Gröbner basis.

Notations: if $\{q_1, \ldots, q_m\}$ is a S-set of $\{p_1, \ldots, p_n\}$, $\{q_1 = 0, \ldots, q_m = 0\}$ is also called a S-set of $\{p_1 = 0, \ldots, p_n = 0\}$.

Since the reduction of the dependency between variables does not guarantee more precise interval computations (this is a heuristic), the S-polynomials are added as redundancies in the constraint system.

5.3 Branching

We propose to split the constraint systems over domains and constraints. Given $j \in \{1,\ldots,n\}$, the constraint system $(C = \{C_1,\ldots,C_m\},\vec{X})$ where $C_i = c_{i,1} \cup \ldots \cup c_{i,l_i}$ and $X_j = [l,r]$:

- Branching over constraints:
  If $l_i > 1$, C_i is splitted, which generates for all $k \in \{1,\ldots,l_i\}$ the disjunction of constraint systems $(\{C_1,\ldots,C_{i-1},\{c_{i,k}\},C_{i+1},\ldots,C_m\},\vec{X})$.
- Branching over domains:
  Let $x_0,x_1,\ldots,x_p$ be an ordered sequence of floating point numbers where $x_0 = l$ and $x_p = r$ and let $X'_1 = [x_0,x_1],\ldots,X'_p = [x_{p-1},x_p]$. Given $k \in \{1,\ldots,p\}$, the disjunction of constraint systems
  $\cup_k(C,(X_1,\ldots,X_{j-1},X'_k,X_{j+1},\ldots,X_n))$ is created.

During the resolution, a step of branching over domains always splits the domain of greatest width (the width of a rational is 0 and the width of an interval is the difference of the bounds). In general, the chosen domain, being necessarily an interval, is splitted in two or three equal parts[4].

5.4 Constructive disjunction

Constructive disjunction was studied in the CLP community [24, 41] to efficiently handle disjunctive constraints. The main idea is to infer informations from disjunctive constraints, i.e. reducing the domains of variables, rather than creating a Prolog choice point.

We use a similar idea to handle disjunctive constraints to be processed by interval Newton methods. Given a disjunctive constraint $c_1 \cup \ldots \cup c_m$ and a box $\vec{X}$, a new box $\vec{X_i}$ is computed from c_i and $\vec{X}$, for all i in $\{1,\ldots,m\}$. The final computed domain is $\underline{apx}_{\mathbb{I}}(\vec{X_1} \cup \ldots \cup \vec{X_m})$.

The motivation is to generate from a constraint $f * g = 0$ in factorized form the disjunction $\{f = 0\} \cup \{g = 0\}$. Intuitively, f and g being less dependent than $f * g$, the computed domains from $f = 0$ and $g = 0$ are included in the computed domain from $f * g = 0$, and so does the approximation of both domains.

5.5 Factorization and substitution

Substitutions replace expressions in constraints by smaller expressions, in the sense that the interval evaluations become tighter. Our (very basic) approach is to transform constraints in the form $f(x_1,\ldots,x_n) = q$ where q is a rational, and to replace f by q in other constraints. This guarantees that the transformed constraints lead to smaller computed intervals. More formally, let $(\{\{f(x_1,\ldots,x_n) = q\},C_2\ldots,C_m\},\vec{X})$ be a constraint system. Then, f is replaced by q in all disjunctions of constraints from $C_2,\ldots,C_m$ containing f. In practice, some other cases are tested. For example, from a constraint $x = y^2$, x is replaced by y^2 in the other constraints sharing x and y. This allows to

[4] Due to an heterogeneous repartition of the floating point numbers, it should be better to split the domains considering the number of the floating point numbers they contain.

decrease the number of variables. We proved in another paper [17] that such transformations reduce the width of the computed intervals.

Factorization is implemented following the subdistributivity property of interval arithmetic which ensures that factorized expressions give more precise interval evaluations. Moreover, even if a constraint cannot be fully factorized, it remains useful to factorize some sub-expressions. It is connected with the concept of constructive disjunction and the generation of disjunctions of constraint systems by splitting the constraints. Actually, a disjunctive constraint is created when a constraint is factorized and is processed by constructive disjunction during interval computations. It is only during a branching step that the disjunctive constraints are splitted for generating several constraint systems. In the implementation, factorizations are computed using the `Maple` [1] primitive `factor`.

The following example illustrates these ideas. Let S be the constraint system $(\{x^3+xy^2+2y^2+x^2-x=1, x^2+y^2=1\}, \vec{X})$. The substitution of x^2+y^2 by 1 in the first equation gives $x^3+xy^2+y^2-x=0$ which can be factorized in $(x-1)(x^2+y^2+x)=0$. From this last equation is generated the disjunction of two constraint systems $S'=(\{x=1, x^2+y^2=1\}, \vec{X})$ and $S''=(\{x^2+y^2+x=0, x^2+y^2=1\}, \vec{X})$. The substitution of x by 1 in S' gives $y^2=1$. Then, the solutions of S' are derived immediately. The substitution of x^2+y^2 by 1 in S'' gives $x=-1$. The substitution of x by -1 in the second equation gives $y^2=0$. Then, the solution of S'' is trivially computed.

5.6 Cooperative algorithm

We implement a branch and prune algorithm for solving systems of polynomial equations. The branching step is essentially described in the section 5.3. A branching over constraints is always preferred. The pruning step is achieved by a cooperation of the techniques presented in the precedent sections. We describe the strategy which guides the pruning step.

Let $S_0=(C_0, \vec{X})$ be a constraint system. The pruning of S_0 is implemented by the next fifth steps, in the following order:

1. S_0 is simplified as much as possible applying substitutions. This generates the new system $S_1=(C_1, \vec{X})$.
2. Let d be a fixed integer and E be the set of equations from S_1 not appearing in a disjunction. The S-set E' of depth d of E is computed. This creates the new system $S_2=(C_2=C_1\cup E', \vec{X})$.
3. The trivial equations of the form $x_i=q$ not appearing in disjunctions are removed after setting the domain of x_i to $X_i\cap q$. If a domain is empty then the process stops, otherwise the system $S_3=(C_3, \vec{X}')$ is derived.
4. Factorizations are enforced. From each constraint c of C_3 being factorized in $p_1*\cdots*p_n=0$, the disjunction $\{p_1=0\}\cup\ldots\cup\{p_n=0\}$ is created. This generates the system $S_4=(C_4, \vec{X}')$.
5. Interval Newton methods combined with constructive disjunction process each disjunctive constraint in C_4, until reaching a stable state.

The algorithm stops when the domains of each generated constraint systems cannot be further reduced or are smaller than the desired precision.

	Parameters			Branching		Computation time				
	n	v	s	$Cstr$	Int	t_S	t_I	$t_S + t_I$	t_{I-S}	$gain$
$parabola$	2	2	2	2	0	0.01	0	0.01	0.10	10
$Morgan$	4	4	2	1	1	0.04	0.17	0.21	7.80	37
$Griewank$	2	2	1	1	0	0.02	0	0.02	3.84	192
$cubic$	2	2	3	0	2	0.01	0.05	0.06	0.12	2
$chemistry$	4	4	1	0	0	0.34	0.25	0.59	0.95	1.6
$kinematics$	8	8	16	1	14	0.02	4.10	4.12	13.72	3
$high-deg$	3	3	12	6	8	0.36	0.42	0.78	10.40	13
$Powell$	4	4	1	1	0	0.02	0	0.02	0.33	16
$Brown$	5	5	2	0	1	0.04	0.19	0.23	∞	$\nearrow$
$Eiger$	4	4	2	0	1	0.02	0.15	0.17	0.34	2
$Kearfott$	9	9	2	2	0	0.02	0.17	0.19	0.42	2
$geometric$	2	2	2	0	1	0.06	0.23	0.29	0.50	2
$cyclohexane$	3	3	16	0	15	0.55	2.94	3.49	8.35	2.4
$cyclic_3$	3	3	2	0	1	0.01	0.10	0.11	∞	$\nearrow$
$cyclic_4$	4	4	4	0	7	0.16	0.97	1.13	∞	$\nearrow$
Cox	3	3	5	2	8	0.02	0.24	0.26	∞	$\nearrow$
$Geisow$	2	2	3	2	1	0.09	0.30	0.39	1.70	4
$piano$	9	9	1	1	0	0.01	0.51	0.52	0.68	1.3
$Czapor$	3	3	2	0	3	0.13	0.40	0.53	5.44	10
$Winkler$	3	3	2	0	1	0.09	0.44	0.53	2.51	5
eco_4	4	4	1	1	1	0.02	0.05	0.07	1.10	15
eco_5	5	5	4	1	9	0.75	0.88	1.65	∞	$\nearrow$
$neuro_1$	6	6	8	8	0	0.15	0	0.15	∞	$\nearrow$
$neuro_2$	6	6	8	0	13	0.47	1.36	1.83	∞	$\nearrow$
$combustion$	10	10	4	2	92	0.02	9.88	9.90	5.74	0.6
$interval_1$	10	10	1	0	0	4.26	1.02	5.28	0.66	0.1
$bifurcation$	3	3	4	0	88	10.35	1.32	11.67	1.09	0.1

Table 1: Experimental results.

6 Experimental results

We have implemented in the `C` language a prototype for solving non-linear polynomial equations, called `Cosinus`. The library `Gnu-Multi-Precision` [16] implements the computations over the rational numbers. `Maple` [1] implements the factorizations. The solver is tested on various examples extracted from [21, 34, 25, 42, 8, 13, 9, 33, 30, 43, 39] (see appendix A for their descriptions).

6.1 Computational results

The table 1 presents the computational results of `Cosinus` on the benchmarks. For each benchmark named in the first column, the labels of the other columns mean: n is the number of constraints in the system – v the number of variables – s the number of solutions – $Cstr$ the number of branching over constraints – Int

the number of branching over domains – t_S the computation time of symbolic methods – t_I the computation time of interval methods – t_S+t_I the sum of t_S and t_I – t_{I-S} the computation time when symbolic transformations are disconnected – *gain* the ratio between t_{I-S} and $t_S + t_I$.

The tests were done on a Sun Sparc 4 (110 Mhz) and the precision is 10^{-12} (the width of the computed intervals must be less than 10^{-12}). The branching over domains splits the greatest domain in two equal parts with respect to the Haussdorf metric. A "∞" in a cell of the table indicates that the computation time takes more than one hour. A "$\nearrow$" means that the ratio between t_{I-S} and $t_S + t_I$ is rather large when $t_{I-S} = \infty$.

6.2 Positive examples of problem solving

In this section, we describe the solving processes for some benchmarks which take advantage of symbolically transforming the initial constraints.

6.2.1 Factorizations and disjunctions

Let $S = (\{1.8125z_1^3 - 2z_1z_2 = 0, z_2 = z_1^2\}, \vec{X})$ be a constraint system where $\vec{X} = ([-10^3, +10^3], [-10^3, +10^3])$ (referred as *Griewank* in the table 1). The variable z_1 is factorized in the first equation. A branching step on constraints generates the disjunction:

$$S' = (\{z_1 = 0, z_2 = z_1^2\}, \vec{X}) \cup S'' = (\{1.8125z_1^2 - 2z_2 = 0, z_2 = z_1^2\}, \vec{X})$$

In the second equation of S', z_1 is replaced by 0 which gives $z_2 = 0$. In the first equation of S'', z_2 is replaced by z_1^2 which gives $-0.1875z_1^2 = 0$ and then $z_1 = 0$. The second equation is then transformed in $z_2 = 0$. Finally, the solution $(0, 0)$ is symbolically derived in $0.02s$.

Actually, the solution is singular and interval Newton methods are very inefficient for this kind of problem, and thus must operate on another expression of the system.

6.2.2 Gröbner bases

Let S be the constraint system (referred as $neuro_2$ in the table 1):

$$S = \begin{cases} x_5x_3^3 + x_6x_4^3 & = \ \ 3 \\ x_5x_1^3 + x_6x_2^3 & = \ \ 2 \\ x_1^2 + x_3^2 & = \ \ 1 \\ x_2^2 + x_4^2 & = \ \ 1 \\ x_5x_1x_3^2 + x_6x_4^2x_2 & = \ \ 1 \\ x_5x_1^2x_3 + x_6x_2^2x_4 & = -1 \\ x_i \in [-10^8, +10^8] \end{cases} \qquad P = \begin{cases} x_1^4 - \frac{2183}{2738}x_1^2 + \frac{37}{2738} & = 0 \\ x_5x_1^3 - \frac{16}{17}x_5x_1 + \frac{7}{17} & = 0 \\ x_5x_2^2x_1 - x_5x_1^3 - 3x_2^2 + 2 & = 0 \\ x_6x_2 + x_5 * x_1 - 3 & = 0 \end{cases}$$

S-set computations allow to generate the set of polynomial equations P in triangular form. Interval Newton methods are particularly efficient on $S \cup P$. In fact, the solutions for x_1 are computed first using the redundant equation over x_1, then the solutions for x_2 ...

6.2.3 Gröbner bases, factorizations and disjunctions

The initial equations cannot often be factorized. However, this is not the case for some redundancies from Gröbner bases if they have some rational roots. We describe the resolution of the *Cox* benchmark containing the equations:

$$\begin{cases} x+y+z^2=1 \\ x+y^2+z=1 \\ x^2+y+z=1 \end{cases}$$

The computed S-set contains the two polynomial equations:

$$\begin{cases} z^6-4z^4+4z^3-z^2=0 \\ y^2-y-z^2+z=0 \end{cases}$$

The first one is factorized in $z^2(z^4-4z^2+4z-1)$ and the following disjunction is generated:

$$S_1=\begin{cases} x+y+z^2=1 \\ x+y^2+z=1 \\ x^2+y+z=1 \\ y^2-y-z^2+z=0 \\ z=0 \end{cases} \bigcup \quad S_2=\begin{cases} x+y+z^2=1 \\ x+y^2+z=1 \\ x^2+y+z=1y^2-y-z^2+z=0 \\ z^4-4z^2+4z=1 \end{cases}$$

In S_1, the domain of z becomes 0 and then z is replaced in the constraints by 0. The equation $y^2-y=0$ is factorized in $y(y-1)=0$. This allows to generate the disjunction:

$$S_1'=\begin{cases} x+y=1 \\ x+y^2=1 \\ x^2+y=1y=0 \\ z=0 \end{cases} \bigcup \quad S_2'=\begin{cases} x+y=1 \\ x+y^2=1 \\ x^2+y=1y=1 \\ z=0 \end{cases}$$

The same operations on S_1' and S_2' lead to symbolically compute the solutions $(1,0,0)$ from S_1' and $(0,1,0)$ from S_2'. Finally, S_2 is solved by interval Newton methods which compute the approximations of the other three solutions $(-1-\sqrt{2},-1-\sqrt{2},-1-\sqrt{2}),(-1+\sqrt{2},-1+\sqrt{2},-1+\sqrt{2}),(0,0,1)$ (the boxes containing these solutions).

6.3 Negative examples

In this section, we present some examples for which the symbolic transformations slow down the whole process.

6.3.1 High convergence of interval methods

The benchmark $interval_1$ is composed of a set of ten non-linear equations over ten variables, and have a complex syntax. However, interval Newton methods efficiently operate on them and the only solution is computed in about half a second. Then, even if some good redundancies are computed in Gröbner bases (this is not the case), the whole computation time would not be greatly improved.

6.3.2 Slow convergence of symbolic methods

The benchmark *bifurcation* consists in three equations over three variables and is solved by interval methods in $1.09s$. The Gröbner basis of the three polynomials with respect to the lexicographic ordering contains an univariate polynomial $\alpha_8 z^8 + \alpha_7 z^7 + \alpha_6 z^6 + \alpha_5 z^5 + \alpha_4 z^4 + \alpha_3 z^3 + \alpha_2 z^2 + \alpha_1 z + \alpha_0$ over which interval Newton is very efficient. However, the computation of the Gröbner basis takes more than $1.09s$ and then is useless. Actually, the rational coefficients of the S-polynomials become very large and their handling slows down the computation. For example:

$$\alpha_3 = \frac{41121004731883258983258747921732678836438053791641108480}{103568706741101039995381188457060198437134092773759}$$

6.3.3 Complexity of redundancies

The benchmark *combustion* is a chemistry problem described by a set of non-linear equations. It is immediate to obtain an univariate equation in x_4 from $x_4 + 2x_7 = 10^{-5}$ and $0.7816278.10^{-15}x_7 = x_4^2$ by replacing x_7 by $(10^{-5} - x_4)/2$ in the second one. Then, this equation is factorized and a disjunction of two systems is generated, each one containing two solutions. Such a (simple) transformation leads to an improvement of the computation time (about $1.2s$ instead of $5.74s$). However, the implemented strategy of the cooperative algorithm imposes to compute a S-set of fixed (by hand) depth. Unfortunately, none simple (in the sense it is efficiently handled by interval methods) redundancy is computed and the total computation time becomes $9.90s$.

6.4 Discussion

We identify several reasons which make the cooperation of symbolic and numerical algorithms inefficient. The redundant equations from S-sets may not improve the convergence of interval methods, due to a fast convergence with initial systems or to the S-polynomials complexity (great number of variables or occurrences of variables ...). Furthermore, it may be hard to compute some interesting S-polynomials and the computation time of S-sets may become too large with respect to the computation time of interval methods. Finally, the strategy in the cooperative algorithm is not flexible enough. The *combustion* problem needs for the application of factorizations and disjunctions while Gröbner bases slow down the process.

To remedy to such problems, we plan to study a data-parallel approach of our algorithm together with some flexible, dynamic, cooperation strategies. An immediate, trivial strategy will be to process interval and symbolic methods over two different processors. In this framework, the symbolic algorithm will stop the interval solving process when some "interesting" redundancies are computed. For this purpose, some criteria to decide if a constraint is "interesting" need to be defined. Then, a synchronisation will operate and the constraints to be processed by interval methods will be modified with respect to the added redundancies. Finally, what is needed is some criteria to stop the computations during a pruning step, before applying a branching.

Another approach is to treat conjunctions (and-parallelism in the literature) and disjunctions (or-parallelism) in data-parallel mode. We will study the dynamic interaction between the amount of parallelism generated in this manner and the cost of the load-balancing required to maintain proportional speedups. Particular attention will be given to the BSP (bulk-synchronous parallel) cost-prediction formulas as dynamic indicators for steering data-parallel solver cooperation.

7 Conclusion and future work

We have presented a branch and prune algorithm for solving polynomial systems, implementing symbolic and numerical pruning and branching. In particular, factorizations, substitutions and partial Gröbner bases enforce simplifications of systems while interval Newton methods reduce the domains of variables. Experimental results show that interval Newton methods generally benefit from the symbolic transformations of constraint systems.

Except what we argue in the above discussion, our future work will take other directions. Interval Newton methods are particularly efficient when Jacobian matrices are diagonally dominant. However, matrices become non square due to the computation of redundancies, which disables preconditionning algorithms that we are aware of. This problem has to be studied. Our Gröbner bases algorithm can be improved by implementing other computation strategies (for example the *sugar cube* [15] strategy). Another approach is to compute Gröbner bases dealing with uncertain data and one may explore Gröbner bases computations for systems of polynomials having interval coefficients. The theory of singularity from computer algebra constantly deals with systems leading to troubles of classical methods. It can inspire another way to simplify polynomial systems. Finally, we plan to experiment some strategies of branching over continuous domains, referred as heuristics on variable ordering for finite domains from the Artificial Intelligence community.

Acknowledgements

We are grateful to Deepak Kapur who gave us the novel ideas about substitutions, factorizations and disjunctions. We are indebted to Frédéric Benhamou, Gérard Ferrand and Gaétan Hains for numerous discussions and their continuous help during this work, and the anonymous referees for their comments.

References

1. M. Abell and J. Braselton. *The Maple V Handbook.* Academic Press, 1994.
2. G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, 1983.
3. G. Alefeld and R. Lohner. On Higher Order Centered Forms. *Computing*, 35:177–184, 1985.
4. F. Benhamou. Heterogeneous Constraint Solving. In *Proceedings of ALP'96*, volume 1139 of *LNCS*, pages 62–76, Aachen, Germany, 1996. Springer-Verlag.

5. F. Benhamou and L. Granvilliers. A Fixpoint Approximate Semantics for Cooperating Numerical Solvers. *Reliable Computing, Proceedings of INTERVAL'96*, pages 20–22, 1996. (Supplement).
6. F. Benhamou and L. Granvilliers. Automatic Generation of Numerical Redundancies for Non-Linear Constraint Solving. *Reliable Computing*, 3(3):335–344, 1997.
7. F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
8. D. Bini and B. Mourrain. Handbook of Polynomial Systems. November 1996.
9. C. Bliek. *Computer Methods for Design Automation.* PhD thesis, MIT, 1992.
10. B. Buchberger. Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory. In *Multidimensional Systems Theory*, pages 184–232. 1985.
11. G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings of the 2nd GI Conference*, volume 33 of *LNCS*, pages 134–183. Springer, Berlin, 1975.
12. G. E. Collins and H. Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.
13. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms.* Springer-Verlag, 1992.
14. E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.
15. A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. One sugar cube, please, or selection strategies in the buchberger algorithm. In *Proceedings of ISSAC'91.* ACM Press, 1991.
16. T. Granlund. *GNU Multiple Precision Arithmetic Library, edition 2.0*, 1996.
17. L. Granvilliers. Transformations symboliques et consistance de bloc de CSP continus. In *Proceedings of JFPLC'97*, pages 195–209. Hermès, 1997.
18. E. R. Hansen and R. I. Greenberg. An Interval Newton Method. *Applied Mathematics and Computation*, 12:89–98, 1983.
19. E. R. Hansen and S. Sengupta. Bounding Solutions of Systems of Equations using Interval Analysis. *BIT*, 21:203–211, 1981.
20. H. Hong. Confluency of Cooperative Constraint Solving. Technical Report 94-08, RISC-Linz, Johannes Kepler University, Linz, Austria, 1994.
21. H. Hong and V. Stahl. Safe Start Region by Fixed points and Tightening. *Computing*, 53(3-4):323–335, 1994.
22. E. Hyvönen. Constraint Reasoning based on Interval Arithmetic. The Tolerance Propagation Approach. *Artificial Intelligence*, 58:71–112, 1992.
23. IEEE. IEEE Standard for Binary Floating-Point Arithmetic. Technical Report IEEE Std 754-1985, 1985. Reaffirmed 1990.
24. J. Jourdan and T. Sola. The Versatility of Handling Disjunctions as Constraints. In *Proceedings of PLILP'93*, pages 60–74, 1993.
25. R. B. Kearfott. Some Tests of Generalized Bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, 1987.
26. R. B. Kearfott and X. Shi. Optimal Preconditionners for Interval Gauss-Seidel Methods. In *Scientific Computing and Validated Numerics*, pages 173–178, 1996.
27. R. Krawczyk. A Class of Interval Newton Operators. *Computing*, 37:179–183, 1986.
28. A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.
29. P. Marti and M. Rueher. A Distributed Cooperating Constraints Solving System. *International Journal on Artificial Intelligence Tools*, 4(1):93–113, 1995.
30. E. Monfroy. Gröbner Bases: Strategies and Applications. In *Proceedings of AISMC'92*, volume 737 of *LNCS*, pages 133–151. Springer-Verlag, 1992.
31. E. Monfroy, M. Rusinowitch, and R. Schott. Implementing Non-Linear Constraints with Cooperative Solvers. In *Proceedings of ACM Symposium on Applied Computing*, pages 63–72. ACM Press, February 1996.

32. R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.
33. A. Morgan and A. Sommese. Homotopy Methods for Polynomial Systems. *Applied Mathematics and Computation*, 24:115–138, 1987.
34. A. Morgan, A. Sommese, and C. Wampler. Computing Singular Solutions to Polynomial Systems. *Advances in Applied Mathematics*, 13:305–327, 1992.
35. A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, 1990.
36. H. Ratschek. Centered forms. *SIAM Journal on Numerical Analysis*, 17(5):656–662, 1980.
37. H. Ratschek and J. Rokne. About the centered form. *SIAM Journal on Numerical Analysis*, 17:333–337, 1980.
38. V. Saraswat. Concurrent Constraint Programming. In *Proceedings of POPL'90*, pages 232–245, 1990.
39. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.
40. P. Van Hentenryck, L. Michel, and F. Benhamou. `Newton` - Constraint Programming over non-linear Constraints. *Science of Programming*, 1997. (Forthcoming).
41. P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, Implementation and Evaluation of the Constraint Language `cc(fd)`. In *Constraint Programming: Basics and Trends*, volume 910 of *LNCS*. Springer-Verlag, 1994.
42. M. Vrahatis. Solving Systems of Nonlinear Equations Using the Nonzero Value of the Topological Degree. *ACM Transactions on Mathematical Software*, 14(4):312–329, 1988.
43. F. Winkler. *Polynomial Algorithms in Computer Algebra.* Springer-Verlag, 1996.

A Appendix: Benchmarks

***parabola* [21]: geometric intersection problem**

$$\begin{cases} x^2 + y^2 = 1 \\ x^2 = y \\ x, y \in [-10^8, +10^8] \end{cases}$$

***Morgan* [21]: kinematics problem**

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 1 \\ x_1 + x_2 - x_3 + x_4 = 3 \\ x_1^2 + x_2^2 + x_3^2 + x_4^2 = 4 \\ x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2x_1 = 3 \\ x_i \in [-10^3, +10^3] \end{cases}$$

***Griewank* [34]: Griewank and Osborne's system**

$$\begin{cases} 1.8125 z_1^3 - 2 z_1 z_2 = 0 \\ z_2 = z_1^2 \\ z_i \in [-10^3, +10^3] \end{cases}$$

***cubic* [25]: intersection cubic-parabola**

$$\begin{cases} 4x_1^3 - 3x_1 - x_2 = 0 \\ x_1^2 = x_2 \\ x_i \in [-2, +3] \end{cases}$$

***chemistry* [25]: combustion chemistry problem**

$$\begin{cases} -1.697.10^7 x_2 x_4 + 2.177.10^7 x_2 \\ \quad +0.55 x_1 x_4 + 0.45 x_1 - x_4 = 0 \\ 1.585.10^{14} x_2 x_4 + 4.126.10^7 x_1 x_3 \\ \quad -8.285.10^6 x_1 x_4 + 2.284.10^7 x_3 x_4 \\ \quad -1.918.10^7 x_3 + 48.4 x_4 = 27.73 \\ x_1^2 = x_2 \\ x_4^2 = x_3 \\ x_i \in [0, 1] \end{cases}$$

***kinematics* [25]: robot kinematics problem**

$$\begin{cases} 0.004731 x_1 x_3 - 0.3578 x_2 x_3 \\ \quad -0.1238 x_1 - 0.001637 x_2 \\ \quad -0.9338 x_4 + x_7 = 0.3571 \\ 0.2238 x_1 x_3 + 0.7623 x_2 x_3 \\ \quad +0.2638 x_1 - 0.07745 x_2 \\ \quad -0.6734 x_4 - 0.6022 = 0 \\ x_6 x_8 + 0.3578 x_1 + 0.004731 x_2 = 0 \\ 0.2238 x_2 - 0.7623 x_1 + 0.3461 = 0 \\ x_1^2 + x_2^2 = 1 \\ x_3^2 + x_4^2 = 1 \\ x_5^2 + x_6^2 = 1 \\ x_7^2 + x_8^2 = 1 \\ x_i \in [-1, +1] \end{cases}$$

***high − deg* [25]: high-degree polynomial system**

$$\begin{cases} 2x_1^2 x_2^3 - 2x_1^6 x_2 + 2x_2 x_3 = 0 \\ x_1^2 + x_2^2 - 0.265625 = 0 \\ 5x_1^9 - 6x_1^5 x_2^2 + x_2^4 x_1 + 2x_1 x_3 = 0 \\ x_1, x_2 \in [-0.6, +0.6] \\ x_3 \in [-5, +5] \end{cases}$$

***Powell* [25]: Powell's singular function**

$$\begin{cases} x_1 + 10x_2 = 0 \\ x_3 - x_4 = 0 \\ x_2^2 - 4x_2 x_3 + 4x_3^2 = 0 \\ x_1^2 - 2x_1 x_4 + x_4^2 = 0 \\ x_i \in [-2, +2] \end{cases}$$

***Brown* [25]: Brown's almost linear system**

$$\begin{cases} x_1 + x_1 + x_2 + x_3 + x_4 + x_5 = 6 \\ x_2 + x_1 + x_2 + x_3 + x_4 + x_5 = 6 \\ x_3 + x_1 + x_2 + x_3 + x_4 + x_5 = 6 \\ x_4 + x_1 + x_2 + x_3 + x_4 + x_5 = 6 \\ x_1 x_2 x_3 x_4 x_5 = 1 \\ x_i \in [-2, +2] \end{cases}$$

Eiger **[42]: extended Eiger-Sikorski-Stenger function**

$$\begin{cases} x_1^2 - 0.2x_1 + x_2 = 0.09 \\ x_2^2 - 0.2x_2 + x_3 = 0.09 \\ x_3^2 - 0.2x_3 + x_4 = 0.09 \\ x_4^2 - 0.2x_4 + x_1 = 0.09 \\ x_i \in [-10^2, +10^2] \end{cases}$$

Kearfott **[42]: extended function**

$$\begin{cases} x_1^2 = x_2 \\ x_2^2 = x_3 \\ x_3^2 = x_4 \\ x_4^2 = x_5 \\ x_5^2 = x_6 \\ x_6^2 = x_7 \\ x_7^2 = x_8 \\ x_8^2 = x_9 \\ x_9^2 = x_1 \\ x_i \in [-10^3, +10^3] \end{cases}$$

geometric **[33]: geometric intersection problem**

$$\begin{cases} 978000z_2^2 - 0.00098z_1^2 - 9.8z_1z_2 \\ \quad -235z_1 + 88900z_2 = 1 \\ 0.00987z_1 - 0.984z_2^2 - 0.01z_1^2 \\ \quad -29.7z_1z_2 - 0.124z_2 = 0.25 \\ z_i \in [-10^8, +10^8] \end{cases}$$

cyclohexane **[8]: 3-dimensional structure of the molecule**

$$\begin{cases} 13 + y^2 + z^2 - 24yz + y^2z^2 = 0 \\ 13 + z^2 + x^2 - 24xz + z^2x^2 = 0 \\ 13 + x^2 + y^2 - 24xy + x^2y^2 = 0 \\ x, y, z \in [-10^2, +10^2] \end{cases}$$

cyclic$_3$ **[8]: variant of Cyclic n-roots [3]**

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1x_2 + x_2x_3 - x_3x_1 = 0 \\ x_1x_2x_3 = 1 \\ x_i \in [-10^5, +10^5] \end{cases}$$

cyclic$_4$ **[8]: variant of Cyclic n-roots [4]**

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 0 \\ x_1x_2 + x_2x_3 + x_3x_4 - x_4x_1 = 0 \\ x_1x_2x_3 + x_2x_3x_4 + x_4x_1x_2 \\ \quad +x_3x_4x_1 = 0 \\ x_1x_2x_3x_4 = 1 \\ x_i \in [-10^5, +10^5] \end{cases}$$

Cox **[13]: test for Gröbner bases**

$$\begin{cases} x + y + z^2 = 1 \\ x + y^2 + z = 1 \\ x^2 + y + z = 1 \\ y \in [-10^8, +10^8] \end{cases}$$

Geisow **[9]: Geisow's multiple crunode equation**

$$\begin{cases} 36x^2 - 24x^3 - 38x - 12xy^2 \\ \quad +11y^2 = 0 \\ 22xy - 12x^2y + 6y - 16y^3 = 0 \\ x \in [-1, +3] \\ y \in [-2, +2] \end{cases}$$

piano **[30]: restriction of the piano mover's problem**

$$\begin{cases} x - l * t^3 - L * w = 0 \\ y - L * t - l * w^3 = 0 \\ L - 1 = 0 \\ l - 2 = 0 \\ x - a = 0 \\ 2 * a - 3 = 0 \\ y - b = 0 \\ b - r * t = 0 \\ w^2 - 1 + t^2 = 0 \\ x, y, l, t, L, b, r, a \in [-10^5, +10^5] \\ w \in [0, +10^5] \end{cases}$$

Czapor **[30]**

$$\begin{cases} zx^2 - 0.5x - y^2 = 0 \\ zy^2 + 2x + 0.5 = 0 \\ z - 16x^2 - 4xy^2 - 1 = 0 \\ x, y, z \in [-10^8, +10^8] \end{cases}$$

Winkler **[43]: test for Gröbner bases**

$$\begin{cases} 4xz - 4xy^2 - 16x^2 - 1 = 0 \\ 2y^2z + 4x + 1 = 0 \\ 2x^2z + 2y^2 + x = 0 \\ x, y, z \in [-10^5, +10^5] \end{cases}$$

***eco*$_4$** **[39]: economics problem**

$$\begin{cases} x_1x_4 + x_1x_2x_4 = 0.35 \\ x_2x_4 + x_1x_3x_4 = 1.086 \\ x_1 + x_2 + x_3 + 1 = 0 \\ x_3x_4 = 2.05 \\ x_i \in [-10^8, +10^8] \end{cases}$$

***eco*$_5$** **[39]: economics problem**

$$\begin{cases} x_1x_5 + x_1x_2x_5 + x_2x_3x_5 \\ \quad +x_3x_4x_5 = 3.55 \\ x_2x_5 + x_1x_3x_5 + x_2x_4x_5 = 0.35 \\ x_3x_5 + x_1x_4x_5 = 1.086 \\ x_4x_5 = 1 \\ x_1 + x_2 + x_3 + x_4 + 1 = 0 \\ x_i \in [-10^8, +10^8] \end{cases}$$

***neuro*$_1$** **[39]: neurophysiologic problem**

$$\begin{cases} x_5x_3^3 + x_6x_4^3 = 3 \\ x_5x_1^3 + x_6x_2^3 = 2 \\ x_1^2 + x_3^2 = 1 \\ x_2^2 + x_4^2 = 1 \\ x_5x_1x_3^2 + x_6x_4^2x_2 = 0 \\ x_5x_1^2x_3 + x_6x_2^2x_4 = 0 \\ x_i \in [-10^8, +10^8] \end{cases}$$

***neuro*$_2$** **[39]: neurophysiologic problem**

$$\begin{cases} x_5x_3^3 + x_6x_4^3 = 3 \\ x_5x_1^3 + x_6x_2^3 = 2 \\ x_1^2 + x_3^2 = 1 \\ x_2^2 + x_4^2 = 1 \\ x_5x_1x_3^2 + x_6x_4^2x_2 = 1 \\ x_5x_1^2x_3 + x_6x_2^2x_4 = -1 \\ x_i \in [-10^8, +10^8] \end{cases}$$

combustion **[39]: combustion problem for a temperature of 3000°**

$$\begin{cases} x_2 + 2x_6 + x_9 + 2x_{10} = 10^{-5} \\ x_3 + x_8 = 3.10^{-5} \\ x_1 + x_3 + 2x_5 + 2x_8 + x_9 \\ \quad +x_{10} = 5.10^{-5} \\ x_4 + 2x_7 = 10^{-5} \\ 0.5140437.10^{-7}x_5 = x_1^2 \\ 0.1006932.10^{-6}x_6 = 2x_2^2 \\ 0.7816278.10^{-15}x_7 = x_4^2 \\ 0.1496236.10^{-6}x_8 = x_1x_3 \\ 0.6194411.10^{-7}x_9 = x_1x_2 \\ 0.2089296.10^{-14}x_{10} = x_1x_2^2 \\ x_i \in [-10^8, +10^8] \end{cases}$$

***interval*$_1$** **[39]: classical interval benchmark**

$$\begin{cases} x_1 - 0.18324757x_4x_3x_9 \\ \quad = 0.25428722 \\ x_2 - 0.16275449x_1x_{10}x_6 \\ \quad = 0.37842197 \\ x_3 - 0.16955071x_1x_2x_{10} \\ \quad = 0.27162577 \\ x_4 - 0.15585316x_7x_1x_6 \\ \quad = 0.19807914 \\ x_5 - 0.19950920x_7x_6x_3 \\ \quad = 0.44166728 \\ x_6 - 0.18922793x_8x_5x_{10} \\ \quad = 0.14654113 \\ x_7 - 0.21180486x_2x_5x_8 \\ \quad = 0.42937161 \\ x_8 - 0.17081208x_1x_7x_6 \\ \quad = 0.07056438 \\ x_9 - 0.19612740x_{10}x_6x_8 \\ \quad = 0.34504906 \\ x_{10} - 0.21466544x_4x_8x_1 \\ \quad = 0.42651102 \\ x_i \in [-2, +2] \end{cases}$$

bifurcation **[8]**

$$\begin{cases} x^2 + y^2 - 0.265625 = 0 \\ 5x^8 - 6x^4y + y^4 + 2z = 0 \\ 2x^2y^2 - 2x^6 + 2z = 0 \\ x, y, z \in [-10, +10] \end{cases}$$