Automatic Generation of Numerical Redundancies for Non-linear Constraint Solving

Frédéric Benhamou and
Laurent Granvilliers
LIFO, Université d'Orléans, I.I.I.A., Rue Léonard de Vinci
B.P. 6759 45067 ORLEANS Cedex 2 France
{benhamou,granvil}@lifo.univ-orleans.fr

Key words: Numerical constraints, interval constraints, approximate solving, local consistency, propagation methods, interval Newton methods, Gröbner bases.

Abstract. In this paper we present a framework for the cooperation of symbolic and propagation-based numerical solvers over the real numbers. This cooperation is expressed in terms of fixed points of closure operators over a complete lattice of constraint systems. In a second part we instantiate this framework to a particular cooperation scheme, where propagation is associated to pruning operators implementing interval algorithms enclosing the possible solutions of constraint systems, whereas symbolic methods are mainly devoted to generate redundant constraints. When carefully chosen, it is well known that the addition of redundant constraint drastically improve the performances of systems based on local consistency (e.g. Prolog IV or Newton). We propose here a method which computes sets of redundant polynomials called partial Gröbner bases and show on some benchmarks the advantages of such computations.

1. Introduction

Several constraint programming languages and systems addressing numerical constraints are based on combinations and/or cooperations of different methods. Among others, one can cite `Prolog IV` [8], `CLP(BNR)` [5], `Newton` [4] based on interval methods and local consistency techniques. Most of these systems combine symbolic computations to transform the initial constraint system and interval-based techniques to compute the solutions. The solving process is based on local applications of operators reducing the domains of possible values for some variables followed by a search phase recursively applying the operators to selected sub-domains.

As it is well-known by the users of such languages and systems, one of the consequences of the local application of these operators is that the computational efficiency can be drastically improved by adding redundancies to the initial constraint set. These additions are generally performed by hand and require from the programmer a high-level knowledge of the constraint engine.

The aim of this paper is to design a framework to describe the constraint solving process that takes into account these symbolic transformations, to instantiate this framework to solve polynomial constraints over the reals and to show that the computation of some "useful" redundancies can be automated in this context.

To be general enough, the proposed framework must describe computations of cooperating solvers and provide a declarative semantics for the resolution process. However, due to size limitations, we restrict here our study to the case of continuous numerical constraint sets. The main idea is to consider a complete lattice of constraint systems and to represent the resolution process as a combination of particular closure operators over this lattice. Under some conditions on the constraint solvers, the corresponding algorithm, inspired from well-known work in Artificial Intelligence on Constraint Satisfaction Problems [15, 14] is shown to be correct, to terminate, to be strategy-independent and to compute a certain fixed point of these closure operators.

In the second part of the paper, this framework is used to describe a solver approximating the solutions of systems of nonlinear polynomial equations over the real numbers. This solver implements a combination of symbolic transformations, automatic addition of redundant constraints, interval Newton methods [16, 1] and enumeration techniques. The generation of redundant constraints is based on Gröbner bases techniques [7]. This technique being essentially developed to improve the computational behaviour of the solver, the main idea here is to reduce the combinatorial explosion by computing some particular sets of S-polynomials. Experimental results are provided to illustrate the approach.

2. Approximate resolution

In this section, we propose a framework to describe the cooperation of solvers for approximating the solutions of constraint systems over the real numbers using local consistency techniques. For a more detailed presentation of the material discussed in this section see [2].

Solving constraint systems using local consistency techniques consists essentially in iterating two main operations, domain reduction and propagation, until reaching a stable state. Roughly speaking, if the domain of a variable x is locally reduced with respect to a constraint then this domain modification is propagated to all the constraints in which x occurs, leading to the reduction of other variables' domains and so on. Being incomplete by nature, these methods have to be combined with enumeration techniques, for example bisection, to separate

the solutions when it is possible. Domain reduction relies on the notion of *constraint narrowing operators* computing over *approximate domains* over the real numbers.

Definition 1. An *approximate domain* A over $\mathbb{R}$ is a subset of the powerset of $\mathbb{R}$, closed under intersection, containing $\mathbb{R}$ and for which the inclusion is a well founded ordering.

Given an approximate domain A over $\mathbb{R}$ the possible values of a variable are represented by an element of A, i.e. a subset of $\mathbb{R}$. Given a finite set of n variables, the possible values of the variables appearing in a given constraint system are represented by an n-ary Cartesian product of elements of A (the set of such elements is denoted A_n).

Definition 2. Let A be an approximate domain over $\mathbb{R}$. Let ρ be an n-ary relation over $\mathbb{R}$. The function $N : A_n \rightarrow A_n$ is a *constraint narrowing operator* for the relation ρ iff for every $u, v \in A_n$, the three following properties hold:

$$\begin{array}{lll} (1) & u \cap \rho \subseteq N(u), & (Correctness) \\ (2) & N(u) \subseteq u, & (Contractance) \\ (3)\ u \subseteq v & implies\ N(u) \subseteq N(v). & (Monotonicity) \end{array}$$

As we will develop in the following, combinations of constraint narrowing operators can be viewed as abstract descriptions of solvers.

To take the operational aspects of constraint solving into account we define *Extended Constraint Systems*, by associating to each constraint a constraint narrowing operator.

Definition 3. Let A be an approximate domain over $\mathbb{R}$. An *Extended Constraint System* (ECS) is a pair (S, X) where $S = \{(C_1, N_1), (C_2, N_2) \ldots, (C_m, N_m)\}$ is a set of pairs made of a real constraint C_i and of a constraint narrowing operator N_i for C_i, and X is an n-ary Cartesian product of elements of A.

The algorithm for processing such systems is essentially an adaptation of AC-3 [14] and of the filtering algorithms used in interval constraint-based systems like BNR-Prolog [19], CLP(BNR) [5] or Newton [4, 20]. Given an initial ECS (S, X) it works by iterating the application of constraint narrowing operators until reaching a stable state as shown below.

`Propagation(`**in** $\{(C_1, N_1), \ldots, (C_m, N_m)\}$`;`**inout** $X = \times_{i=1}^{n} X_i$`)`
begin

```
      S := {C_1, ..., C_m}
      while S ≠ ∅ and X ≠ ∅ do
            Extract C_i from S
            X' := N_i(X)
            if X' ≠ X then
                  S := S ∪ {C_j | ∃v_k ∈ var(C_j) ∧ X'_k ≠ X_k}
                  X := X'
            endif
      endwhile
end
```

The main properties of the algorithms are:

1. it terminates,
2. it is correct (no solutions of the initial system are lost)
3. it is confluent (the output is strategy independent)
4. if the input is (S, X) and the output (S, X'), then X' is the greatest common fixed point of the narrowing operators $N_1, \ldots, N_m$ included in X.

In most cases, the reduction of domains intrinsically depends on the form of the constraints. For example, it is well known that different expressions of a real function gives different ranges of variations when lifted to interval functions. In all constraint systems, this leads to the implementation of a pre-processing step, in which symbolic transformations of the initial set are achieved. This step preserves the set of initial solutions and generates a "better" expression of the problem, with respect to the involved combination of algorithms. Here follows the complete algorithm. Given an initial constraint system as input, the algorithm computes the list *Final* of canonical domains.

```
Resolution( in (C, X) ; out Final )
begin
      (C', X) := Preprocess(C, X)
      (S, X) := Generate an ECS from (C', X)
      Add X in Current
      while Current ≠ ∅
            Extract X from Current
            X' := Propagation(S, X)
            if X' ≠ ∅ and X' is canonical
            then
                  Add X' in Final
```

```
        else
            X'_1, ..., X'_m := Bisection(X')
            ∀i ∈ {1, ..., m} Add X'_i in Current
        endif
    endwhile
end
```

3. Automatic generation of redundant constraints

Another well-known fact by the users of systems based on local consistency properties is that the clever addition of redundant constraints is often crucial to prune the search space. In this section, we describe a possible automation of this process by means of a partial computation of Gröbner bases.

In a previous paper [3], we have proposed to handle this problem by computing Gröbner bases either for the initial problem or for some well-chosen sub-parts of it. The principal drawbacks were the exponential complexity of the algorithm in the first case, leading to unrealistic execution times, and the difficulty to compute a relevant partitioning in the second case. We propose here an alternative method based on two parameters, depth and filtering. For self-containment purposes, we recall briefly some basics of Gröbner bases computations.

3.1. Gröbner bases

The definitions can be found in [7, 9]. 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 every polynomial is ordered wrt this ordering. For every polynomial p, the *leading term* of p, denoted $LT(p)$ is the monomial of p which is maximal wrt the monomial ordering.

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

Definition 4. Let p and q be nonzero polynomials. If x^γ is the least common multiple of the power products appearing in $LT(p)$ and $LT(q)$, then the S-polynomial of p and q is the combination

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

Given p a nonzero polynomial and $S = \{p_1, \ldots, p_n\}$ 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 which is divisible by any of $LT(p_1), \ldots, LT(p_n)$. r is obtained as the remainder of the division of p wrt S and is called the reduction of p wrt S, denoted $\overline{p}^S$.

The following definition for Gröbner bases gives their algorithmic characterization and is due to Buchberger [7]:

Definition 5. Let $S = \{p_1, \ldots, p_n\}$ be a set of polynomials. S is a Gröbner basis iff for all pairs $(i, j) \in \{1, \ldots, n\}$ the reduction wrt 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).

3.2. Partial Gröbner bases

We first introduce the notion of *depth of S-polynomials.*

Definition 6. Let S be an ordered set of polynomials. We define the ordered sets S_i of S-polynomials

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

where $S(p, q)$ is the S-polynomial of p and q. The set S_i is called the *S-set of depth i.*

The polynomials of S are ordered according to their order of appearance and the sets S_i are computed wrt this ordering. Since the number of nonzero S-polynomials that can be computed to reach a Gröbner basis is finite, S-set computations are finite, i.e. it exists n such that $S_n = S_{n-1}$.

Proposition 1. If n is such that $S_n = S_{n-1}$ and does not contain any constant S-polynomial then S_n is a Gröbner basis.

Inconsistency can be detected during S-set computations. By application of the Weak Nullstellensatz theorem, if a constant nonzero S-polynomial is computed the initial set of polynomials has no common root.

From a practical (and heuristic) point of view, the idea is to compute S-sets of a certain depth, verifying suitable properties. Such sets are called *partial Gröbner bases*. One of the problems is to experimentally determine a "reasonable" depth and adequate properties of the selected S-polynomials.

We give now a basic example of Gröbner bases computations related in terms of S-sets.

Example 1. 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 with $x > y$:

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

S_2 is a Gröbner basis for S. Computing S_1 from S is quite a good result for solving the equations $\{x^3 - 2xy = 0, x^2y - 2y^2 + x = 0\}$ because the resolution of $x^2 = 0$ is easy using eitheralgebraic or interval methods.

4. Combining partial Gröbner bases and box-consistency

We propose to instantiate the general scheme defined in section 2 for solving real polynomial equations.

The constraint systems are pre-processed by computing S-sets over the rational numbers. Then from the transformed system an ECS is generated and defined over intervals. The approximate domain we consider is the set made of all the closed floating point intervals.

The notion of box-consistency was introduced and formally defined in [4]. Without entering the details it characterizes the common fixed-point of narrowing operators acting over interval projections of multivariate polynomials. We use in our system the narrowing operators used in the `Newton` system. For a detailed presentation of these operators see [20].

Basically, 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 [16, 11, 1, 13, 18]. 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

$$N(X) = m(X) - F(m(X))/F^{'}(X)$$

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.

The second narrowing operator is derived from the mean value form [16] where the function is approximated using a Taylor expansion of order 1 around the center of the domains of the variables. Given f a real function, F its natural interval extension and J an interval,

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

If this equality is projected on a variable x_i then the interval for the variable can be expressed as

$$X_i = m_i(J) - \frac{1}{\frac{\partial F}{\partial x_i}}[F(m(J)) + \sum_{j=1, j\neq i}^{n} \frac{\partial F}{\partial x_j}(J)(X_j - m_j(J))]$$

This expression gives a method to reduce the interval for x_i using the Taylor narrowing operator, denoted T_i and computing a new interval for x_i with this formula. The new interval for x_i can be computed directly as $X_i \cap T_i(X)$.

To compute the ECS, the main idea is to consider, for each constraint over n variables, $2n$ "copies" of the constraint. To each copy is associated one of the two constraint narrowing operators previously defined, over closed floating point intervals.

5. Experimental results

We have designed in the C language a prototype of constraint solver, called `Cosinus`, implementing exactly the method presented in the section 4. The Gnu-Multi-Precision library [10] implements computations over the rationals.

The table I presents the computational results given by `Cosinus` on various benchmarks: *parabola* (geometric intersection problem), *cubic* (cubic-parabola), *chemistry* (combustion chemistry problem), *Brown* (Brown's almost linear system) from [12], *Eiger* (extended Eiger-Sikorski-Stenger function), *hexane*, $cyclic_3$ (variant of Cyclic n-roots [3]), $cyclic_4$ (variant of Cyclic n-roots [4]) from [6], *geometric* (geometric intersection problem) from [17], *Czapor*, *Winkler* from [21] and *neuro* (neurophysiologic problem) from [20].

Table I. Experimental results.

	S-set			Interval		Total				
	S	d	t	$\div$	t	n	N	t_1	t_2	R
parabola	1	1	0.01	0	0.01	2	2	0.02	0.10	5
cubic	1	1	0.01	2	0.05	2	3	0.06	0.73	12
chemistry	2	2	0.34	0	0.25	4	1	0.59	1.19	2
Brown	4	1	0.04	1	0.19	5	2	0.23	∞	$\nearrow$
Eiger	2	3	0.02	1	0.15	4	2	0.17	0.34	2
geometric	1	3	0.06	1	0.23	2	2	0.29	7.00	24
hexane	3	4	0.55	15	3.75	3	16	4.30	279.45	65
$cyclic_3$	2	3	0.01	1	0.10	3	2	0.11	∞	$\nearrow$
$cyclic_4$	3	5	0.16	7	0.97	4	4	1.13	∞	$\nearrow$
Czapor	2	4	0.13	3	0.40	3	2	0.53	26.60	50
Winkler	2	5	0.09	1	0.44	3	2	0.53	9.17	17
neuro	4	4	0.47	13	1.36	6	8	1.83	∞	$\nearrow$

In their order of appearance in the table, the labels of columns are: "S" is the number of S-polynomials added in the initial system, "d" the depth of the last computed S-set and "t" the computation time of S-sets; "$\div$" is the number of bisections on domains and "t" the computation time of interval methods; "n" is the number of polynomials in the initial system, "N" the number of solutions of the initial system, "t_1" the total computation time in seconds given by `Cosinus`, "t_2" the total computation time of interval methods if S-set computations are disconnected and "R" the ratio between "t_2" and "t_1" to point out the improvement following S-set computations.

The results were computed on a Sun Sparc 4 (110 Mhz) and the precision is 10^{-12} (the width of the resulting intervals of the numerical resolution process). A "∞" indicates that the computation time takes more than an hour. A "$\nearrow$" means that the ratio between t_1 and t_2 is large.

The collection of benchmarks illustrates the advantages of using Gröbner bases to speed-up interval computations. However, in some cases, interval methods have no need of Gröbner bases: a fast convergence of interval methods on the initial system ensures a good computation time which may not be improved adding redundant constraints; the computation time of S-sets may become too large with respect to the computation time of interval methods on the initial system.

6. Conclusion

In this paper, we have proposed a uniform framework for the combination of symbolic, interval-based and local consistency techniques. We

have applied this framework to the computation of box-consistent systems using interval Newton and centered form methods combined with the computation of redundant constraints based on partial Gröbner bases. Further work concerns the design of other techniques for redundancy generation and extensive benchmarking to determine precisely the average depth and the most relevant criterions for partial Gröbner bases computations.

References

1. G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983.
2. F. Benhamou. Heterogeneous Constraint Solving. In *Proceedings of ALP'96*, volume 1139 of *LNCS*, pages 62–76, Aachen, Germany, 1996. Springer-Verlag.
3. F. Benhamou and L. Granvilliers. Combining Local Consistency, Symbolic Rewriting and Interval Methods. In *Proceedings of AISMC-3*, volume 1138 of *LNCS*, pages 144–159, Steyr, Austria, 1996. Springer-Verlag.
4. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS'94*, Ithaca, NY, USA, 1994.
5. F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1997. (Forthcoming).
6. D. Bini and B. Mourrain. Handbook of Polynomial Systems. November 1996.
7. B. Buchberger. Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory. In *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Company, 1985.
8. A. Colmerauer. Specifications of Prolog IV. Draft, 1996.
9. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms*. Springer-Verlag, 1992.
10. T. Granlund. *GNU Multiple Precision Arithmetic Library, edition 2.0*, 1996.
11. E. R. Hansen and S. Sengupta. Bounding Solutions of Systems of Equations using Interval Analysis. *BIT*, 21:203–211, 1981.
12. R. B. Kearfott. Some Tests of Generalized Bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, 1987.
13. R. Krawczyk. A Class of Interval Newton Operators. *Computing*, 37:179–183, 1986.
14. A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.
15. U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95–132, 1974.
16. R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
17. A. Morgan and A. Sommese. Homotopy Methods for Polynomial Systems. *Applied Mathematics and Computation*, 24:115–138, 1987.
18. A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.
19. W. Older and A. Vellino. Constraint Arithmetic on Real Intervals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*. MIT Press, 1993.
20. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 1997. (Forthcoming).
21. F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag, 1996.