AN OVERVIEW OF INTERVAL AND CONSTRAINT PROGRAMMING TECHNIQUES

DAVID DANEY[§][¶], YAHIA LEBBAH[†][¶], CLAUDE MICHEL[‡][¶], AND MICHEL RUEHER[‡][¶]

1. Introduction. Many applications in engineering sciences require finding all isolated solutions to systems of constraints over real numbers. These systems may be non-polynomial and are difficult to solve: the inherent computational complexity is NP-hard and numerical issues are critical in practice (e.g., it is far from being obvious to guarantee correctness and completeness as well as to ensure termination). These systems, called numeric CSP[1], have been approached in the past by different interesting methods: interval methods [23, 15, 25, 11, 26], continuation methods [35] and constraint satisfaction methods [19, 20, 4, 8, 34, 3]. Of particular interest is the mathematical and programming simplicity of the latter approach: the general framework is a branch and prune algorithm that only requires specifying the constraints and the initial range of the variables.

2. Notations and basic definitions. This paper focuses on CSPs where the domains are intervals and the constraints are continuous. A n-ary continuous constraint $C_j(x_1, \ldots, x_n)$ is a relation over the reals. $\mathcal{C}$ stands for the set of constraints.

X or D_x denotes the domain of variable x, that's to say, the set of allowed values for x. $\mathcal{D}$ stands for the set of domains of all the variables of the considered constraint system. $\mathbb{R}$ denotes the set of real numbers whereas $\mathbb{F}$stands for the set of floating point numbers used in the implementation of non linear constraint solvers; if a is a constant in $\mathbb{F}$, a^+ (resp. a^-) corresponds to the smallest (resp. largest) number of $\mathbb{F}$strictly greater (resp. lower) than a.

$X = [\underline{x}, \overline{x}]$ is defined as the set of real numbers x verifying $\underline{x} \leq x \leq \overline{x}$. x, y denote real variables or vectors whereas X, Y denote interval variables or vectors. The *width* $w(X)$ of an interval X is the quantity $\overline{x} - \underline{x}$ while the *mid-point* $M(X)$ of the interval is $(\overline{x} + \underline{x})/2$. A *point interval* X is obtained if $\underline{x} = \overline{x}$. A *box* is a set of intervals and its width is defined as the largest width of its interval members, while its centre is defined as the point whose coordinates is the mid-point of the ranges. $\mathbb{I}(\mathbb{R})^n$ denotes the set of intervals and is ordered by set inclusion.

We also use the "reformulation-linearization technique" notations introduced in [32, 1] with slight modifications.

3. Interval Programming.

3.1. Interval Arithmetics.

Interval arithmetic has been proposed by [23]. It is based on the representation of variables as intervals.

Let f be a real-valued function of n unknowns $\mathbf{X} = \{x_1, \ldots, x_n\}$. An *interval evaluation* F of f for given ranges $\{X_1, \ldots, X_n\}$ for the unknowns is an interval Y such that

$$\forall \mathbf{X} = \{x_1, \ldots, x_n\} \in \mathcal{X} = \{X_1, \ldots, X_n\} \qquad \underline{Y} \leq f(\mathbf{X}) \leq \overline{Y} \tag{3.1}$$

In other words $\underline{Y}, \overline{Y}$ are lower and upper bounds for the values of f when the values of the unknowns are restricted to the box $\mathcal{X}$.

There are numerous ways to calculate an interval evaluation of a function [11, 24]. However, in general, it is not possible to compute the exact enclosure of the range for an arbitrary function over the real numbers [16]. Moore has introduced the concept of *interval extension*: the interval

[†]Université d'Oran, Déartement d'Informatique, 31000 Oran, Algeria

[‡]Université de Nice–Sophia Antipolis

[§]INRIA, 2004 Route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, Franc

[¶]COPRIN Project INRIA–I3S–CNRS

[1]CSP stands for Constraint Satisfaction Problem.

extension of a function is an interval function that computes outer approximations on the range of the function over a domain [11, 24].

The natural interval extension of a real function f is defined by replacing all the mathematical operators in f by their interval equivalents to obtain F. Interval equivalenst exists for all classical mathematical operators and hence interval arithmetics allows to calculate an interval evaluation for allf non-linear expressions, whether algebraic or not. For example if $f(x) = x + \sin(x)$, then the interval evaluation of f for $x \in [1.1, 2]$ can be calculated as follows:

$$F([1.1, 2]) = [1.1, 2] + \sin([1.1, 2]) = [1.1, 2] + [0.8912, 1] = [1.9912, 3]$$

Thus, if $0 \notin F(\mathcal{X})$, then no value exists in the box $\mathcal{X}$ such that $f(\mathbf{X}) = 0$. In other words the equation $f(\mathbf{X})$ has no root in the box $\mathcal{X}$. In general, the bounds of the interval evaluation F overestimate the minimum and maximum of function F over the box $\mathcal{X}$. Interval arithmetics can be implemented taking into account round-off errors. There are numerous interval arithmetics packages implementing this property: one of the most famous library is BIAS/Profil[2] but a promising new package is MPFI [31], based on the multi-precision software MPFR[3].

Let f be a real-valued function of n unknowns $\mathbf{x} = \{x_1, \dots, x_n\}$. An *interval evaluation* F of f for given ranges $\mathbf{X} = \{X_1, \dots, X_n\}$ for the unknowns is an interval Y such that $\forall\ \mathbf{x} \in \mathbf{X},\ \ f(x) \in Y = [\underline{y}, \overline{y}]$. Interval arithmetic preserves *inclusion monotonicity*: if $Y \subseteq X \quad \Rightarrow \quad F(Y) \subseteq F(X)$ but many algebraic proprieties of scalar arithmetic are no longer valid for interval arithmetics. For example, interval arithmetics is sub-distributive: $X(Y + X) \subseteq XY + XZ$ (see [23] for a more detailled description.).

The main limitation of interval arithmetics is *the over-estimation of interval functions.* This is due to two well known problems :

- the so-called *wrapping effect* [14], which overstimates by a unique vector the image of an interval vector (which is in general not a vector).
- the so-called *dependency problem*[11], which is due the independence the different occurences of some variable during the interval evaluation of an expression. In other words, during the interval evaluation process there is no correlation between the different occurrences of a same variable in an equation: these different occurrences are just considered as identical intervals. For instance, consider $X = [0, 10]$. $X - X = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-10, 10]$ instead of $[0, 0]$ as one could expect.

Due to the proprieties of interval arithmetics, the evaluation of a function may yield different results according to the literal form of the equations. Thus, many literal forms may be used as, for example, factorized form (Horner for polynomial system) or distributed form [33].

The Taylor interval extension, noted F_{tay}, of a real function f, over the interval vector X, is defined by the natural extension of a first-order Taylor development of f [29]:

$$F_{\text{tay-1}}(X) = f(x) + J(X).(X - x) \tag{3.2}$$

with $\forall x \in X$, $J()$ being the Jacobian of f. Hansen [11] has proposed a tricky evaluation the equation 3.2.

Second order Taylor development may also be used but their computation is much more expensive.

In general, first order Taylor extension yield a better enclosure than the natural extension on small intervals; moreover they have a quadratic convergence whereas the natural extension has a linear convergence [29].

[2]http://www.ti3.tu-harburg.de/Software/PROFILEnglissch.html
[3]http://www.mpfr.org

Nevertheless, in general, neither F_{nat} nor $F_{\text{tay-1}}$ won't allow to compute the exact range of a function f.

For instance, consider $f(x) = 1 - x + x^2$, and $\mathbf{x} = [0, 2]$, we have:

$$\begin{array}{l} f_{\text{tay}}([0,2]) = f(x) + (2\mathbf{x} - 1)(\mathbf{x} - x) = f(1) + (2[0,2] - 1)([0,2] - 1) = [-2, 4], \\ f([0,2]) = 1 - \mathbf{x} + \mathbf{x}^2 = 1 - [0,2] + [0,2]^2 = [-1, 5], \\ f_{\text{factor}}([0,2]) = 1 + \mathbf{x}(\mathbf{x} - 1) = 1 + [0,2]([0,2] - 1) = [-1, 3] \end{array} \tag{3.3}$$

whereas the range of f over $X = [0, 2]$ is $[3/4, 3]$. In this case, this result could directly be obtained by a second form of factorisation: $f_{\text{factor}_2}([0,2]) = (\mathbf{x} - 1/2)^2 + 3/4 = ([0,2] - 1/2)^2 + 3/4 = [3/4, 3]$.

3.2. Interval Analysis methods. This section provides a short introduction of interval analysis methods. We limit this overview to interval Newton-like methods for solving a multi-variate system of non-linear equations. Their use is complementary to methods provided by the Constraint Programming community.

The aim is to determine the zeros of a system of n equations $f_i(x_1, \ldots, x_n)$ in n unknowns x_i inside the interval vector $X = \{X_1, \ldots, X_n\}$ with $x_i \in X_i$ for $i = 1, \ldots, n$.

First, consider the case of interval linear equations defined by the following system:

$$\mathbf{A}.x = \mathbf{b} \tag{3.4}$$

with $\mathbf{A}$ an interval matrix and $\mathbf{b}$ an interval vector. Solving this linear interval system requires to determine an interval vector X_0 containing all solutions of all scalar linear systems noted $A.x = B$ such that $A \in \mathbf{A}$ and $b \in \mathbf{b}$. Finding X_0 is a difficult problem but two basic interval methods may provided an over estimated interval vector X_1 including X_0.

Note that Gaussian elimination works also for interval linear systems. Indeed, it is possible to get X_1 if the Gauss pivots do not contain any zero in the triangularisation process. However, in general the computed interval are too wide and a preconditionning is required. That's to say a multiplication of both side of equation 3.4 by the inverse of a mid-point of $\mathbf{A}$. The matrix $m(\mathbf{A})^{-1}\mathbf{A}$ is then "closer" to the identity matrix and the width of X_1 is smaller [28].

Another well know method for solving interval system, is the Gauss-Seidel iterative method. For each unknowns X_i, the algorithm [11] is defined by the following iterative process:

$$X_i^{k+1} = (\mathbf{b}_i - \sum_{j=1}^{i-1} \mathbf{A}_{i,j} X_j^{k+1} - \sum_{j=i+1}^{n} \mathbf{A}_{i,j} X_j^k)/\mathbf{A}_{i,i} \cap X_i^k \tag{3.5}$$

Again, a pre-conditioning step may allow to shrink the width of the computed intervals. Details on the implementation of this algorithm can be found in [11].
Note that this method is very close to 2B–consistency filtering techniques (see section 4).

To solve non-linear systems, the Interval Newton algorithm is often used. Here is its general schema:

$$X_{k+1} = N(\tilde{x}_k, X_k) \cap X_k \quad \text{with} \quad N(\tilde{x}_k, X_k) = \tilde{x}_k - A.f(\tilde{x}_k)) \tag{3.6}$$

A is an interval matrix that contains all the inverse of the Jacobian matrix of the system F. The scalar $\tilde{x}_k$ must be chosen inside X_k (for example the mid-point of X_k). Thus, following properties[4] hold:

- If $N(\tilde{x}_k, X_k) \cap X_k = \emptyset$, then the system F has no solution in X_k

[4]For other proprieties and implementation see [11]

- if $N(\tilde{x}_k, X_k)_k \cap X_k \subset X_k$, there is one or more solution in X_{k+1}.

The determination of the matrix A is a critical issue of these type of algorithm. Matrix A is the inverse of the Jacobian matrix $A = [F^{'}(X_k)]^{-1}$ evaluated over the interval vector X_k. However, inverting an interval matrix may be very expensive in time and space. The alternative consists to solve the linear system $F^{'}(X_k)(N(\tilde{x}_k, X_k) - \tilde{x}_k) = -f(\tilde{x}_k)$ to determine $N(\tilde{x}_k, X_k)$. This job can be done by one the algorithms presented previously (see [28, 30]).

The Krawczyk scheme is an interesting adaptation of the scalar secant method. It is defined through the iterative scheme:

$$X_{k+1} = K(\tilde{x}_k, X_k) \cap X_k \text{ with } K(\tilde{x}_k, X_k) = \tilde{x}_k - [f(\tilde{x}_k)]^{-1} f(\tilde{x}_k) + (I - [f(\tilde{x}_k)]^{-1} F^{'}(X_k))(X_k - \tilde{x}_k) \tag{3.7}$$

The properties of this scheme are used by Moore [23] to check the existence and the unicity of a zero and the convergence of the scheme. Note that this scheme uses only the inverse of a scalar matrix compared to the interval matrix inversion of the Newton scheme. This method is fast but efficient only for small width interval variables.

4. Constraint programming .

This section recalls the basics of constraint programming techniques which are required to understand the rest of this paper. Detailed discussion of these concepts and techniques can be found in [4, 17].

4.1. The general framework.

The constraint programming framework is based on a branch & prune schema which was inspired by the traditional branch and bound approach used in optimisation problems. That's to say, it is best viewed as an iteration of two steps[33]:

1. Pruning the search space;
2. Making a choice to generate two (or more) sub-problems.

The pruning step ensures that some local consistency holds. In other words, the pruning step reduces an interval when it can prove that the upper bound or the lower bound does not satisfy some constraint. Informally speaking, a constraint system C satisfies a partial consistency property if a relaxation of C is consistent. For instance consider $X = [\underline{x}, \overline{x}]$ and $C(x, x_1, \ldots, x_n) \in \mathcal{C}$. Whenever $C(x, x_1, \ldots, x_n)$ does not hold for any values $a \in X = [\underline{x}, x']$, then X may be shrinked to $X = [x', \overline{x}]$. Local consistencies are detailed in the next subsection. Roughly speaking, they are relaxations of arc-consistency, a notion well known in artificial intelligence [21, 22].

The branching step usually splits the interval associated to some variable in two intervals with the same width. However, the splitting process may generate more than two sub-problems and one does not need to split an interval in its middle. The choice of the variable to split is a critical issue in difficult problems. Sophisticated splitting strategies have been developed for finite domains but few results are available for continuous domains.

4.2. Local consistencies[8, 17].

Local consistencies are conditions that filtering algorithms must satisfy. A filtering algorithm can be seen as a fixed point algorithm defined by the sequence $\{\mathcal{D}_k\}$ of domains generated by the iterative application of an operator $Op : \mathbb{I}(\mathbb{R})^n \longrightarrow \mathbb{I}(\mathbb{R})^n$ (see Figure 4.1).

$$\boxed{\mathcal{D}_k = \begin{cases} \mathcal{D} & \text{if } k = 0 \\ Op(\mathcal{D}_{k-1}) & \text{if } k > 0 \end{cases}}$$

FIG. 4.1. *Filtering algorithms as fixed point algorithms*

The operator Op of a filtering algorithm generally satisfies the following three properties:

- $Op(\mathcal{D}) \subseteq \mathcal{D}$ (contractance)
- Op is conservative; that is, it cannot remove any solution.
- $\mathcal{D}' \subseteq \mathcal{D} \Rightarrow Op(\mathcal{D}') \subseteq Op(\mathcal{D})$ (monotonicity)

Under those conditions, the limit of the sequence $\{\mathcal{D}_k\}$, which corresponds to the greatest fixed point of the operator Op, exists and is called a *closure*. A fixed point for Op may be characterised by a property lc-consistency, called a local consistency. The algorithm achieving filtering by lc-consistency is denoted lc-filtering. A CSP is said to be lc-satisfiable if lc-filtering of this CSP does not produce an empty domain.

Consistencies used in numeric CSPs solvers can be categorised in two main classes : *arc-consistency*-like consistencies and strong consistencies. Strong consistencies will not be discussed in this paper (see [19, 17] for a detailed description).

Most of the numeric CSPs systems (e.g., BNR-prolog [27], Interlog [10, 6], CLP(BNR) [5], PrologIV [9], UniCalc [2], Ilog Solver [13], Numerica [34] and RealPaver [3][5] compute an approximation of arc-consistency [21] which will be named AC-like-consistency in this paper. AC-like-consistency states a local property on a constraint and on the bounds of the domains of its variables. Roughly speaking, a constraint C_j is AC-like-consistent if for any variable x_i in $var(C_j)$, the bounds $\underline{D_i}$ and $\overline{D_i}$ have a support in the domains of all other variables of C_j.

The most famous AC-like consistencies are 2B–consistency and Box–consistency .

2B–consistency (also known as hull consistency) [7, 5, 18, 19] only requires to check the Arc–Consistency property for each bound of the intervals. The key point is that this relaxation is more easily verifiable than Arc–Consistency itself. Informally speaking, variable x is 2B–consistency for constraint $''f(x, x_1, \ldots, x_n) = 0''$ if the lower (resp. upper) bound of the domain of x is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$. Box–consistency [4, 12] is a coarser relaxation (i.e., it allows more stringent pruning) of Arc–Consistency than 2B–consistency . Variable x is Box–Consistent for constraint $''f(x, x_1, \ldots, x_n) = 0''$ if the bounds of the domain of x correspond to the leftmost and the rightmost zero of the optimal interval extension of $f(x, x_1, \ldots, x_n)$. 2B–consistency algorithms actually achieve a weaker filtering (i.e., a filtering that yields bigger intervals) than Box–consistency , especially when a variable occurs more than once in some constraint (see proposition 6 in [8]). This is due to the fact that 2B–consistency algorithms require a decomposition of the constraints with multiple occurrences of the same variable.

2B–consistency [19] states a local property on the bounds of the domains of a variable at a single constraint level. A constraint c is 2B–Consistent if, for any variable x, there exist values in the domains of all other variables which satisfy c when x is fixed to $\underline{x}$ and $\overline{x}$. More formally, we have :

Definition 4.1 (2B–consistency).
Let (X, C) be a CSP and $c \in C$ a k-ary constraint over $(x_1, \ldots, x_k)$. c is 2B–consistency iff :
$\forall i, \mathbf{x_i} = \Box\{\tilde{x}_i \mid \exists \tilde{x}_1 \in \mathbf{x_1}, \ldots, \exists \tilde{x}_{i-1} \in \mathbf{x_{i-1}}, \exists \tilde{x}_{i+1} \in \mathbf{x_{i+1}}, \ldots, \exists \tilde{x}_k \in \mathbf{x_k}$ *such that* $c(\tilde{x}_1, \ldots, \tilde{x}_{i-1}, \tilde{x}_i, \tilde{x}_{i+1} \ldots, \tilde{x}_k)$ *holds*$\}$.
A CSP is 2B–consistency iff all its constraints are 2B–consistency .

Definition 4.2 (Closure by 2B–consistency). *[19]*
The filtering by 2B–consistency of $P = (X, C)$ is the CSP $P' = (X', C)$ such that :

- *P and P' have the same solutions;*
- *P' is 2B–Consistent;*
- *$X' \subseteq X$ and the domains in X' are the largest ones for which P' is 2B–Consistent.*

We note $\Phi_{2B}(P)$ the filtering by 2B–consistency of P. In the following we will use the term *closure* by 2B–consistency to emphasise the fact that this filtering always exists and is unique [19].

Box–consistency [4, 12] is a coarser relaxation of Arc–Consistency than 2B–consistency . It mainly consists of replacing every existentially quantified variable but one with its interval in

[5]See also http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver/main.html

the definition of 2B–consistency . Thus,Box–consistency generates a system of univariate interval functions which can be tackled by numerical methods such as Newton. Contrary to 2B–consistency , Box–consistency does not require any constraint decomposition and thus does not amplify the locality problem. Moreover, Box–consistency can tackle some dependency problems when each constraint of a CSP contains only one variable which has multiple occurrences. More formally :

Definition 4.3 (Box–consistency).
Let (X, C) *be a CSP and* $c \in C$ *a* k*-ary constraint over the variables* $(x_1, \ldots, x_k)$. c *is Box–Consistent if, for all* x_i *the following relations hold :*
1. $\mathbf{c}(\mathbf{x_1}, \ldots, \mathbf{x_{i-1}}, [\underline{\mathbf{x_i}}, \underline{\mathbf{x_i}}^{+}), \mathbf{x_{i+1}}, \ldots, \mathbf{x_k})$,
2. $\mathbf{c}(\mathbf{x_1}, \ldots, \mathbf{x_{i-1}}, (\overline{\mathbf{x_i}}^{-}, \overline{\mathbf{x_i}}], \mathbf{x_{i+1}}, \ldots, \mathbf{x_k})$.

Closure by Box–consistency of P is defined similarly to closure by 2B–consistency of P, and is denoted by $\Phi_{Box}(P)$.

Benhamou et all have introduced HC4 [33] an AC-like-consistency that merges 2B–consistency and Box–consistency and which optimises the computation process.

4.3. Local consistency filtering[17]. The algorithms that achieve a local consistency filtering are based upon projection functions.

To compute the projection $\pi_{j,i}(\mathcal{D})$ of the constraint C_j on the variable x_i, we need to introduce the concept of *solution function* that expresses the variable x_i in terms of the other variables of the constraint. For example, for the constraint $x + y = z$, the solution functions are: $f_x = z - y, f_y = z - x, f_z = x + y$.

Assume a solution function is known that expresses the variable x_i in terms of the other variables of the constraint. Thus an approximation of the projection of the constraint over x_i given a domain $\mathcal{D}$ can be computed thanks to any interval extension of this solution function. Thus we have a way to compute $\pi_{j,i}(\mathcal{D})$.

Nevertheless, for complex constraints, there may not exist such an analytic solution function; for example, consider $x + log(x) = 0$. The interest of numeric methods as presented in this paper is precisely for those constraints that cannot be solved algebraically. Three main approaches have been proposed:

- The first one exploits the fact that analytic functions always exist when the variable to express in terms of the others appears only one time in the constraint. This approach simply considers that each occurrence of a variable is a different new variable. In the previous example this would give: $x_{(1)} + \log(x_{(2)}) = 0$. That way, it is trivial to compute a solution function: it suffices to know the inverse of basic operators. In our example, we obtain $f_{x_{(1)}} = -\log(x_{(2)})$ and $f_{x_{(2)}} = \exp^{-x_{(1)}}$.
  An approximation of the projection of the constraint over x_i can be computed by intersecting the natural interval extensions of the solution functions for all occurrences of x_i in C_j. For the last example, we could take $\pi_{x+log(x)=0,x}(X) = -\log(X) \cap \exp^{-X}$.
  This approach is used to achieve 2B–consistency filtering. In general, the initial constraints are decomposed into primitive constraints –for which the approximation of the projections functions are easy to compute– by introducing new variables. Decomposition does not change the semantics of the initial constraints system : the initial system and the decomposed one do have the same solutions. However, a local consistency like 2B–consistency is not preserved by such a rewriting. Indeed, the decomposition amplifies the dependency problem, and thus, the yields a weaker filtering (see [8] for a detailed discussion of this point).
- The second idea uses the Taylor extension to transform the constraint into an interval

linear constraint. The non-linear equation $f(X) = 0$ becomes

$$f(c) + \sum_{i=1}^{n} nat(\frac{\partial f}{\partial x_i})(X) * (X_i - c_i) = 0$$

where $c = m(X)$. Now consider that the derivatives are evaluated over a box $\mathcal{D}$ that contains X. $\mathcal{D}$ is considered as constant, and let $c = m(\mathcal{D})$. The equation becomes:

$$f(c) + \sum_{i=1}^{n} nat(\frac{\partial f}{\partial x_i})(\mathcal{D}) * (X_i - c_i) = 0$$

This is an interval linear equation in X, which does not contain multiple occurrences. The solution functions could be extracted easily. But, instead of computing the solution functions of the constraint without taking into account the other constraints, we may prefer to group together several linear equations in a squared system. Solving the squared interval linear system allows much more precise approximations of projections to be computed. (See the following Section.)

- A third approach [4] does not use any analytical solution function. Instead, it transforms the constraint $C_j(x_{j_1}, ...x_{j_k})$ into k mono-variable constraints $C_{j,l}, l = 1 \ldots k$. The mono-variable constraint $C_{j,l}$ on variable x_{j_l} is obtained by substituting their intervals for the other variables. The projection π_{j,j_l} is computed thanks to $C_{j,l}$. The smallest zero of $C_{j,l}$ in the interval under consideration is a lower bound for the projection of C_j over x_{j_l}. And the greatest zero of $C_{j,l}$ is an upper bound for that projection. Hence, an interval with those two zeros as bounds gives an approximation of the projection. Projection functions computed in that way are called π^{box}.
  This approach is well adapted for Box–consistency filtering : in [4], the two extremal zeros of $C_{j,l}$ are found by a dichotomy algorithm combined with a mono-variable version of the interval Newton method[6].

REFERENCES

[1] C. Audet, P. Hansen, B. Jaumard, and G. Savard. Branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, pages 87(1), 131–152, 2000.

[2] A.B. Babichev, O.P. Kadyrova, T.P. Kashevarova, A.S. Leshchenko, and Semenov A.L. Unicalc, a novel approach to solving systems of algebraic equations. *Interval Computations 1993 (2)*, pages 29–47, 1993.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising hull and box consistency. In *Proceedings of ICLP'99, The MIT Press*, pages 230–244, 1999.

[4] F. Benhamou, D. McAllester, and P. Van-Hentenryck. Clp(intervals) revisited. In *Proceedings of the International Symposium on Logic Programming*, pages 124–138, 1994.

[5] F. Benhamou and W. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, pages 32(1):1–24, 1997.

[6] B. Botella and P. Taillibert. Interlog: constraint logic programming on numeric intervals. In *3rd International Workshop on Software Engeneering, Artificial Intelligence and Expert Systems, Oberammergau, October 4-8*, 1993.

[7] J. C. Cleary. Logical arithmetic. *Future Computing Systems*, pages 2(2) :125–149, 1987.

[8] H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, pages Vol.5(3),213–228, 1999.

[9] A. Colmerauer. Spécifications de prolog iv. Technical report, GIA, Faculté des Sciences de Luminy,163, Avenue de Luminy 13288 Marseille cedex 9 (France), 1994.

[10] Electronique. Dassault. Interlog 1.0: Guide d'utilisation. Technical report, Dassault Electronique, 55 Quai M. Dassault, 92214 Saint Cloud, France, 1991.

[11] Eldon R. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, New York, 1992.

[6]The general (multi-variable) interval Newton method is briefly presented in Section 2.3.

[12] H. Hong and V. Stahl. Starting regions by fixed points and tightening. *Computing*, pages 53 :323–335, 1994.

[13] Ilog, editor. *ILOG Solver 4.0, Reference Manual.* Ilog, 1997.

[14] R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers Group, 1996.

[15] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[16] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl. *Computational complexity and feasibility of data processing and interval computations.* Kluwer, 1998.

[17] Y. Lebbah and O. Lhomme. Accelerating filtering techniques for numeric CSPs. *Artificial Intelligence*, 139(1):109–132, 2002.

[18] J. H. M. Lee and M. H. van Emden. Interval computation as deduction in CHIP. *Journal of Logic Programming*, pages 16 :3–4,255–276, 1993.

[19] O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 232–238, Chambéry(France), 1993.

[20] O. Lhomme, A. Gotlieb, M. Rueher, and P. Taillibert. Boosting the interval narrowing algorithm. In *Proceedings of Joint International Conference and Symposium on Logic Programming*, pages 378–392. MIT Press, 1996.

[21] A. Mackworth. Consistency in networks of relations. *Journal of Artificial Intelligence*, pages 8(1):99–118, 1977.

[22] U. Montanari. Networks of constraints : Fundamental properties and applications to image processing. *Information science*, 7:95–132, 1974.

[23] R. Moore. *Interval Analysis.* Prentice Hall, 1966.

[24] R. E. Moore. *Methods and Applications of Interval Analysis.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[25] A. Neumaier. *Interval Methods for Systems of Equations*, volume 37 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, UK, 1990.

[26] A. Neumaier. *Introduction to Numerical Analysis.* Cambridge Univ. Press, Cambridge, 2001.

[27] W.J. Older and A. Velino. Extending prolog with constraint arithmetic on real intervals. In *Proc. of IEEE Canadian conference on Electrical and Computer Engineering*, pages 14.1.1–14.1.4. IEEE Computer Society Press, 1990.

[28] M. Novoa R. B. Kearfott and C. Hu. Interval Gauss-Seidel method. *Interval Computations*, (1) 1:59–85, 1991.

[29] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Ellis Horwood Ser.: Math. Appl. Ellis Horwood, 1984.

[30] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Ltd., Chichester, 1988.

[31] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the mpfi library. In *Validated Computing conference*, May 2002.

[32] H.D. Sherali and C.H. Tuncbilek. A global optimization algorithm for polynomial using a reformulation-linearization technique. *Journal of Global Optimization*, pages 7, 1–31, 1992.

[33] P. Van-Hentenryck, D. Mc Allester, and D. Kapur. Solving polynomial systems using branch and prune approach. *SIAM Journal on Numerical Analysis*, pages 34(2):797–827, 1997.

[34] P. Van-Hentenryck, L. Michel, and Y. Deville. *Numerica: A Modeling Language for Global Optimization.* The MIT Press, Cambridge, MA, USA, 1997.

[35] J. Verschelde. The database of polynomial systems. Technical report, http://www.math.uic.edu/ jan/Demo/, 2003.