Solving Continuous Constraint Systems*

Michel Rueher
rueher@essi.fr
Université de Nice Sophia-Antipolis
COPRIN project I3S-CNRS/INRIA/CERTIS
930,route des colles
BP 145, 06902 Sophia-Antipolis

Abstract

This paper provides an overview of the constraint techniques for solving non-linear systems of equations over the real numbers. It introduces the main concepts behind the different pruning techniques and searching strategies implemented in the most famous constraint solvers over continuous domains.

Local consistencies are a key issue in finite domains where arc-consistency is very popular. However, in general it is not possible to achieve arc-consistency on constraint systems over the reals. That's why different relaxations of arc consistency have been proposed. Two of most popular local consistency techniques in continuous domains are analysed. The very general scheme of the filtering algorithm is also described.

On difficult problems, local consistencies are often unable to achieve any significant pruning. This is due to the fact that the information provided by a single constraint is too poor to reduce the domains of the variables. The characteristics and capabilities of strong consistency filtering algorithms – which have been developed to achieve better pruning of the domains– are explained.

Global constraints did really boost constraint techniques over finite domains. On continuous domains only a few global constraints are available. This paper describes two global constraints based more or less on dual approaches.

Finally, searching strategies are also briefly discussed and some pointers to successful applications are provided.

*Published in Proc. of "3IA'2005 (8^{th} International Conference On Computer Graphics And Artifi cial Intelligence), Limoges (France), 11 - 12 of May, 2005, pp. 35–55, ISBN number 2-914256-07-8

1 Introduction

Many applications in engineering sciences require finding all isolated solutions to systems of constraints over real numbers. Sometimes the best solution is also expected, that is to say the user wants to minimize an objective function under nonlinear equalities and inequalities.

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 (it is far from being obvious to guarantee correctness and completeness as well as to ensure termination). Recently, these systems have been approached by constraint satisfaction methods (e.g., [27, 5, 10, 48, 7]). Of particular interest is the mathematical and programming simplicity of this approach: the general framework is a branch and prune algorithm that only requires specifying the constraints and the initial range of the variables.

This paper provides an overview of the different pruning techniques and searching strategies implemented in the most famous solvers (e.g., `Numerica` [48]), `RealPaver` [1], `ILOG Solver` [16], `QuadSolver` [23, 22].

Since all these solvers are based on interval techniques, we first recall the basics on interval arithmetic and interval analysis methods. Then, we introduce the overall constraint programming framework. Different local consistency filtering techniques are introduced. Stronger consistencies and global constraints are also investigated. Finally, searching strategies are also briefly addressed.

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

2 Basics

2.1 Notations and definitions

This paper focuses on CSP[2] where the domains are intervals and the constraints are n-ary relations over the reals. $c_j(x_1, \ldots, x_n)$ denotes a constraint over the real number whereas $\mathcal{C}$ stands for the set of constraints.

$\mathbf{x}$ denotes the domain of variable x, that is 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. $\mathbb{IR}$ denotes the set of real numbers whereas $\mathbb{IF}$ stands for the set of floating point numbers used in the implementation of nonlinear constraint solvers; if a is a constant in $\mathbb{IF}$, a^+ (resp. a^-) corresponds to the smallest (resp. largest) number of $\mathbb{IF}$ strictly greater (resp. lower) than a.

$\mathbf{x} = [\underline{x}, \overline{x}]$ is defined as the set of real numbers x verifying $\underline{x} \leq x \leq \overline{x}$. x, y denote real variables, X, Y denote vectors whereas $\mathbf{X}, \mathbf{Y}$ denote interval vectors. The *width* $w(\mathbf{x})$ of an interval $\mathbf{x}$ is the quantity $\overline{x} - \underline{x}$ while the *midpoint* $m(\mathbf{x})$ of the interval $\mathbf{x}$ is $(\overline{x} + \underline{x})/2$. A *point interval* $\mathbf{x}$ is obtained if $\underline{x} = \overline{x}$. A *box* is a set of intervals: its width is defined as the largest width of its interval members, while its center is defined as the point whose coordinates is the midpoint of the ranges. $\mathbb{IIR}^n$ denotes the set of boxes and is ordered by set inclusion. $\Box S$ denotes the smallest interval containing all values of set S.

2.2 Interval arithmetic

The interval arithmetic framework has been introduced by Moore [31]. It is based on the representation of the domains of the variables as intervals. Moore introduced also the con-

[2] CSP stands for Constraint Satisfaction Problem

cept of *natural interval extension*: the natural interval extension of an expression e is defi ned by replacing all the mathematical operators in e by their interval equivalents to obtain E. The interval equivalents of the basic mathematical operators are :

- $[a,b] \ominus [c,d] = [a-d, b-c]$
- $[a,b] \oplus [c,d] = [a+c, b+d]$
- $[a,b] \otimes [c,d] =$
  $[\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]$
- $[a,b] \oslash [c,d] =$
  $[\min(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d}), \max(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d})]$
  if $0 \notin [c,d]$

Interval equivalents exist for all classical mathematical operators. Hence interval arithmetic allows to calculate an interval evaluation for all nonlinear expressions, whether algebraic or not.

Let f be a real-valued function of n unknowns $X = (x_1, \ldots, x_n)$. An *interval evaluation* of f for given ranges $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ for the unknowns is an interval $\mathbf{y}$ such that $\underline{y} \leq f(\mathbf{X}) \leq \overline{y}$, for all $X = (x_1, \ldots, x_n) \in \mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$. 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 $\mathbf{X}$.

There are numerous ways to calculate an interval evaluation of a function [13, 40]. The simplest is the *natural interval evaluation* in which all the mathematical operators in f are substituted by their interval equivalents. For example if $f(x) = x + \sin(x)$, then the natural interval evaluation of f for $x \in [1.1, 2]$ can be calculated as follows:

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

However, in general it is not possible to compute the exact enclosure of the range for an arbitrary function over the real numbers [20]. Thus, the actual interval extension of a function is an interval function that computes outer approximations on the range of the function over a domain [13, 32].

2.3 Properties and limitations of interval arithmetic

Interval arithmetic has some nice properties:

- If $0 \notin F(\mathbf{x})$, then no value exists in the box $\mathbf{X}$ such that $f(X) = 0$, that's to say the equation $f(\mathbf{X})$ has no root in the box $\mathbf{X}$;
- Interval arithmetic can be implemented taking into account round-off errors;
- Interval arithmetic preserves *inclusion monotonicity*:
  $\mathbf{Y} \subseteq \mathbf{X} \Rightarrow \mathbf{F}(\mathbf{Y}) \subseteq \mathbf{F}(\mathbf{X})$ but interval arithmetic is *sub-distributive*
  $\mathbf{X}(\mathbf{Y} + \mathbf{X}) \subseteq \mathbf{XY} + \mathbf{XZ}$.

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

- the so-called *wrapping effect*;
- the so-called *dependency problem*.

The wrapping effect [31, 34] overestimates by a unique vector the image of an interval vector (which is in general not a vector). That is to say, $\{f(x) | x \in \mathbf{x}\}$ is contained in $f(\mathbf{x})$ but is usually not equal to $f(\mathbf{x})$.

The dependency problem [13] is due to the independence of the different occurrences of some variables 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. For instance, consider $\mathbf{x} = [0, 10]$:
$\mathbf{x} - \mathbf{x} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-10, 10]$ instead of $[0, 0]$ as one could expect. Likewise, consider $\mathbf{x} = [\mathbf{0}, \mathbf{5}]$, then we have:
$\mathbf{x}^2 - \mathbf{x} = [0, 25] - [0, 5] = [-5, 25]$ whereas
$\mathbf{x}(\mathbf{x} - 1) = [0, 5]([0, 5] - [1, 1])$
$= [0, 5][-1, 4] = [-5, 20]$.

3 Constraint Programming

This section recalls the basics of constraint programming techniques (see [5, 21] and Alain Colmerauer's tutorial[3] for more details).

A numerical CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables
- $\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ is a set of domains ($\mathbf{x_i}$ contains all acceptable values for variable x_i)
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of constraints

The constraint programming framework is based on a *branch and prune* scheme which was inspired by the traditional branch and bound approach used in optimisation problems. It is best viewed as an iteration of two steps [47]:

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. Roughly speaking, local consistencies are relaxations of arc-consistency[28, 30], a notion well known in artificial intelligence.

[3] http://www.lif-sud.univ-mrs.fr/ colmer/

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 subproblems and one may split an interval at a point different from its midpoint. The choice of the variable to split is a critical issue in difficult problems. Search techniques are addressed in section 6.

4 Consistency techniques

Local consistencies are a key issue in finite domains where arc-consistency [28, 30] is very popular. A constraint c_j is arc-consistent if for any variable x_i in $\mathcal{X}_j$, each value in $\mathbf{x}$ has a support in the domains of all other variables of $\mathcal{X}_j$. In other words, a constraint $\mathcal{X}_j$ is arc-consistent for variable x, if values exist in the domains of all other variables such that constraint $\mathcal{X}_j$ holds when x is assigned to any value of $\mathbf{x}$.

The essential observation is that local consistency filtering algorithms try to reduce the size of the domain of some variable by considering only one constraint. However, in general it is not possible to achieve arc-consistency on constraint systems over the reals. That's why different relaxations of arc consistency have been proposed. This section introduces the main concepts behind two of most popular local consistency techniques in continuous domains.

The very general scheme of the filtering algorithms is also described.

4.1 Local consistencies [10, 21]

Informally speaking, a constraint system c satisfies a local consistency property if *a relaxation* of c is consistent.

Consider $\mathbf{x} = [\underline{x}, \overline{x}]$ and some constraint $c(x, x_1, \ldots, x_n) \in \mathcal{C}$, whenever $c(x, x_1, \ldots, x_n)$ does not hold for any values $a \in [\underline{x}, x']$, then $\mathbf{x}$ may be shrinked to $\mathbf{x} = [x', \overline{x}]$

Consistencies used in numerical CSP can be categorised in two main classes:

- *arc-consistency*-like consistencies;
- *strong* consistencies.

Most of the numerical CSP systems (e.g., BNR-prolog [37], Interlog [8], CLP(BNR) [6], PrologIV [11], UniCalc [4], Ilog Solver [16], Numerica [48] and RealPaver [7]) compute an approximation of arc-consistency [28] which will be named ac-like-consistency in this paper. An ac-like-consistency states a local property on a constraint and on the bounds of the domains of its variables.

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

2B–consistency [27] states a local property on the bounds of the domains of a variable at a single constraint level. 2B–consistency [4] [9, 6, 26, 27] 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.
Intuitively, 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 $\underline{x}$ or $\overline{x}$. In other words, variable x is 2B–consistent for constraint $'f(x, x_1, \ldots, x_n) = 0$ if $\underline{x}$ (resp. $\overline{x}$) is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$.

More formally, let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and $C \in \mathcal{C}$ a k-ary constraint over $(x_1, \ldots, x_k)$
C is 2B–consistency iff :
$\forall i, \mathbf{x}_i = \square\{v_i \mid \exists v_1 \in \mathbf{x}_1, \ldots, \exists v_{i-1} \in \mathbf{x}_{i-1}, \exists v_{i+1} \in \mathbf{x}_{i+1}, \ldots, \exists v_k \in \mathbf{x}_k$ such that $c(v_1, \ldots, v_{i-1}, v_i, v_{i+1} \ldots, v_k)$ holds$\}$

[4] Also known as hull consistency.

A CSP is 2B–consistency iff all its constraints are 2B–consistent.

The filtering by 2B–consistency of $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the CSP $P' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ such that :

- P and P' have the same solutions;
- P' is 2B–consistent;
- $\mathcal{D}' \subseteq \mathcal{D}$ and the domains in $\mathcal{D}'$ are the largest ones for which P' is 2B–consistent.

Filtering by 2B–consistency of P always exists and is unique [27], that is to say it is a closure.

Box–consistency [5, 15] is a coarser relaxation (i.e., it allows less stringent pruning) of arc–consistency than 2B–consistency . Informally speaking, 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 $f(x, x_1, \ldots, x_n)$.
However, 2B–consistency algorithms actually achieve a weaker filtering (i.e., a filtering that yields bigger intervals) than Box–consistency , more precisely when a variable occurs more than once in some constraint (see proposition 6 in [10]). This is due to the fact that 2B–consistency algorithms require a decomposition of the constraints with multiple occurrences of the same variable.

Box–consistency may be defined by replacing every existentially quantified variable but one with its interval in the definition of 2B–consistency . Thus, Box–consistency generates a system of univariate interval functions which can be tackled by numerical methods such as interval Newton.

In contrast 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, let $(\mathcal{D}, \mathcal{C})$ be a CSP and $c \in \mathcal{C}$ 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. $c(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, [\underline{x}_i, \underline{x}_i^+), \mathbf{x}_{i+1}, \ldots, \mathbf{x}_k)$,
2. $c(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, (\overline{x}_i^-, \overline{x}_i], \mathbf{x}_{i+1}, \ldots, \mathbf{x}_k)$.

Closure by Box–consistency of P is defined similarly to closure by 2B–consistency of P.

Benhamou et al. have introduced BC4 [7], an ac-like-consistency that merges 2B–consistency and Box–consistency and which optimises the computation process.

4.2 Filtering algorithms

Local consistencies are conditions that filtering algorithms must enforce. 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 $Op : I\!I\!R^n \longrightarrow I\!I\!R^n$ (see Figure 1).

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

Figure 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 lc-consistency property, 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.

Algorithms that achieve a local consistency filtering are based upon projection functions. To compute the projection $\pi_{j,i}$ of the constraint c_j on the variable x_i, we need to introduce the concept of *solution function*. Solution functions express the variable x_i in terms of the other variables of the constraint. The solution functions of $x + y = z$ 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 $\mathbf{x}_i$ can be computed with any interval extension of this solution function. So, we have a way to compute $\pi_{j,i}$.

For complex constraints, no analytic solution function may exist. For instance, consider $x + log(x) = 0$.

The interest of numerical methods 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:
$\pi_{x+log(x)=0,x} = -\log(\mathbf{x}) \cap \exp^{-\mathbf{x}}$.
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 projection 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, yields a weaker filtering (see [10] for a detailed discussion of this point).

The second approach 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. However, 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.

A third approach [5] 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 [5], 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.

4.3 Strong consistencies

On difficult problems local consistencies are often unable to achieve any significant pruning. This is due to the fact that the information provided by a single constraint is too poor to reduce the domains of the variables.

The scope of strong consistency filtering algorithms is not restricted to a single constraint, and thus, they are no more strictly local consistencies. Strong consistency filtering algorithms can achieve very impressive filtering on some variables of difficult problems, but

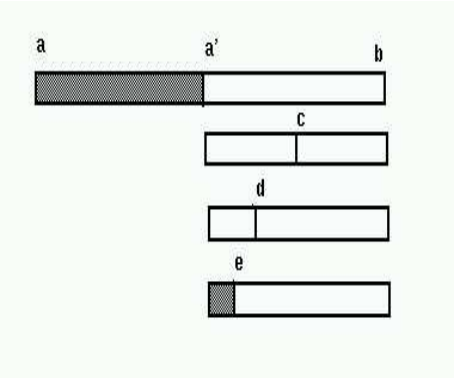

Figure 2: 3B–consistency filtering process

they may also be very time consuming. So, the challenge is to find a good trade-off between the efficiency of the filtering process and its cost.

3B–consistency [27] checks whether 2B–consistency can be enforced when the domain of a variable is reduced to the value of one of its bounds in the whole system. Bound-consistency[39] is similar to 3B–consistency but it is based on Box–consistency.

More formally, let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and x a variable of $\mathcal{X}$ with $\mathbf{x} = [a, b]$. Let also:

- $P_{\mathbf{x}^1 \leftarrow [a, a^+)}$ be the CSP derived from P by substituting $\mathbf{x}$ in $\mathcal{D}$ with $\mathbf{x}^1 = [a, a^+)$
- $P_{\mathbf{x}^2 \leftarrow (b^-, b]}$ be the CSP derived from P by substituting $\mathbf{x}$ in $\mathcal{D}$ with $\mathbf{x}^2 = (b^-, b]$

X is 3B–consistent iff $\Phi_{2B}(P_{x \leftarrow [\underline{x}, \underline{x}^+)}) \neq P_\emptyset$ and $\Phi_{2B}(P_{x \leftarrow (\overline{x}^-, \overline{x}]}) \neq P_\emptyset$.

The filtering process is based on a kind of refutation algorithm. The point is that this algorithm tries to prove that no solution can exist on some subpart of the domain. However, the only subparts "on the bounds" of the domains are investigated so that the resulting domains remain single intervals.

For instance, let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and $\mathbf{x} = [a, b]$, if $\Phi_{2B}(P_{\mathbf{x} \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$

- then the part $[a, \frac{a+b}{2})$ of $\mathbf{x}$ will be removed and the filtering process continues on the interval $[\frac{a+b}{2}, b]$
- otherwise, the filtering process continues on the interval $[a, \frac{3a+b}{4}]$.

This process is depicted in fig 2.

5 Global constraints

The drawback of local consistencies like 2B–consistency and Box–consistency comes from the fact that the constraints are handled independently and in a blind way. 3B–consistency and Bound-consistency are partial consistencies that can achieve a better pruning since they are "less local" [10]. However, they may require numerous splitting steps to find the solutions of a system of nonlinear constraints; so, they may become rather slow.

Global constraints did really boost constraint techniques over finite domains. On continuous domains only few global constraints are available. In this section we briefly describe two global constraints based more or less on dual approaches.

5.1 A syntactical approach based on linear approximations

Classical local consistencies do not exploit the semantic of quadratic terms; that's to say, these approaches do not take advantage of the very specific properties of quadratic constraints to reduce the domains of the variables. Linear programming techniques [18, 45, 1] do capture most of the semantic of quadratic terms (e.g., convex and concave envelopes of these particular terms). In [23, 24], the authors did introduce a global filtering algorithm (named `Quad`) for handling systems of quadratic equations and inequalities over the real numbers.

The Quad-algorithm computes convex and concave envelopes of bilinear terms xy as well as concave envelopes and convex underestimations for square terms x^2.

More precisely, since every nonlinear term can be rewritten as sums of products of univariate terms, Quad introduces relaxations for handling the following terms:

- power term x^n
- product of variables $x_1x_2...x_n$
- univariate term $f(x)$

The Quad-algorithm is used as a global filtering algorithm in a branch and prune approach [47].

So, Quad is a global constraint that works on a tight and safe linear relaxation of nonlinear terms.
This relaxation is adapted from a classical linearization method, the "Reformulation-Linearization Technique (RLT)" [45, 44]. The global filtering algorithm is based on an iterative process.
First, the Simplex algorithm is used to narrow the domain of each variable with respect to the subset of the linear set of constraints generated by the relaxation process.
Then, the coefficients of these linear constraints are updated with the new values of the bounds of the domains and the process is restarted until no more significant reduction can be done.

The use of the Simplex in the filtering process is a critical issue: (1) The coefficients of the generated linear constraints are real numbers. However, when they are computed on floating point numbers the whole linearization may be incorrect due to rounding errors; (2) Most linear programming solvers are implemented with floating point numbers that require rounding operations, and thus, are unsafe.

Quad provides a safe procedure to handle these two problems, and thus to prevent the linearization process and the Simplex algorithm from removing any solution.
The linearization process first decomposes each nonlinear term in sums and products of univariate terms, then, it replaces nonlinear terms with their associated new variables.
For example, consider constraint c : $x_2x_3x_4^2(x_6 + x_7) + \sin(x_1)(x_2x_6 - x_3) = 0$, a simple linearization transformation may yield the following sets:

- $[c]_L = \{y_1 + y_3 = 0,\ \ y_2 = x_6 + x_7,\ \ y_4 = y_5 - x_3\}$
- $[c]_{LI} = \{y_1 = x_2x_3x_4^2y_2,$ $y_3 = \sin(x_1)y_4,\ \ y_5 = x_2x_6\}$.

$[c]_L$ is the set of linear constraints generated by replacing the nonlinear terms by new variables and $[c]_{LI}$ denotes the set of equations that keep the link between the new variables and the nonlinear terms. Note that the nonlinear terms which are not directly handled by Quad may be taken into account by a local–filtering process.

A key issue is the introduction of a new linear relaxation to capture the semantic of these quadratic terms. A tight linear (convex) relaxation, or outer-approximation to the convex and concave envelope of the quadratic terms over the constrained region, is built by generating new linear inequalities.

We just detail here the linearization process for x^2.

The term x^2 with $\underline{x} \leq x \leq \overline{x}$ is approximated by the following relations:

$$\begin{array}{rcl} L1(\alpha) & \equiv & [(x-\alpha)^2 \geq 0]_L \text{ where } \alpha \in [\underline{x}, \overline{x}] \\ L2 & \equiv & [(\underline{x}+\overline{x})x - y - \underline{x}\overline{x} \geq 0]_L \end{array} \quad (1)$$

where $[S]_L$ denotes the set of linear constraints generated by replacing the nonlinear terms of S by new variables.

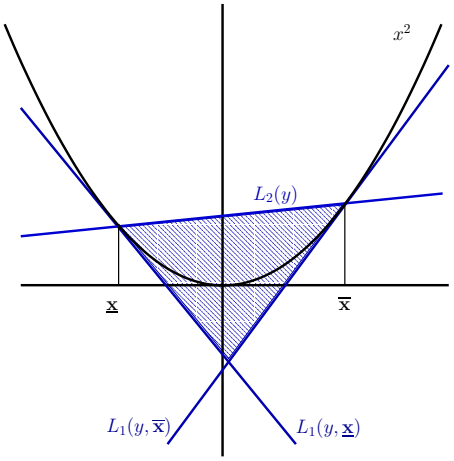

Figure 3: Illustration of x^2 relaxations

Inequality $L1(\alpha)$ trivially holds whereas inequality $L2$ comes from the following valid inequality:

$$0 \leq [(x-\underline{\mathbf{x}})(\overline{\mathbf{x}}-x)]_L = -y+(\underline{\mathbf{x}}+\overline{\mathbf{x}})x-\underline{\mathbf{x}}\overline{\mathbf{x}}$$

Note that $[(x-\alpha)^2 = 0]_L$ generates the tangent line to the curve $y = x^2$ at the point $x = \alpha$. Actually, `Quad` computes only $L1(\overline{x})$ and $L1(\underline{x})$. Consider for instance the quadratic term x^2 with $x \in [-4, 5]$. Figure 3 displays the initial curve (i.e., x^2), and the lines corresponding to the equations generated by the relaxations: $L1(-4) \equiv y + 8x + 16 \geq 0$, $L1(5) \equiv y - 10x + 25 \geq 0$, and $L2 \equiv -y + x + 20 \geq 0$.

We may note that $L1(\overline{x})$ and $L1(\underline{x})$ are underestimations of x^2 whereas $L2$ is an overestimation. $L2$ is also the concave envelope, which means that it is the optimal concave overestimation.

A combination of this filtering technique with Box–consistency filtering algorithm has been investigated. Experimental results on difficult problems show that a solver based on this combination outperforms classical solvers.

5.2 An approach based on semantic properties

Michael Heusch [14] did investigate a specific class of quadratic constraints : n-ary euclidean distance relations. `distn`, the global constraint he did introduce works on both minimal and maximal distances.

The syntax of the global constraint `distn` is given by :

$$distn([P_1, \ldots, P_n], d, D) \qquad (2)$$

where $P_i = (x_i, y_i)$ is the Cartesian product of the domains of the coordinates of some point, d and D are symmetric non-negative matrices of size $n \times n$ defining the minimal and maximal distance factors.
Constraint 2 holds if :

$$\forall p_i = (x_i, y_i) \in P_i, \forall j, \exists p_j \in P_j : dist(p_i, p_j) \geq d_{ij} \text{ and}$$
$$\forall p_i = (x_i, y_i) \in P_i, \forall j, \exists p_j \in P_j : dist(p_i, p_j) \leq D_{ij}.$$

In other words, constraint 2 holds if P_i is the box containing the set of points p_i such that in all other boxes P_j there exists at least one point p_j at a distance greater than d_{ij} whilst not farther than D_{ij}.

Heusch's approach takes advantage of the geometrical properties of euclidean distance constraints. That's to say, the filtering process is based on a set of geometrical rules. The resulting filtering is stronger than classical local filtering.
For instance in the example on figure 5.2 neither 2B–consistency nor Box–consistency can achieve any filtering. The geometricalrules used in `distn` will remove part of the domain of box **A** above the doted line.

The filtering agorithm he did propose actually generalise a circle packing algorithm proposed in [29].

In [2] the authors did also introduce a new pruning technique based on a decomposition

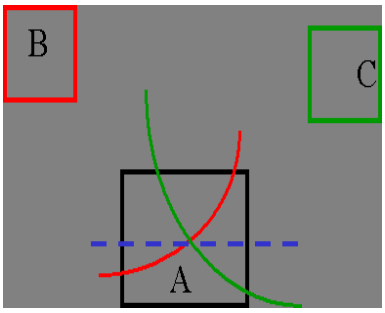

Figure 4: Geometrical reasoning in `distn`

of domain which is guided by the semantical properties of the constraints but their approach is not restricted to distance constraints.

6 Searching

Classical techniques for solving numerical CSP are based on a branch and prune algorithm. The pruning step is based on local consistency techniques such as 2B–consistency or Box–consistency . Sophisticated splitting strategies have been developed for finite domains but few results [19] are available for continuous domains. Roughly speaking, the standard search process is based on classical bisection techniques. In other words, the general solving scheme consist in a dichotomic enumeration interleaved with a local consistency filtering. More precisely, given a box $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the search selects a splitting direction k and splits the interval $\mathbf{x}_k$ in its middle $m(\mathbf{x}_k)$. Then, the two generated subproblems are solved.

Chronological backtracking is often used to handle the two corresponding CSP. However, more sophisticated strategies may also be used. For instance, a dynamic backtracking strategy for solving numerical CSP has been introduced by [17]. The authors replace the chronological backtracking by a non destructive backtracking technique.

Selecting the splitting direction is usually made by the Round Robin method (`RR`) : the domains of the variables are processed alternatively; this method is considered as the most efficient in average. Other domain selection strategies use syntactic or semantic information. The Largest First (`LF`) strategy –called geometric splitting [41]– consists in selecting the domain of maximal width. The Maximal Smear (`MS`) strategy has been introduced by [13] for interval newton methods. The selected domain maximizes the smear function[5]. Informally speaking, `MS` identifies the variable the projection of which has the strongest slope.

In [3], the authors introduce a new search strategy based on the distribution of the solutions within the domains. This search strategy takes advantage of the gaps that are computed during the filtering process for the domain of each variable, *i.e.* a set of disjoint intervals that do not contain any solution of the CSP.

These gaps provide very useful information for the selection of the domain to split as well as for choosing the point where the selected domain has to be split. Different heuristics that exploits these informations may be used, for instance some heuristics based on the width of the identified gaps. If no gap has been found, then bisection is used in combination with a classical domain selection strategy.

7 Conclusion

This paper did provide an overview of the constraint techniques for solving non-linear systems of equations over the real numbers. These

[5]The smear function of x_k is :

$$s_k = \max_{1 \le j \le m} \{\max\{|\underline{J}_{i,j}|, |\overline{J}_{i,j}|\} w(\mathbf{x}_i)\},$$

where $\mathbf{J}_{i,j} = [\underline{J}_{i,j}, \overline{J}_{i,j}]$ is the (i,j)-th entry of the interval extension of the jacobian matrix of the system

methods have shown their ability to locate (and often to prove the existence) of isolated solutions in a safe and rigorous way. Performances on diffi cult benchmarks can be found in [47, 39, 22].

Numerous benchmarks and successful applications on various areas can also be found on the following website:
http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html.

A new challenge concerns the optimization problems. Effi cient solvers for optimization problems are based on linear relaxations [43]. However, the latter are unsafe, and thus may overestimate, or worst, underestimate the very global minima. Interval methods and constraint techniques have recently been used to tackle optimization problems. First experimental results on global optimization problems can be found on the following web site:
http://www.mat.univie.ac.at/ neum/glopt/test.html.

Acknowledgments

Thanks to Yahia LEBBAH and Claude MICHEL for their careful reading of an earlier draft of this paper.

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] H. Batnini and M. Rueher. Décomposition sémantique pour la résolution de systèmes d'équations de distances *JEDAI(Journal Electronique d'Intelligence Artificielle)*, 2(1),2004.

[3] H. Batnini and M. Rueher and C. Michel. Mind the gaps. *Submitted for publication*, 2005.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] J. C. Cleary. Logical arithmetic. *Future Computing Systems*, pages 2(2) :125–149, 1Markot-2003987.

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

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

[12] C. A. Floudas, editor. *Deterministic global optimization: theory, algorithms and applications*. Kluwer Academic Publishers, 2000.

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

[14] M. Heusch. *distn: An Euclidean Distance Global Constraint.* Proc. of CP 2003,LNCS 2066,pp. 975 LNCS, 2003.

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

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

[17] N. Jussien and O. Lhomme. Dynamic Domain Splitting for Numeric CSPs. *European Conference on Artificial Intelligence*, pages :224–228,1998.

[18] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages 8:2:273–286, 1983.

[19] R.B. Kearfott. Tests of generalized bisection. *ACM Transactions on Mathematical Software*, pages 13(3):197–220, 1987.

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

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

[22] Y. Lebbah, C. Michel, M. Rueher, D. Daney and J.P. Merlet. Efficient and Safe Global Constraints for handling Numerical Constraint Systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.

[23] Y. Lebbah, M. Rueher, and C. Michel. A global filtering algorithm for handling systems of quadratic equations and inequations. *Lecture Notes in Computer Science*, 2470:109–123, 2002.

[24] Y. Lebbah, M. Rueher, and C. Michel. A rigorous global filtering algorithm for quadratic constraints *Selected papers of COCOS 2003*, LNCS 3478, 2005.

[25] Y. Lebbah, M. Rueher, and C. Michel. A rigorous global filtering algorithm for quadratic constraints *CONSTRAINTS Journal* 10()1,January 2005.

[26] 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.

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

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

[29] Mihály C. Markót. An Interval Method to Validate Optimal Solutions of the "Packing Circles in a Unit Square" Problems. Central European Journal of Operational Research 8:63–78,2000.

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

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

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

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

[34] A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence region. *Computing Supplementum*, 9:175–190, 1993.

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

[36] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *to appear in: Acta Numerica 2004 (A. Iserles, ed.), Cambridge University Press*, 2004.

[37] 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.

[38] G. Pesant and M. Boyer. Quad-clp(r): Adding the power of quadratic constraints. In *Proc. CP94 (Principles and Practice of Constraint Programming'94, LNCS 874*, pages 95–107, 1994.

[39] J. Puget and P. Van-Hentenryck. A constraints satisfaction approach to a circuit design problem. *Journal of global optimization*, 13(1):75–93, 1998.

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

[41] D. Ratz. Box-Splitting Strategies for the Interval Gauss–Seidel Step in a Global Optimization Method. *Computing*,53:337–354,1994.

[42] A. Rikun. A convex envelope formula for multilinear functions. *Journal of Global Optimization*, pages 10:425–437, 1997.

[43] V. Sahinidis, M. Twarmalani. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*. Kluwer Academic Publishers Group, 2002.

[44] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishing, 1999.

[45] H.D. Sherali and C.H. Tuncbilek. A global optimization algorithm for polynomial using a reformulation-linearization technique. *Journal of Global Optimization*, pages (2):101–112, 1992.

[46] N.Z. Shor. Dual quadratic estimates in polynomial and boolean programming. *Annals of Operations Research*, pages 25:163–168, 1990.

[47] 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.

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

[49] K. Yamamura, H. Kawata, and A. Tokue. Interval solution of nonlinear equations using linear programming. *BIT*, pages 38(1):186–199, 1998.