Constraints, 10, 47–65, 2005

A Rigorous Global Filtering Algorithm for Quadratic Constraints*

YAHIA LEBBAH ylebbah@sophia.inria.fr
COPRIN (I3S/CNRS–INRIA), Université de Nice–Sophia Antipolis, 930, route des Colles, B.P. 145, 06903, Sophia Antipolis Cedex, France
Département Informatique, Faculté des Sciences, Université d'Oran Es-Senia, B.P. 1524, El-M'Naouar Oran, Algeria

CLAUDE MICHEL AND MICHEL RUEHER {cpjm, rueher}@essi.fr
COPRIN (I3S/CNRS–INRIA), Université de Nice–Sophia Antipolis, 930, route des Colles, B.P. 145, 06903, Sophia Antipolis Cedex, France

Abstract. This article introduces a new filtering algorithm for handling systems of quadratic equations and inequations. Such constraints are widely used to model distance relations in numerous application areas ranging from robotics to chemistry. Classical filtering algorithms are based upon *local* consistencies and thus, are often unable to achieve a significant pruning of the domains of the variables occurring in quadratic constraint systems. The drawback of these approaches comes from the fact that the constraints are handled independently. We introduce here a *global* filtering algorithm that works on a tight linear relaxation of the quadratic constraints. The Simplex algorithm is then used to narrow the domains. Since most implementations of the Simplex work with floating point numbers and thus, are unsafe, we provide a procedure to generate safe linearizations. We also exploit a procedure provided by Neumaier and Shcherbina to get a safe objective value when calling the Simplex algorithm. With these two procedures, we prevent the Simplex algorithm from removing any solution while filtering linear constraint systems. Experimental results on classical benchmarks show that this new algorithm yields a much more effective pruning of the domains than local consistency filtering algorithms.

Keywords: global constraints, quadratic constraints, safe linearizations

1. Introduction

This article introduces a new filtering algorithm for handling systems of quadratic equations and inequations over the reals. Such *quadratic continuous constraints* can be formulated as follows:

$$\sum_{(i,j)\in N\times N} C_{i,j}^k x_i x_j + \sum_{i\in N} C_i^k x_i^2 + \sum_{i\in N} d_i^k x_i = b_k \tag{1}$$

with $C_{i,j}^k, C_i^k, d_i^k \in \mathbb{R}$ for all $(i,j) \in N \times N$ and $k \in 1\,..\,K$; N being the number of variables and K the number of quadratic constraints.

*This article is an extended version of [23].

For sake of simplicity, we will only consider equations in the rest of this article; handling of inequations is straightforward since our framework is based on the Simplex algorithm.

Quadratic constraints are widely used to model distance relations in numerous application areas ranging from robotics to chemistry. Thus, an efficient filtering of quadratic constraint systems is a key issue for solving many nonlinear constraint systems.

Classical filtering algorithms are based upon *local* consistencies such as 2*B*-consistency [24] or *Box*-consistency [6], and thus often achieve a very poor pruning of quadratic constraint systems. The drawback of these approaches comes from the fact that the constraints are handled independently and in a blind way. That's to say, classical local consistencies do not take advantage of the properties of quadratic constraints to reduce the domains of the variables.

3*B*-consistency and *kB*-consistency are partial consistencies which can achieve a better pruning since they are "less local" [14]. However, they require numerous splitting steps to find the solutions of a system of quadratic constraints; so, they may become rather slow.

The purpose of this article is to introduce a *global* filtering algorithm that works on a tight and safe linear relaxation of the quadratic constraints. This relaxation is adapted from a classical linearization method, the "Reformulation-Linearization Technique (RLT)" [30, 31]. 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.

This article provides a safe procedure to handle these two problems, and thus to prevent the linearization process and the Simplex algorithm from removing any solution. We take advantage of a result of A. Neumaier and O. Shcherbina that provides a cheap but rigorous process to correct the value of the objective function. This process exploits information from the dual problem to compute safe bounds of the approximated solutions.

The experimental results show that this new algorithm yields a much more effective pruning of the domains than filtering algorithms based upon local consistencies. It outperforms systems like Ilog Solver [17] and RealPaver [12] on the *Gough-Stewart platform* [10] benchmark. That is, the integration of *Quad* in a general branch and prune solver allows to tackle difficult applications with a significant subset of quadratic constraints.

Before going into the details, let us illustrate our framework on a short example.

1.1. An Illustrative Example

Consider the constraint system $\mathcal{C} = \{2xy + y = 1,\ xy = 0.2\}$ which represents two intersecting curves (see Figure 1).

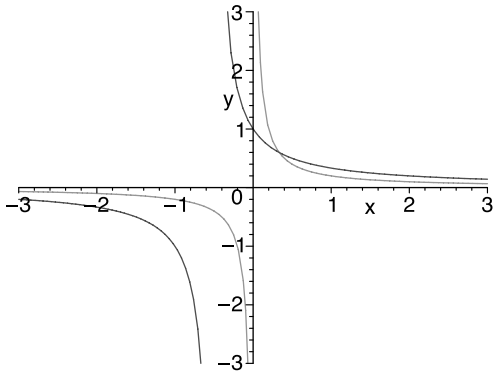

Figure 1. Geometrical representation of $\{2xy + y = 1,\ xy = 0.2\}$.

Suppose that $D_x = D_y = [-10, +10]$. Interval $[\underline{x}, \overline{x}]$ denotes the set of reals $\{r : \underline{x} \leq r \wedge r \leq \overline{x}\}$.

The reformulation-linearization technique (see Section 3) yields the following constraint system:

$$(a)\begin{cases} y + 2w = 1, \quad w = 0.2 \\ \underline{y}x + \underline{x}y - w \leq \underline{x}\,\underline{y}, \quad \overline{y}x + \underline{x}y - w \geq \underline{x}\,\overline{y}, \\ \underline{y}x + \overline{x}y - w \geq \overline{x}\,\underline{y}, \quad \overline{y}x + \overline{x}y - w \leq \overline{xy} \\ x \geq \underline{x}, \quad x \leq \overline{x}, \quad y \geq \underline{y}, \quad y \leq \overline{y} \\ w \geq \min\{\underline{x}\,\underline{y}, \overline{x}\,\underline{y}, \overline{xy}, \underline{x}\,\overline{y}\} \\ w \leq \max\{\underline{x}\,\underline{y}, \overline{x}\,\underline{y}, \overline{xy}, \underline{x}\,\overline{y}\} \end{cases}$$

where w is a new variable that stands for the product xy.

Substituting $\underline{x}, \underline{y}, \overline{x}$ and $\overline{y}$ by their values and minimizing (resp. maximizing) x, y and w with the Simplex algorithm yields the following new bounds:

$$D_x = [-9.38, 9.42], \quad D_y = [0.6, 0.6], \quad D_w = [0.2, 0.2]$$

By substituting the new bounds of x, y and w in the constraint system (a), we obtain the following linear constraint system:

$$(b)\begin{cases} y + 2w = 1, \quad w = 0.2 \\ 0.6x - 9.38y - w \leq -5.628 \quad 0.6x - 9.38y - w \geq -5.628 \\ 0.6x + 9.42y - w \geq 5.652, \quad 0.6x + 9.42y - w \leq 5.652 \\ x \geq -9.38, \quad x \leq 9.42, \quad y \geq 0.6, \quad y \leq 0.6, \\ w \geq 0.2, \quad w \leq 0.2 \end{cases}$$

Two more minimizing (resp. maximizing) steps of x, y and w are required to obtain the bounds displayed in Table 1.

Table 1. Filtering of $\{2xy + y = 1, xy = 0.2\}$ with 2*B*, *Box* and *Quad*

	2*B*	*Box*	*Quad*
$x \in$	[0, 10]	[0, 10]	[0.333333, 0.33333 . . . 4]
$y \in$	[−10, 10]	[−10, 10]	[0.600000, 0.600000]

Let us examine a second example that underlines the need for a rigorous computation of safe approximations of the linearizations. Consider the following set of linear constraints:

$$y - 2x \geq 0$$
$$y + 2x \geq 0$$
$$-y \geq k$$
with x and $y \in [-2.0, 2.0]$

A standard implementation of the Simplex algorithm over the floating point numbers (e.g., CPLEX [16]) finds the solution $x = 0$ and $y = 0$ when $k = 0$. However, suppose that k is the result of some computation with a rounding to $+\infty$, then the value of k is $4.94065\ldots \times 10^{-324}$, which is nothing but the float following 0. Then, no solution is found by CPLEX, even when the rigorous corrections from Neumaier and Shcherbina [28] are applied. The safe rounding procedure introduced in this article preserves all the solutions, that is to say, $x = 0$ and $y = 0$ for the considered example.

1.2. Outline

Section 2 defines the notations and points out the limits of local consistencies. Section 3 gives an overview of our framework and introduces different relaxation classes. Section 4 details the filtering process. The safe rounding procedure is introduced in Section 5. Finally, Section 6 reports some experimental results.

2. Preliminaries

2.1. Notations

This article 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.

$D_x = [\underline{x}, \overline{x}]$ denotes the domain of variable 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.

Let $e \in \mathbb{R}$, $\Delta(e)$ (resp. $\nabla(e)$) denotes the smallest (resp. the biggest) floating point number $e_{\mathbb{F}}$ such that $e \leq e_{\mathbb{F}}$ (resp. $e \geq e_{\mathbb{F}}$). For convenience, $\Delta(e)$ (resp. $\nabla(e)$), where e stands for an arithmetic expression, means that any basic operation has to be done according to the specified rounding direction, i.e., with a rounding to $+\infty$ (resp. to $-\infty$). We assume an IEEE 754 floating point unit, that is to say, that basic operations like +, −, /, * are exactly rounded [15].

The RLT (reformulation-linearization technique) [31] notation $[E]_L$ stands for E where the quadratic terms have been replaced by new variables in expression E. $[c]_R$ denotes the set of linear inequalities that approximate the nonlinear terms of constraint c.

2.2. *Limits of Local Consistencies*

Formal definitions of 2*B*-consistency and *Box*-consistency can be found in [14]. We will just recall here the basic idea of these local consistencies.

2*B*-consistency [24] states a local property on the bounds of the domains of a variable at a single constraint level. Roughly speaking, constraint c is 2*B*-consistent if, for any variable x, there exists values in the domains of all other variables which satisfy c when x is fixed to $\underline{x}$ or $\overline{x}$.

Box-consistency [6] is a coarser relaxation than 2*B*-consistency. It mainly consists of replacing every existentially quantified variable but one with its interval in the definition of 2*B*-consistency.

Different approximations have been introduced in the implementations of the corresponding filtering algorithms:

- 2*B*-filtering decomposes the initial constraints in binary and ternary basic constraints for which it is trivial to compute the projection functions [9, 24].
- *Box*-filtering generates a system of univariate interval functions which can be tackled by numerical methods such as Newton. Contrary to 2*B*-filtering, *Box*-filtering does not require any constraint decomposition of the initial constraint systems. However, *Box*-filtering requires the generation of a new constraint for each variable of each constraint.

The main drawback of these local consistencies is *the over-estimation of interval functions*. This is due to two well known problems:

- the so-called *wrapping effect* [19], which overestimates by a unique vector the image of an interval vector (which is in general not a vector).
- the so-called *dependency problem* [13], which is due to the independence of the different occurrences 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 $X = [0, 10]$, then $X - X = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-10, 10]$ instead of [0, 0] as one could have expected.

The success of 2*B*-consistency depends on the precision of the generated projection functions. However, 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 but a local consistency like 2*B*-consistency is not preserved by the decomposition. Indeed, the decomposition amplifies the dependency problem, and thus, it yields a weaker filtering (see [14] for a detailed discussion of this point).

In this article, we introduce an efficient way to handle quadratic constraints without using direct projection functions. Indeed, the difficulty when solving quadratic equations comes from the quadratic terms $f(x) = x^2$ and $g(x, y) = xy$: f is a convex function whereas g is neither concave nor convex. It is easy to underestimate convex functions such as f, whereas it is difficult to handle non-convex and non-concave functions. That is why we will define tight approximations of these constraints. Next section defines the linear relaxations we have used to define this approximations.

3. Using Linear Programming Techniques to Filter Quadratic Constraints

The goal is to transform the quadratic constraint system into a linear program to be able to approximate the upper and lower bounds of the variables with the Simplex algorithm.

Quadratic constraints may be approximated by linear constraints in the following way:

- create a new variable for each quadratic term:
  - v for x^2 with domain $\left[\underline{v}, \max\left(\underline{x}^2, \overline{x}^2\right)\right]$ where $\underline{v} = 0$ if $0 \in D_x$, $\underline{v} = \min\left(\underline{x}^2, \overline{x}^2\right)$ otherwise.
  - w for xy with domain $[\min\{\underline{x}\,\underline{y}, \underline{x}\,\overline{y}, \overline{x}\underline{y}, \overline{xy}\}, \max\{\underline{x}\,\underline{y}, \underline{x}\,\overline{y}, \overline{x}\underline{y}, \overline{xy}\}]$.
- linearize each quadratic constraint of the constraint system (1) by using the previous new variables; we denote the produced system

$$\left[\sum_{(i,j)\in N\times N} C^k_{i,j} x_i x_j + \sum_{i\in N} C^k_i x_i^2 + \sum_{i\in N} d^k_i x_i = b_k\right]_L$$

  with $k \in 1 \ldots K$. N being the number of variables and K the number of quadratic constraints. $[E]_L$ denotes E where the quadratic terms have been replaced by their variables.

So, we obtain a first naive linear relaxation of the quadratic constraints (1). We denote it LP1. Of course, if no quadratic terms occurs in the initial constraints, LP1 is an exact relaxation. As stated before, we have to introduce a new linear relaxation to capture the semantic of quadratic terms. A tight linear (convex) relaxation, or outer-approximation to the convex and concave envelope[1] of the quadratic terms over the constrained region, is built by generating new linear inequalities. In the next sub-sections we introduce two

tight linear relaxation classes that preserve most of the semantic of equations $v = x^2$ and $w = xy$ and that provide a better approximation than interval arithmetic.

3.1. Linearization of Quadratic Terms

The problem of quadratic term linearization has been studied in quadratic optimization; for a deeper overview see [1, 3, 30]. We introduce here a simplification of these linearizations adapted to our purpose.

The proposed approach is based on Reformulation-Linearization Techniques (RLT) [30].

Linearization of x^2

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

$$[x^2]_R = \begin{cases} L1(\alpha) \equiv [(x-\alpha)^2 \geq 0]_L \text{ where } \alpha \in [\underline{x}, \overline{x}] \\ L2 \equiv [(\underline{x}+\overline{x})x - y - \underline{x}\overline{x} \geq 0]_L \end{cases} \tag{2}$$

Inequality $L1(\alpha)$ trivially holds whereas inequality $L2$, stated by [2, 26], comes from the following valid inequality:

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

Note that $[(x-\alpha)^2 = 0]_L$ generates the tangent line to the curve $y = x^2$ at the point $x = \alpha$. Actually, *Quad*, the new algorithm we propose, computes only $L1(\overline{x})$ and $L1(\underline{x})$. Consider for instance the quadratic term x^2 with $x \in [-4, 5]$. Figure 2 displays the initial

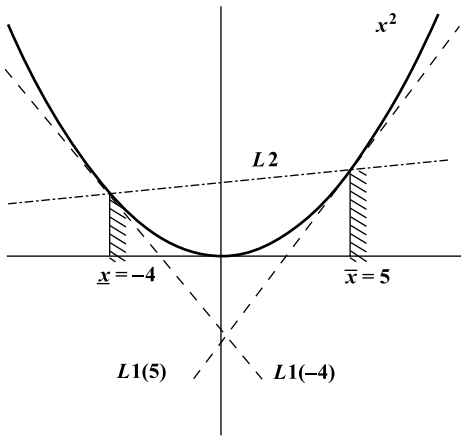

Figure 2. Illustration of x^2 relaxations.

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.

Of course, the selection of additional points for the $L1$ underestimations improves the precision of the approximation. However, it also increases the size of the linear constraint system. A key issue is to find a good balance between the precision of the approximation and the number of generated inequalities. Up to now, we did not find any application where adding more approximations did pay off.

Linearization of xy

In the case of bilinear terms xy, [26] has proposed the following relaxations of xy over the box $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$:

$$[xy]_R = \begin{cases} BIL1 \equiv [(x - \underline{x})(y - \underline{y}) \geq 0]_L \\ BIL2 \equiv [(x - \underline{x})(\overline{y} - y) \geq 0]_L \\ BIL3 \equiv [(\overline{x} - x)(y - \underline{y}) \geq 0]_L \\ BIL4 \equiv [(\overline{x} - x)(\overline{y} - y) \geq 0]_L \end{cases} \tag{3}$$

BIL1 and BIL3 define a convex envelope of xy whereas BIL2 and BIL4 define a concave envelope of xy over the box $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$. Al-Khayyal and Falk [2] showed that these relaxations are the optimal convex/concave outer-estimations of xy.

Consider for instance the quadratic term xy with $x \in [-5, 5]$ and $y \in [-5, 5]$. The work done by the linear relaxations of the 3D curve $z = xy$ is well illustrated in 2D by fixing z. Figure 3 displays the 2D shape for the level $z = 5$ of the initial curve (i.e., $xy = 5$), and the

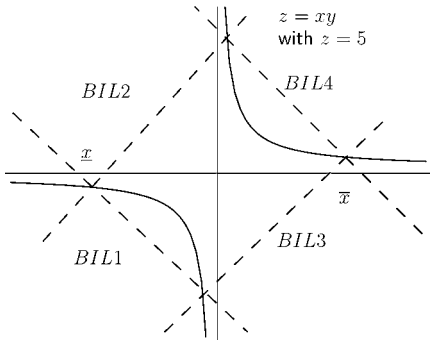

Figure 3. Illustration of xy relaxations.

lines corresponding to the equations generated by the relaxations (where $z = 5$): $BIL1 \equiv z + 5x + 5y + 25 \geq 0$, $BIL2 \equiv -z + 5x - 5y + 25 \geq 0$, $BIL3 \equiv -z - 5x + 5y + 25 \geq 0$, and $BIL4 \equiv z - 5x - 5y + 25 \geq 0$.

The observant reader may have noticed that the relaxation of xy do not take advantage of the discontinuity of the curve (see Figure 3). However, as soon as the splitting process starts, the relaxations become much more tighter.

Many other relaxations have been proposed by [3, 30]. For simplicity, we prefer to restrict the presentation to these two classes which are sufficient to achieve a better filtering than 2*B*-consistency and *Box*-consistency do.

Next section gives an overview of the whole filtering algorithm of quadratic constraints.

4. *Quad*, The New Filtering Algorithm

Now, we are in position to define *LRQ*, the set of linear relations we use to approximate quadratic constraints:

$$LRQ \equiv \begin{cases} \left[\sum_{(i,j)\in N\times N} C^k_{i,j} x_i x_j + \sum_{l\in N} C^k_l x_l^2 + \sum_{i\in N} d^k_i x_i = b_k\right]_L \\ L1(\underline{x}_l),\ L1(\overline{x}_l) \text{ for all } x_l \text{ with } l \in N \\ L2 \text{ for all } x_l \text{ with } l \in N \\ BIL1, BIL2, BIL3, BIL4 \text{ for all } x_i x_j \text{ with } (i,j) \in N \times N \end{cases}$$

LRQ will be embedded in a linear program (LP) for filtering quadratic constraints.

The filtering process is shown in Algorithm 1.

At each iteration step, the reduction achieved by the Simplex algorithm must be greater than ϵ for at least one bound of some variable. The iteration process is stopped as soon as the domain of some variable becomes empty. So, the algorithm converges and terminates if ϵ is greater than zero.

The next section introduces a procedure to get a safe *LRQ* approximation of the initial constraint system with a safe call to the Simplex algorithm. Hence, it guarantees the completeness of the *Quad* algorithm.

5. A Safe Procedure for the *Quad* Algorithm

The linearizations involved in the *Quad* algorithm preserve all the solutions over the reals as long as their coefficients are exactly computed. The point is that these coefficients are generally computed with floating point numbers. Computations with floating point numbers require rounding operations that may introduce some approximations of the initial linearizations.

Algorithm 1: The *Quad* algorithm

Function $Quad\text{–}filtering(\texttt{IN}: \mathcal{D}, Q, \epsilon)$ **return** $\mathcal{D}'$
% $\mathcal{D}$: input domains; Q: quadratic constraints (1)
% ϵ: minimal reduction
$\mathcal{D}' := \mathcal{D}$
do
  for all $x \in vars(Q)$ **do**
    Construct and solve the following LP
      $\begin{cases} Z = minimize(x) \\ w.r.t.LRQ \text{ from the quadratic constraints } Q \end{cases}$
    $\underline{x}' := max(\underline{x}, Z)$
    Solve the same LP with $Z = maximize(x)$
    $\overline{x}' := min(\overline{x}, Z)$
  endfor
while the reduction amount of some bound is greater than ϵ **and** $\emptyset \notin \mathcal{D}'$

First, we show how to guarantee that the computed approximations actually contain all the solutions over the reals. Second, we point out a procedure of Neumaier and Shcherbina which ensures a safe behaviour of the Simplex algorithm on floating point numbers.[2]

5.1. Computing Safe Coefficients

A Safe Approximation of $L1$

The linearizations $L1(\alpha) \equiv y - 2\alpha x + \alpha^2$, where y stands for the variable which replaces x^2 in the linear constraints, defines a convex approximation of the term x^2. Linearization $L1(\alpha)$ generates the tangent lines to the curve $y = x^2$ at the point (α, α^2).

The computation with floating point arithmetic of -2α and α^2 may change the slope as well as the intersection point with the y axis. *Quad* computes only this linearization for the points $(\underline{x}, \underline{x}^2)$ and $(\overline{x}, \overline{x}^2)$. So, α is always a float and the computation of the safe approximation is somewhat simplified. Roughly speaking, consider the case where $\alpha \geq 0$, the slope is decreased as well as the intersection with the y axis. Therefore, we have just to compute an approximation that defines a line which, for any $x \geq 0$, is under the line defined by the linearization $L1$. As all solutions are above $L1$, the approximation $L1_{\mathbb{F}}(\alpha)$ of the $L1(\alpha)$ linearization, is conservative (See Proposition 1).

Proposition 1 *(*$L1_{\mathbb{F}}(\alpha)$ *approximations [27]).*
Assume that $L1(\alpha)$ *holds, and let* $\alpha \in \mathbb{F}$ *and*

$$L1_{\mathbb{F}}(\alpha) \equiv \begin{cases} y - \nabla(2\alpha)x + \Delta(\alpha^2) \geq 0 & \textit{iff } \alpha \geq 0 \\ y - \Delta(2\alpha)x + \Delta(\alpha^2) \geq 0 & \textit{iff } \alpha < 0 \end{cases}$$

Then, for all $x \in D_x$, *and* $y \in \left[0, max\ \underline{x}^2, \overline{x}^2\right\}\big]$, $L1_{\mathbb{F}}(\alpha)$ *holds.*

Proof: Let us consider the first case. In order to prove that $L1_{\mathbb{F}}(\alpha)$ holds whenever $L1(\alpha)$ holds, we will show that the following relation holds:

$$y - \nabla(2\alpha)x + \Delta(\alpha^2) \geq y - 2\alpha x + \alpha^2 \geq 0 \text{ for all } x \in D_x \text{ and } y \in [0, \max\{\underline{x}^2, \overline{x}^2\}] \tag{4}$$

It results from the rounding to $-\infty$ that $\nabla(2\alpha) \leq 2\alpha$. So, we have $\nabla(2\alpha)\, x \leq 2\alpha x$ for all positive x and then $-\nabla(2\alpha)x \geq -2\alpha x$.

Using the properties of the rounding to $+\infty$, we have $\Delta(\alpha^2) \geq \alpha^2$. Therefore $y - \nabla(2\alpha)\, x + \Delta(\alpha^2) \geq y - 2\alpha x + \alpha^2$ holds for all positive x. The second case can be proved in a similar way. ■

Safe Approximations of L2

The case of $L2$ is a bit more tricky since the "rotation axis" of the line defined by $L2$ could be between the extremum values of x^2 ($L2$ is an overestimation of x^2). Thus, to keep all solutions we have to strengthen the slope of this line at its smallest extremum. Proposition 2 introduces a safe apporximation $L2_{\mathbb{F}}$ of $L2$:

Proposition 2 *($L2_{\mathbb{F}}$ approximations [27]).*
Assume that L2 holds, and let

$$L2_{\mathbb{F}} \equiv \begin{cases} \Delta(\underline{x}+\overline{x})x - y - \nabla(\underline{x}\overline{x}) \geq 0 \textit{ iff } \underline{x} \geq 0 \\ \nabla(\underline{x}+\overline{x})x - y - \nabla(\underline{x}\overline{x}) \geq 0 \textit{ iff } \overline{x} < 0 \\ \Delta(\underline{x}+\overline{x})x - y - \nabla(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}) \geq 0 \textit{ iff } \overline{x} > 0 \wedge \underline{x} < 0 \wedge |\underline{x}| \leq |\overline{x}| \\ \nabla(\underline{x}+\overline{x})x - y - \nabla(\underline{x}\overline{x} - \Delta(Ulp(\Delta(\underline{x}+\overline{x}))\overline{x})) \geq 0 \textit{ iff } \overline{x} > 0 \wedge < 0 \wedge |\underline{x}| > |\overline{x}| \end{cases}$$

where Ulp(a)—which stands for Unit in the last place*—computes the distance between floating point number a and its predecessor.*

Then, for all $x \in D_x$ and $y \in \left[0,\ max\ \left\{\underline{x}^2, \overline{x}^2\right\}\right]$, $L2_{\mathbb{F}}$ holds.

Proof: The proofs of the first and second cases are similar to the proof of $L1_{\mathbb{F}}$.

The proof of the third case is a bit more tricky. The essential observation is that both $L2_{\mathbb{F}}$ and $L2$ contain the same term y. Thus the proof reduces to show that the following relation holds:

$$\Delta(\underline{x}+\overline{x})x - \nabla(\underline{x}\overline{x} + Ulp(\,\Delta(\underline{x}+\overline{x}))\underline{x}) \geq (\underline{x}+\overline{x})x + \underline{x}\overline{x} \geq 0, \forall x \in [\underline{x}, \overline{x}] \tag{5}$$

First, let us consider the coefficient of the slope. Rounding to $+\infty$ ensures that $\Delta(\underline{x}+\overline{x}) = (\underline{x}+\overline{x}) + \epsilon$ with $Ulp(\Delta(\underline{x}+\overline{x})) \geq \epsilon \geq 0$. Thus, we just need to show that

$$\epsilon x - \nabla(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}) \geq \underline{x}\overline{x} \text{ for all } x \in [\,\underline{x}, \overline{x}]$$

Due to the properties of the rounding to $-\infty$, we have

$$\nabla(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}) \leq \underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}$$

Thus

$$-\nabla(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}) \geq -(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x})$$

and

$$\epsilon x - \nabla(\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}) \geq \epsilon x - (\underline{x}\overline{x} + Ulp(\Delta(\underline{x}+\overline{x}))\underline{x})$$

Therefore, relation (5) holds if $\epsilon x - Ulp(\Delta(\underline{x}+\overline{x}))\underline{x} \geq 0$. That is to say, relation (5) holds if $\epsilon x \geq Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}$ holds.

As $Ulp(\Delta(\underline{x}+\overline{x})) \geq \epsilon \geq 0$ and $\underline{x} \leq 0, \epsilon x \geq Ulp(\Delta(\underline{x}+\overline{x}))\underline{x}$ holds for all $x \in [0, \overline{x}]$. When $x \in [\underline{x}, 0]$, we have $\min(\epsilon x) = \epsilon\underline{x}$ since ϵ is positive. Here again, as $Ulp(\Delta(\underline{x}+\overline{x})) \geq \epsilon$, then relation (5) holds for any $x \in [\underline{x}, 0]$. The fourth case can be proved with a similar reasoning. ■

Note that the condition $|\underline{x}| \leq |\overline{x}|$ is not used by the proof. We only differentiate third and fourth cases to obtain a better correction.

Note that a finer correction of the third and fourth cases could be obtained by replacing the function $Ulp(a)$ by $\left|\Delta(a) - \nabla(a)\right|$. This would avoid to add an unnecessary correction when the computation of the slope is exact.

Safe Approximations of BIL1, BIL2, BIL3 and BIL4

The general form of linearizations *BIL*1, *BIL*2, *BIL*3 and *BIL*4 is

$$w + s_1 b_1 x + s_2 b_2 y + s_3 b_1 b_2 \geq 0$$

where the b_k are floating point numbers corresponding to the bounds of x and y, and where $s_i \in (-1, 1)$. Thus, the coefficients of the linearization are always computed exactly.

The rounding operation only applies to the computation of the constant values. So, these linear constraints can be rewritten in the following form: $Y + s_3 b_1 b_2 \geq 0$.

A rounding of $s_3 b_1 b_2$ to $+\infty$ enlarges the solution space, and thus ensures that all these linear constraints are safe approximations of xy.

It follows that $BIL1_{\mathbb{F}}, BIL2_{\mathbb{F}}, BIL3_{\mathbb{F}}$ *and* $BIL4_{\mathbb{F}}$ are equivalent to $Y + \Delta\{s_3 b_1 b_2\} \geq 0$.

5.2. Correction of the Simplex Algorithm

The Simplex method can solve linear problems of the following form:

$$\begin{aligned} &\text{minimize } c^T x \\ &\text{subject to } \underline{b} \leq Ax \leq \overline{b} \end{aligned}$$

The solution of such a problem is a vector $x_{\mathbb{R}} \in \mathbb{R}^n$. However, the solution computed by solvers like CPLEX [16] or Soplex [33] is a vector $x_{\mathbb{F}} \in \mathbb{F}^n$ that may differ from $x_{\mathbb{R}}$ due to rounding errors. More precisely, $x_{\mathbb{F}}$ is safe for the objective if $c^T x_{\mathbb{R}} \geq c^T x_{\mathbb{F}}$.

Neumaier and Shcherbina [28] have provided a cheap method to obtain a rigorous bound of the objective. The essential observation is that the dual of the problem is:

$$\begin{aligned} &\text{maximize } \underline{b}^T y - \overline{b}^T z \\ &\text{subject to } A^T(y-z) = c, y \geq 0, z \geq 0 \end{aligned}$$

Associated with the approximate solution of the dual is an approximate multiplier $\lambda \approx y - z$ which is used to compute a rigorous interval enclosure of the residual: $r = A^T\lambda - c$. It follows that

$$c^T x = \left(A^T\lambda - r\right)^T x = \lambda^T A x - r^T x \in \lambda^T [\underline{b}, \overline{b}] - r^T [\underline{x}, \overline{x}]$$

and the value of μ, the lower bound of the value of the objective function is:

$$\mu = \inf\left(\lambda^T [\underline{b}, \overline{b}] - r^T [\underline{x}, \overline{x}]\right)$$

This value is trivially correct by construction. Note that the precision of such a safe bound depends on the width of the interval $[\underline{x}, \overline{x}]$.

The computation of r and μ could be rigorously done using interval arithmetic. Neumaier et al. [28] have also proposed a certificate of infeasibility when the Simplex algorithm is in the infeasible state.

So, we have to apply this correction before updating the lower bound or the upper bound of each variable to ensure that the obtained domains contain all the solutions.

6. Experimentations

To evaluate the contribution of the *Quad* algorithm we have compared its performances with classical filtering algorithms[3] (i.e., 2*B*-filtering, *Box*-filtering and 3*B*-filtering) and with the Ilog Solver system [17] on two benchmarks: a planar case of the Gough-Stewart platform [10] and a Kinematics application, named "kinema" [5]. We have performed the following experimentations:

- Filtering initial domains containing exactly one solution;
- Filtering initial domains containing many solutions;
- Combining filtering and splitting to isolate all the solutions.

Experimentations covering 2*B*-filtering, *Box*-filtering and 3*B*-filtering have been performed with the implementation of iCOs [20, 21], one of the most efficient library for this kind of algorithms [7].

Quad has been implemented with the linear programming solver "CPLEX" [16]. *Quad* and Ilog Solver use the same splitting strategies to isolate the different solutions. The tests with Ilog Solver uses a *Box*-filtering which is based on *Numerica* [32]. All the experimentations have been run on PC-Notebook/1Ghz. CPU times are given in seconds.

6.1. The Gough-Stewart Platform Benchmark

The first benchmark [10] comes from robotics and describes the kinematics of a planar case of the Gough-Stewart platform.

Problem 1 *(Gough-Stewart [10])*

$$\begin{cases} x_1^2+y_1^2+z_1^2=31;\ x_2^2+y_2^2+z_2^2=39;\ x_3^2+y_3^2+z_3^2=29; \\ x_1x_2+y_1y_2+z_1z_2+6x_1-6x_2=51; \\ x_1x_3+y_1y_3+z_1z_3+7x_1-2y_1-7x_3+2y_3=50; \\ x_2x_3+y_2y_3+z_1z_3+x_2-2y_2-x_3+2y_3=34; \\ -12x_1+15y_1-10x_2-25y_2+18x_3+18y_3=-32; \\ -14x_1+35y_1-36x_2-45y_2+30x_3+18y_3=8; \\ 2x_1+2y_1-14x_2-2y_2+8x_3-y_3=20; \\ x_1\in[-2.00,5.57];\ y_1\in[-5.57,2.70];\ z_1\in[0,5.57] \\ x_2\in[-6.25,1.30];\ y_2\in[-6.25,2.70];\ z_2\in[-2.00,6.25] \\ x_3\in[-5.39,0.70];\ y_3\in[-5.39,3.11];\ z_3\in[-3.61,5.39] \end{cases}$$

Table 2 shows the results of the different filtering algorithms for some domains containing only one solution. The symbol "−" means that no reduction has been done. *Quad* is the only algorithm who isolates the unique solution. These results outline the advantage of the global view of *Quad* over the local or partial view of the other filtering algorithms.

Table 3 shows that the different algorithms yield almost the same pruned domains for some initial domains containing several solutions. Indeed, this first pruning is rather poor

Table 2. Filtering results on the Gough-Stewart problem for some domains containing only one solution

	Initial domains	2*B*	*Box*	3*B*	*Quad*
x_1	[0.00, 5.57]	[0.00, 5.56]	[0.00, 5.56]	[0.55, 4.68]	[2.93, 2.93]
y_1	[0.00, 2.70]	–	–	–	[0.45, 0.45]
z_1	[0.00, 5.57]	[0.00, 5.56]	[0.00, 5.56]	[1.14, 5.54]	[4.70, 4.70]
x_2	[−6.00, 0.00]	[−4.50,−0.15]	[−4.50,−0.15]	[−4.07,−0.84]	[−1.81,−1.81]
y_2	[−2.00, 0.00]	–	–	–	[−0.48,−0.48]
z_2	[0.00, 6.25]	[3.83, 6.24]	[3.83, 6.24]	[4.43, 6.17]	[5.95, 5.95]
x_3	[−5.39,−1.00]	[−5.38,−1.00]	[−5.38,−1.00]	[−5.05,−1.00]	[−1.66,−1.66]
y_3	[−5.39, 0.00]	[−5.29, 0.00]	[−5.29, 0.00]	[−4.38, 0.00]	[−0.20,−0.20]
z_3	[0.00, 5.39]	[0.00, 5.29]	[0.00, 5.29]	[0.91, 5.29]	[5.11, 5.11]

Table 3. Filtering results on the Gough-Stewart problem for some domains containing many solutions

	Initial domains	2*B*	*Box*	3*B*	*Quad*
x_1	[−2.00, 5.57]	[−2.00, 5.56]	[−2.00, 5.56]	[−2.00, 5.44]	–
y_1	[−5.57, 2.70]	[−5.56, 2.70]	[−5.56, 2.70]	[−5.56, 2.70]	–
z_1	[0.00, 5.57]	[0.00, 5.56]	[0.00, 5.56]	[0.00, 5.56]	–
x_2	[−6.25, 1.30]	[−6.19, 1.30]	[−6.19, 1.30]	[−5.64, 0.75]	[−5.17, 0.34]
y_2	[−6.25, 2.70]	[−6.24, 2.70]	[−6.24, 2.70]	[−6.24, 2.70]	[−1.77, 2.70]
z_2	[−2.00, 6.25]	[−2.00, 6.24]	[−2.00, 6.24]	[−2.00, 6.24]	–
x_3	[−5.39, 0.70]	[−5.38, 0.69]	[−5.38, 0.69]	[−5.38, 0.69]	[−5.38, 0.69]
y_3	[−5.39, 3.11]	[−5.38, 3.10]	[−5.38, 3.10]	[−5.38, 3.10]	[−5.39, 3.10]
z_3	[−3.61, 5.39]	[−3.60, 5.38]	[−3.60, 5.38]	[−3.60, 5.38]	[−3.60, 5.38]

since it has to preserve all solutions. Like many strong filtering, *Quad* becomes really efficient when the splitting process starts.

Table 4 provides the execution timing for finding all the solutions on the initial domains. *Quad* outperforms Ilog Solver and "interval Newton" on this problem. Ilog Solver was run with *Box*-consistency; results were worst with *Bound*-consistency [29]. *Quad* is also about 3 times faster than RealPaver [12].

More than 3 hours CPU time are required to isolate the solutions when using 3*B*-consistency filtering and a splitting process.

The results of "interval Newton" are those published by [10]; the computations where done on a Pentium 90 computer, more than 10 time slower than our computer. Removing multiple occurrences of the variables, enables [10] to solve the problem in about three hours. Didrit [10] solves in 24 minutes an other formulation of that problem which is far from being obvious and where three variables are removed. Note that *Quad* requires very few splittings to find all the solutions of this problem. So, even if the filtering process is more complex, it really pays off on difficult problems.

6.2. The Kinematics Benchmark

Now, let us consider a kinematics application, named "kinema" [5]. This second benchmark describes a robot kinematics problem. The results obtained with the different algorithms are similar to the previous one.

Table 4. Filtering performance for the Gough-Stewart problem

	Interval Newton	Ilog Solver	*Quad*	RealPaver
CPU time	14400	371.6	24.41	78.2
Splittings	?	45612	95	–

Table 5. Filtering results on the kinematics problem for some domains containing a single solution

	Initial domains	2*B*	*Box*	3*B*	*Quad*
z_1	[11, 13]	–	–	–	[11.99, 12.00]
z_2	[7, 9]	–	–	–	[7.99, 8.00]
z_3	[1, 3]	–	–	–	[1.99, 2.00]
z_4	[7, 9]	–	–	–	[7.99, 8.00]
z_5	[11, 13]	–	–	–	[11.99, 12.00]
z_6	[1, 3]	–	–	–	[1.99, 2.00]
z_7	[7, 9]	–	–	–	[7.99, 8.00]
z_8	[15, 17]	–	–	–	[15.99, 16.00]
z_9	[5, 7]	–	–	–	[5.99, 6.00]

Problem 2 *(Kinema [5])*

$$\begin{cases} z_1^2 + z_2^2 + z_3^2 - 12z_1 - 68 = 0; \\ z_4^2 + z_5^2 + z_6^2 - 12z_5 - 68 = 0; \\ z_7^2 + z_8^2 + z_9^2 - 24z_8 - 12z_9 + 100 = 0; \\ z_1z_4 + z_2z_5 + z_3z_6 - 6z_1 - 6z_5 - 52 = 0; \\ z_1z_7 + z_2z_8 + z_3z_9 - 6z_1 - 12z_8 - 6z_9 + 64 = 0; \\ z_4z_7 + z_5z_8 + z_6z_9 - 6z_5 - 12z_8 - 6z_9 + 32 = 0; \\ 2z_2 + 2z_3 - z_4 - z_5 - 2z_6 - z_7 - z_9 + 18 = 0; \\ z_1 + z_2 + 2z_3 + 2z_4 + 2z_6 - 2z_7 + z_8 - z_9 - 38 = 0; \\ z_1 + z_3 - 2z_4 + z_5 - z_6 + 2z_7 - 2z_8 + 8 = 0; \\ z_i \in [-100, 100], i = 1 \ldots 9 \end{cases}$$

Table 5 shows the results of the different filtering algorithms for some domains containing only one solution. *Quad* is again the only algorithm who isolates the unique solution.

The pruning achieved by *Quad* is relatively more significant than 2*B*/*Box*/3*B*-filtering when the initial domains contain several solutions (see Table 6).

Table 6. Filtering results on the kinematics problem for some domains containing many solutions

	Init. Dom.	2*B*	*Box*	3*B*	*Quad*
z_1	[−100, 100]	[−5.66, 16.19]	[−5.66, 16.19]	[−5.66, 16.19]	[−4.19, 16.19]
z_2	[−100, 100]	[−16.19, 16.19]	[−16.19, 16.19]	[−12.14, 12.65]	[−11.54, 12.85]
z_3	[−100, 100]	[−16.19, 16.19]	[−16.19, 16.19]	[−12.14, 12.65]	[−6.62, 12.85]
z_4	[−100, 100]	[−16.19, 16.19]	[−16.19, 16.19]	[−12.14, 12.65]	[−7.91, 12.85]
z_5	[−100, 100]	[−5.66, 16.19]	[−5.66, 16.19]	[−5.66, 16.19]	[−4.19, 16.19]
z_6	[−100, 100]	[−16.19, 16.19]	[−16.19, 16.19]	[−12.14, 12.65]	[−4.47, 12.85]
z_7	[−100, 100]	[−32.96, 32.96]	[−32.96, 32.96]	[−18.45, 15.43]	[−11.36, 16.26]
z_8	[−100, 100]	[−12.31, 32.96]	[−12.31, 32.96]	[0.96, 24.27]	[−1.13, 25.13]
z_9	[−100, 100]	[−32.96, 32.96]	[−20.96, 32.96]	[−5.12, 19.11]	[−9.41, 21.41]

Table 7. Filtering performance for the kinematics problem

	Ilog Solver	*Quad*	RealPaver
CPU time	762.60	33.40	824.12
Splittings	244000	220	–

Table 7 shows that *Quad* outperforms Ilog Solver and RealPaver [12]. We note also that *Quad* requires much less splittings than Ilog Solver to find the solutions.

7. Conclusion

This article has introduced a new algorithm for handling systems of quadratic constraints. This algorithm performs a global filtering on a linear relaxation of the initial constraint system. Experimentations on robotics benchmarks are very promising and show the capabilities of this framework. We have also introduced a procedure which allows us to tackle linear relaxation of continuous CSPs without losing any solution. Though these corrections were defined in the context of the *Quad* algorithm, the principles presented here could be extended to other problems where the Simplex algorithm has to be integrated into a CSP system.

Quad may be integrated in a general branch and prune solver or a cooperative framework [25] to tackle non-polynomial problems with a significant subset of quadratic constraints. Combining *Quad* with local consistencies and interval methods is also a very promising approach for solving numerical CSP [22].

So, *Quad* could play the role of global constraint for handling all quadratic constraints in many geometric or robotics applications where numerous distance constraints occur.

Acknowledgments

Thanks to Prof. Arnold Neumaier who suggested us to explore the capabilities of linearization techniques for a global handling of distance relations. Thanks to Bertrand Neveu for his careful reading of an earlier draft of this article, and to the anonymous reviewer for their helpful comments. We also like to thank Jean-Pierre Merlet for fruitful discussions.

Notes

1. "Let $f : S \to E_1$, where $S \subseteq E_n$ is a nonempty convex set. Then, the **convex envelope** of f over S, denoted $f_s(x)$, $x \in S$, is a convex function such that: (1) $f_s(x) \leq f(x)$ for all $x \in S$, (2) if g is any other convex function for which $g(x) \leq f(x)$ for all $x \in S$, then $f_s(x) \geq g(x)$ for all $x \in S$. Hence, $f_s(x)$ is the component-wise supremum over all convex

underestimation of f over S. **Concave envelope** is defined in a similar way, and defines component-wise infemum over all convex overestimation of f over S'' [4], page 125.

2. The Simplex algorithm performs safe computations when it works with rational numbers. In this case no rounding problems occur but computations with rational numbers may be very time and space consuming.
3. Alternative methods have been proposed for solving nonlinear systems. For instance, algebraic constraints can be handled with symbolic methods [8] (e.g., Groebner Basis, Resultant). However, these methods can neither handle non-polynomial systems nor deal with inequalities.

References

1. Al-Khayyal, F. (1990). Jointly constrained biconvex programming and related problems: An overview. *Comput. Math. Appl.* 19: 53–62.
2. Al-Khayyal, F., & Falk, J. (1983). Jointly constrained biconvex programming. *Math. Oper. Res.* 8: 273–286.
3. Audet, C., Hansen, P., Jaumard, B., & Savard, G. (2000). Branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Math. Prog.* 87(1): 131–152.
4. Bazarra, M., Sherali, H., & Shetty, C. (1993). *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons.
5. Bellido, A. (1992). Construction of iteration functions for the simultaneous computation of the solutions of equations and algebraic systems. *Numer. Algorithms* 6(3–4): 317–351.
6. Benhamou, F., McAllester, D., & Van-Hentenryck P. (1994). CLP (intervals) revisited. In *Proceedings of the International Symposium on Logic Programming*, pages 124–138.
7. COCONUT. (2002). A benchmark for global optimization and constraint satisfaction. Technical report, WWW-document (2002). http://www.mat.univie.ac.at/neum/glopt/coconut/benchmark.html.
8. Cox, D., Little, J., & O'Shea, D. (1997). *Ideals, Varieties, and Algorithms*. 2nd edition. New York: Springer-Verlag.
9. Davis, E. (1987). Constraint propagation with interval labels. *J. Artif. Intell.* 32: 281–331.
10. Didrit, O. (1997). Analyse par intervalles pour l'automatique: Résolution globale et garantie de problèmes non linéaires en robotique et en commande robuste. Ph.D. thesis, Université Parix XI Orsay.
11. Faltings, B. (1994). Arc consistency for continuous variables. *J. Artif. Intell.* 60(2): 363–376.
12. Granvilliers, L. (2004). RealPaver, Version 0.4, http://www.sciences.univernantes.fr/info/perso/permanents/granvil/realpaver/main.html.
13. Hansen, E. R. (1992). *Global Optimization Using Interval Analysis*. New York: Marcel Dekker.
14. Collavizza, H., Delobel, F., & Rueher, M. (1999). Comparing partial consistencies. *Reliab. Comput.* 5(3): 213–228.
15. IEEE-754. (1985). IEEE Standard for Binary Floating Point Arithmetic. ANSI/IEEE, New York, Std 754-1985 edition.
16. Ilog, ed. (2000a). *ILOG CPLEX 7.0, Reference Manual*. Ilog.
17. Ilog, ed. (2000b). *ILOG Solver 5.0, Reference Manual*. Ilog.
18. Jermann, C., Trombettoni, G., Neveu, B., & Rueher, M. (2000). A constraint programming approach for solving rigid geometric systems. In *Proceedings of CP'00: Sixth International Conference on* "*Principles and Practice of Constraint Programming*". Singapore, pages 233–248.
19. Kearfott, R. B. (1996). *Rigorous Global Search: Continous Problems*. Kluwer Academic Publishers Group.
20. Lebbah, Y. (1999). Contibution à la résolution de contraintes par consistance forte. Ph.D. thesis, Ecole des Mines de Nantes, France.
21. Lebbah, Y., & Lhomme, O. (2002). Accelerating filtering techniques for numeric CSPs'. *Artif. Intill.* 139(1): 109–132.

22. Lebba, Y., Michel, C., Rueher, M., Daney, D., & Merlet, J. (2005). Efficient and safe global constraints for handling numerical constraint system. *SIAM journal of Numerical Analysis* p. to appear.
23. Lebbah, Y., Rueher, M., & Michel, C. (2002). A global filtering algorithm for handling systems of quadratic equations and inequations. *Proc. CP'2002, Lect. Notes Comput. Sci.* 2470: 109–123.
24. Lhomme, O. (1993). Consistency techniques for numeric CSPs'. In *Proceedings of IJCAI'93*, pages 232–238.
25. Marti, P., & Rueher, M. (1995). A distributed cooperating constraints solving system. *Int. J. Artif. Intell. Tools* 4(1–2): 93–113.
26. McCormick, G. (1976). Computability of global solutions to factorable non-convex programs—Part I—Convex underestimating problems. *Math. Prog.* 10: 147–175.
27. Michel, C., Lebbah, Y., & Rueher, M. (2003). Safe embedding of the Simplex Algorithm in a CSP framework. In *Proceedings of 5th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems CPAIOR 2003, CRT, Université de Montréal*, pages 210–220.
28. Neumaier, A., & Shcherbina, O. (2004). Safe bounds in linear and mixed-integer programming. *Math. Prog A*. 99: 283–296.
29. Puget, J., & Van-Hentenryck, P. (1998). A constraints satisfaction approach to a circuit design problem. *J. Glob. Optim.* 13(1): 75–93.
30. Sherali, H., & Adams, W. (1999). *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishing.
31. Sherali, H., & Tuncbilek, C. (1992). A global oprtimization algorithm for polynomial using a reformulation-linearization technique. *J. Glob. Optim.* 7: 1–31.
32. Van-Hentenryck, P., Michel, L., & Deville, Y. (1997). *Numerica: a Modelling Language for Global Optiization*. MIT Press.
33. Wunderlings, R. (1996). Paralleler und Objektorientierter Simplex-Algorithmus (in German). Ph.D. thesis, Berlin.