# A Global Filtering Algorithm for Handling Systems of Quadratic Equations and Inequations

Yahia Lebbah[2], Michel Rueher[1], and Claude Michel[1]

[1] Université de Nice–Sophia Antipolis, I3S–CNRS
930 route des Colles, B.P. 145, 06903 Sophia Antipolis Cedex, France
{cpjm,rueher}@essi.fr
[2] Université d'Oran, Département d'Informatique
31000 Oran, Algeria
ylebbah@yahoo.fr

**Abstract.** This paper 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 unable to achieve a significant pruning of the domains of the variables occurring in quadratic constraints 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. First experimentations show that this new algorithm yields a much more effective pruning of the domains than local consistency filtering algorithms.

## 1 Introduction

This paper introduces a new filtering algorithm for handling systems of quadratic equations and inequations over the reals[1]. Such *quadratic continuous constraints* can be formulated as follows :

$$\sum_{(i,j)\in M} C^k_{i,j} x_i * x_j + \sum_{i\in N} C^k_i x_i^2 + \sum_{i\in N} d^k_i x_i = b_k \tag{1}$$

where $C^k_{i,j}, C^k_i, d^k_i \in I\!R$ for all $(i,j) \in M$ and $k \in 1..K$; $M$ and $N$ being sets of indices.

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 non linear constraint systems.

[1] For sake of simplicity, we will only consider equations in the rest of this paper; handling of inequations is straightforward in our framework since it is based on the simplex algorithm.

Classical filtering algorithms are based upon *local* consistencies such as 2B–consistency [16] or Box–consistency [7]), and thus, are unable to achieve a significant pruning of the domains of the variables occurring in quadratic constraints systems. The drawback of these approaches comes from the fact that the constraints are handled independently.

$3B$–consistency and $kB$–consistency are partial consistencies which can achieve a better pruning since they are "less local" [12]. However, they require numerous splitting steps to find the solutions of a system of quadratic constraints; so, they may become rather slow.

We introduce here a *global* filtering algorithm that works on a tight linear relaxation of the quadratic constraints. This relaxation is adapted from a classical linearization method, the "Reformulation-Linearization Technique (RLT)" [22, 21]. The simplex algorithm is then used to narrow the domain of each variable with respect to the subset of the linear set of constraints generated by the relaxation process. The coefficient 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.

First experimentations 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 Numerica [24] on the classical *Gough-Stewart platform*[11] benchmark.

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

### 1.1 An Illustrative Example

Consider the constraint system $\mathcal{C} = \{2x * y + y = 1, x * y = 0.2\}$ which represent two intersecting curves (see figure 1).
Suppose that $D_x = Dy = [-10, +10]$. Interval $[\underline{x}, \overline{x}]$ denotes the set of reals $S = \{r : \underline{x} \leq r \wedge r \leq \overline{x}\}$.

The reformulation-linearization technique (see section 3) yields the following constraints system:

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

where $xy$ is a new variable that stands for the product $x * y$.
Substituting $\underline{x}, \underline{y}, \overline{x}$ and $\overline{y}$ by their values and minimizing (resp. maximizing) of $x$, $y$ and $xy$ with the simplex algorithm yields the following new bounds:
$D_x = [-9.38, 9.42], D_y = [0.6, 0.6], D_{xy} = [0.2, 0.2]$.
By substituting the new bounds of $x$, $y$ and $xy$ in the constraint system $(a)$, we

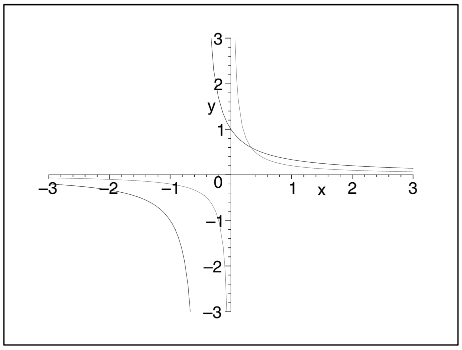

**Fig. 1.** Geometrical representation of $\{2xy + y = 1, xy = 0.2\}$

obtain the following linear constraint system :

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

Two more minimizing (resp. maximizing) steps of $x$, $y$ and $xy$ are required to obtain the bounds displayed in figure 2. Note that several splitting operations are required to find the unique solution of the problem with a 3B-consistency filtering algorithm. The proposed algorithm solves the problem by generating 5 linear constraints and with 18 calls to the simplex algorithm. It finds the same solution than a solver based on 3B–consistency but without splitting and in less time (especially on more complex problem, see section 5).

### 1.2 Outline of the Paper

Section 2 introduces the notations and recalls the basics on local consistencies that are needed in the rest of the paper. Section 3 gives an overview of our frame-

| | $2B$ | $Box$ | $3B$ | $quad$ |
|---|---|---|---|---|
| $x \in$ | $[0, 10]$ | $[0, 10]$ | $[0.333372, 0.333379]$ | $[0.333333, 0.333333]$ |
| $y \in$ | $[-10, 10]$ | $[-10, 10]$ | $[0.599917, 0.599930]$ | $[0.599999, 0.599999]$ |
| $splits$ | – | – | 2 | 0 |
| $time(sec)$ | 2 | 1 | 3 | 1 |

**Fig. 2.** Filtering of $\{2xy + y = 1, xy = 0.2\}$

work and introduces different relaxation classes. Section 4 details the filtering process while section 5 provides some experimental results.

## 2 Preliminaries

### 2.1 Notations

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

$D_x$ denotes the domain of variable $x$, that's to say, the interval $[\underline{x}, \overline{x}]$ of allowed values for $x$. $\mathcal{D}$ stands for the set of domains of all the variables of the considered constraint system.

We also use the "reformulation-linearization technique" notations introduced in [22, 4] with slight modifications.

### 2.2 Projection Functions

The algorithms used over numeric CSPs typically work by narrowing domains and need to compute the projection $\Pi_{C_j,x_i}(\mathcal{D})$, or also $\Pi_{j,i}(\mathcal{D})$, of a constraint $C_j(x_1, \ldots, x_n)$ over each variable $x_i$ in the space delimited by $D_1 \times \ldots \times D_n$.

$$\begin{aligned}\Pi_{j,i}(\mathcal{D}) = \{d_i | d_i \in D_i, \exists d_{j_1}, \ldots, d_{i-1}, d_{i+1}, \ldots, d_{j_k}\\ (d_{j_1} \in D_{j_1}, \ldots, d_{i-1} \in D_{i-1}, d_{i+1} \in D_{i+1}, \ldots, d_{j_k} \in D_{j_k},\\ \langle d_{j_1}, \ldots, d_i, \ldots, d_{j_k} \rangle \in C_j)\}.\end{aligned} \tag{2}$$

Such a projection cannot be computed exactly due to several reasons : (1) the machine numbers are floating point numbers and not real numbers so round-off errors occur; (2) the projection may not be representable as floating point numbers; (3) the computations needed to have a close approximation of the projection of only one given constraint may be very expensive; (4) the projection may be discontinuous whereas it is much more easy to handle only closed intervals for the domains of the variables.

Thus, what is usually done is that the projection of the constraint $C_j$ over the variable $x_i$ is approximated. Let $\pi_{C_j,x_i}(\mathcal{D})$ denote such an approximation. All that is needed is that $\pi_{C_j,x_i}(\mathcal{D})$ includes the exact projection; this is possible thanks to interval analysis [17, 19].

$$\pi_{j,i}(\mathcal{D}) \supseteq [min(\Pi_{j,i}(\mathcal{D})), max(\Pi_{j,i}(\mathcal{D}))] \supseteq \mathcal{D}.$$

$\pi_{C_j,x_i}(\mathcal{D})$ hides all the problems seen above. In particular, it allows us not to go into the details of the relationships between floating point and real numbers (see for example [3] for those relationships) and to consider only real numbers.

For example, most of the numeric CSPs systems (e.g., BNR-prolog [20], CLP(BNR) [8], PrologIV [9], UniCalc [5], Ilog Solver [13] and Numerica [24]) compute an approximation of the projection functions. They are based on the two most popular local consistencies: $2B$-consistency [16], *Box*-consistency [7] and the associated higher consistencies[2].

### 2.3 Limits of Local Consistencies

Formal definitions of $2B$-consistency and *Box*-consistency can be found in [12]. We will just recall here the basic idea of these local consistencies.

$2B$-consistency [16] states a local property on the bounds of the domains of a variable at a single constraint level. Roughly speaking, a constraint $c$ is $2B$-consistent if, for any variable $x$, there exist values in the domains of all other variables which satisfy $c$ when $x$ is fixed to $\underline{x}$ and $\overline{x}$.

*Box*-consistency [7] is a coarser relaxation of *Arc*-consistency than $2B$-consistency. It mainly consists of replacing every existentially quantified variable but one with its interval in the definition of $2B$-consistency.

Different approximations are introduced in the implementations of the corresponding filtering algorithms:

– $2B$-filtering decomposes the initial constraints in ternary basic constraints for which it is trivial to compute the projection [10, 16].
– *Box*-filtering generates a system of univariate interval functions which can be tackled by numerical methods such as Newton. Contrary to $2B$-filtering, *Box*-filtering does not require any constraint decomposition of the initial constrain systems.

The success of $2B$-consistency depends on the precision of the projection function $\pi_{j,i}$. In this paper, 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 $g(x_i, x_j) = x_i * x_j$. $f$ 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.

## 3 Quadratic Constraint Filtering Based on Linear Programming

The problem of quadratic terms linearization has been studied in quadratic optimization; for a deeper overview see [1, 21]. We introduce here a simplification of these linearizations adapted to our purpose.

The proposed approach is based on Reformulation-Linearization Technique (RLT) [21] for filtering efficiently a quadratic constraints systems:

[2] In the same way that *arc*-consistency has been generalized to higher consistencies (e.g. path-consistency), $2B$–consistency can be generalized to $3B$-consistency and *Box*–consistency can be generalized to Bound-consistency [12].

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

where $C_{i,j}^k, C_i^k, d_i^k \in I\!R$ for all $(i,j) \in M$ et $k \in 1..K$.

## 3.1 An Overview of the Proposed Framework

The goal is to transform the quadratic constraints 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 : $y$ for $x^2$ with a domain $[\underline{x}, \overline{x}]$; $y_{i,j}$ for $x_i * x_j$ with a domain $[\min\{\underline{x_i} * \underline{x_j}, \underline{x_i} * \overline{x_j}, \overline{x_i} * \underline{x_j}, \overline{x_i} * \overline{x_j}\}, \max\{\underline{x_i} * \underline{x_j}, \underline{x_i} * \overline{x_j}, \overline{x_i} * \underline{x_j}, \overline{x_i} * \overline{x_j}\}]$
– linearize each quadratic constraint of the constraint system (1) by using the previous new variables; we denote the produced system

$$[\sum_{(i,j)\in M} C_{i,j}^k x_i * x_j + \sum_{i\in N} C_i^k x_i^2 + \sum_{i\in N} d_i^k x_i = b_k]_l$$

where $(i,j) \in M$ and $k \in 1..K$; $[E]_l$ denotes $E$ where the quadratic terms are replaced by their variables.

A tight linear (convex) relaxation, or outer-approximation to the convex and concave envelope[3] of the quadratic terms over the constrained region, is built by generating new linear inequalities.

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 take into account quadratic terms. In the next subsections we introduce two tight linear relaxation classes that preserves equations $y = x^2$ and $y_{i,j} = x_i * x_j$ and that provide a better approximation than interval arithmetic. In section 4, we give an overview of the whole filtering algorithm of quadratic constraints.

[3] "Let $f : S \rightarrow E_1$, 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 $S$" [6],page 125.

### 3.2 Linearization of $x^2$

Proposition 1 introduces the simplest and straightforward linearizations.

**Proposition 1 (class I: linearizations for $f(x) = x^2$).**
*Consider the function $f(x) = x^2$ and suppose that $\underline{x} \leq x \leq \overline{x}$, then the following relations preserve all values $(x, y)$ satisfying $y = f(x)$:*

$$L1(y, \alpha) \equiv [(x - \alpha)^2 \geq 0]_l \text{ where } \alpha \in [\underline{x}, \overline{x}] \tag{3}$$

*and*

$$L2(y) \equiv (\underline{x} + \overline{x})x - y - \underline{x} * \overline{x} \geq 0 \tag{4}$$

*Proof :* Inequality (3) is straightforward.
The second inequality (4), stated by [2], comes from the valid inequality :

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

□

Note that $[(x - \alpha_i)^2 = 0]_l$ generates the tangent line to the curve $y = x^2$ at the point $x = \alpha_i$. Consider for instance the quadratic term $x^2$ with $x \in [-4, 5]$. Figure 3 displays the initial curve (i.e., $D_1$), and the lines corresponding to the equations generated by the relaxations: $D_2$ for $L1(y, -4) \equiv y + 8x + 16 \geq 0$, $D_3$ for $L1(y, 5) \equiv y - 10x + 25 \geq 0$ , and $D_4$ for $L2(y) \equiv -y + x + 20 \geq 0$. We may note that $L1(y, -4)$ and $L1(y, 5)$ are an underestimations of $y$ whereas $L2(y)$ is an overestimation.

The following proposition shows that Class I relaxations respect the interval square as defined in interval arithmetic [17].

**Proposition 2 (interval square).**
*Consider the function $f(x) = x^2$ and suppose that $\underline{x} \leq x \leq \overline{x}$, then it results from relations $L1(y, \underline{x})$, $L1(y, \overline{x})$ and $L2(y)$ that*

$$y \in [min(\underline{x}^2, \overline{x}^2), max(\underline{x}^2, \overline{x}^2)] \text{ if } 0 \notin [\underline{x}, \overline{x}]$$

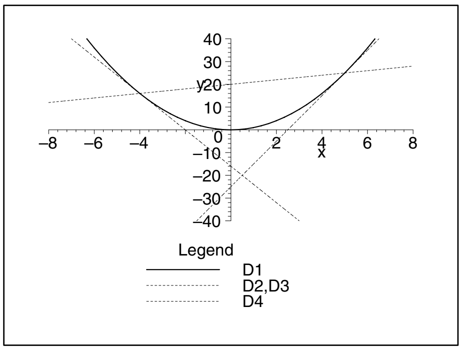

**Fig. 3.** Illustration of class I relaxations

*and*

$$y \in [0, max(\underline{x}^2, \overline{x}^2)] \textit{ otherwise}$$

*Proof :* Relations $L1(y, \underline{x})$, $L1(y, \overline{x})$ and $L2(y)$ state that

$$y - 2\underline{x} * x + \underline{x}^2 \geq 0 \Rightarrow y \geq 2\underline{x} * x - \underline{x}^2$$

$$y - 2\overline{x} * x + \overline{x}^2 \geq 0 \Rightarrow y \geq 2\overline{x} * x - \overline{x}^2$$

$$(\underline{x} + \overline{x})x - y - \underline{x} * y \geq 0 \Rightarrow y \leq (\underline{x} + \overline{x})x - \underline{x} * y$$

Suppose that $x \geq \underline{x} \geq 0$, then the first inequation entails $y \geq \underline{x}^2$ and the second one implies $y \leq \overline{x}^2$. The other cases can be proven in a similar way. □

## 3.3 Linearization of $x_i * x_j$

Now, consider the function $g(x_i, x_j) = x_i x_j$ over $[\underline{x_i}, \overline{x_i}] \times [\underline{x_j}, \overline{x_j}]$. Proposition 3 has been stated by [2, 1]. It introduces a linear outer approximations of the function $g$ on the hyper-rectangle $[\underline{x_i}, \overline{x_i}] \times [\underline{x_j}, \overline{x_j}]$.

**Proposition 3 (Class II : linearization of $g(x_i, x_j) = x_i x_j$).**
*Consider* $g(x_i, x_j) = x_i x_j$ *with* $\underline{x_i} \leq x_i \leq \overline{x_i}$ *and* $\underline{x_j} \leq x_j \leq \overline{x_j}$*, then for all* $x_i \in [\underline{x_i}, \overline{x_i}]$ *and* $x_j \in [\underline{x_j}, \overline{x_j}]$*, the following relations preserve all values* $(x_i, x_j, y_{i,j})$ *satisfying* $y_{i,j} = g(x_i, x_j)$*:*

$$L3(y_{i,j}) \equiv [(x_i - \underline{x_i})(x_j - \underline{x_j}) \geq 0]_l \tag{5}$$

$$L4(y_{i,j}) \equiv [(x_i - \underline{x_i})(\overline{x_j} - x_j) \geq 0]_l \tag{6}$$

$$L5(y_{i,j}) \equiv [(\overline{x_i} - x_i)(x_j - \underline{x_j}) \geq 0]_l \tag{7}$$

$$L6(y_{i,j}) \equiv [(\overline{x_i} - x_i)(\overline{x_j} - x_j) \geq 0]_l \tag{8}$$

*Proof :* Relations $L3(y_{i,j})...L6(y_{i,j})$ are of the form $AB \geq 0$, where $A$ and $B$ are always positive. □

These relations define respectively the concave and convex envelopes of $g(x_i, x_j)$. Consider for instance the quadratic term $x * y$ with $x \in [-5, 5]$ and $y \in [-5, 5]$. The work done by the linear relaxations of the 3D curve $z = x * y$ is well illustrated in 2D by fixing $z$. Figure 4 displays the 2D shape, for the level $z = 5$, of the initial curve (i.e., $Cu$), and the lines corresponding to the equations generated by the relaxations (where $z = 5$): $D_1$ for $L3(z) \equiv z + 5x + 5y + 25 \geq 0$, $D_2$ for $L4(z) \equiv -z + 5x - 5y + 25 \geq 0$, $D_3$ for $L5(z) \equiv -z - 5x + 5y + 25 \geq 0$, and $D_4$ for $L6(z) \equiv z - 5x - 5y + 25 \geq 0$. We may note that these relaxations are the optimal linear relaxations of $z = x * y$.

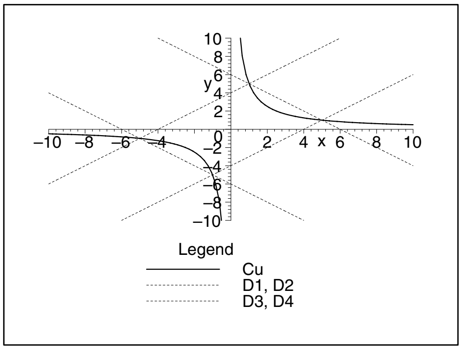

**Fig. 4.** Illustration of class II relaxations

**Proposition 4 (Convex/concave envelopes of $g$ [2, 1]).** *$L3(y_{i,j})$ and $L6(y_{i,j})$ are the convex envelope of $g$. $L4(y_{i,j})$ and $L5(y_{i,j})$ are the concave envelope of $g$.*

*Proof :* See [2, 1]. □

Thus with Class II, we have the tightest linear relaxations of the function $g$.

The following proposition shows that Class II relaxations respect interval multiplication as defined in interval arithmetic [17].

**Proposition 5 (interval multiplication).**
*Consider $g(x_i, x_j) = x_i * x_j$ with $\underline{x_i} \leq x_i \leq \overline{x_i}$ and $\underline{x_j} \leq x_j \leq \overline{x_j}$, then for all $x_i \in [\underline{x_i}, \overline{x_i}]$ and $x_j \in [\underline{x_j}, \overline{x_j}]$. It results from relations $L3(y_{i,j})$, $L4(y_{i,j})$, $L5(y_{i,j})$ and $L6(y_{i,j})$ that*

$$y_{i,j} \in [min(\underline{x_i} * \underline{x_j}, \underline{x_i} * \overline{x_j}, \overline{x_i} * \underline{x_j}, \overline{x_i} * \overline{x_j}), max(\underline{x_i} * \underline{x_j}, \underline{x_i} * \overline{x_j}, \overline{x_i} * \underline{x_j}, \overline{x_i} * \overline{x_j})]$$

*Proof :* According to the sign of $\underline{x_i}, \overline{x_i}, \underline{x_j}$ and $\overline{x_j}$ : $L3(y_{i,j})$ or $L6(y_{i,j})$ yields the interval multiplication lower bound whereas $L4(y_{i,j})$ or $L5(y_{i,j})$ provides the interval multiplication upper bound. □

Many other relaxations have been proposed by [21, 4]. For simplicity, we prefer to restrict the presentation to only these two classes which are sufficient to achieve a better filtering than $2B$-consistency and $Box$-consistency do.

## 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:

$$LRQ \equiv \begin{cases} [\sum_{(i,j)\in M} C_{i,j}^k x_i * x_j + \sum_{i\in N} C_i^k x_i^2 + \sum_{i \in N} d_i^k x_i = b_k]_l \\ L1(y, \underline{x_i}), L1(y, \overline{x_i}) \text{ for all } i \in N \\ L2(y) \text{ for all } i \in N \\ L3(y_{i,j}), L4(y_{i,j}), L5(y_{i,j}), L6(y_{i,j}) \text{ for all } (i,j) \in M \end{cases}$$

*LRQ* will be embedded in a linear program (LP) for filtering *quad*-constraints.

The filtering process is shown in Algorithm 1.

**Algorithm 1** The `Quad` algorithm

```
Function $quad-filtering$(IN: $\mathcal{D}$, $Q$, $\epsilon$) return $\mathcal{D}'$
% $\mathcal{D}$: input domains; $Q$: quadratic constraints (1)
% $\epsilon$: minimal reduction
$\mathcal{D}' := \mathcal{D}$
do
    $\mathcal{D} := \mathcal{D}'$
    for all $x_i \in vars(Q)$ do
        Construct and solve the following LP
            $\begin{cases} Z = minimize(x_i) \\ LRQ \text{ from the quadratic constraints } Q \end{cases}$
        $\underline{x_i}' := max(\underline{x_i}, Z)$
        Solve the same LP with $Z = maximize(x_i)$
        $\overline{x_i}' := min(\overline{x_i}, Z)$
    endfor
while  the reduction amount of some bound is greater than $\epsilon$ and $\emptyset \notin \mathcal{D}'$
```

Since *LRQ* is a safe approximation of the initial constraint system, `Quad` achieves a safe pruning of the domains[4].

At each iteration step, the reduction achieved by the simplex must be greater than $\epsilon$ 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 terminated if $\epsilon$ is greater than zero.

Next section shows that this algorithm behaves well on different benchmarks.

## 5 Experimentations

To evaluate the contribution of the `Quad` algorithm we have compared its performances with classical filtering algorithms (i.e., $2B$-filtering, *Box*-filtering and $3B$-filtering) and with the `Numerica` system [24] on two benchmarks : the Gough-Stewart platform [11] and a Kinematics application, named "kin2" [18, 23]. We have performed the following experimentations:

– Filtering initial domains containing exactly one solution;

[4] Provided that the computation done by the simplex algorithm are corrects, that's to say that the rounding problem are properly handled in the implementation of the simplex algorithm; actually a safe rounding can also be achieved after calling an LP-solver (see 'Safe bounds in linear and mixed-integer programming' http://www.mat.univie.ac.at/ neum/papers.html#mip)

– Filtering initial domains containing many solutions;
– Combining of filtering and splitting to isolate all the solutions.

Experimentations concerning $2B$-filtering, *Box*-filtering and $3B$-filtering have been performed with the implementation of `iCOs` [14, 15], one of the most efficient library for this kind of algorithms.

`Quad` has been implemented with the linear programming solver "SOPLEX" [25]. `Quad` and `Numerica` use the same splitting strategies to isolate the different solutions. Experimentations with `Numerica` have been run on a Sun Enterprise 4000 with two Ultrasparc II at 336Mhz whereas all other experimentations have been done on PC-Notebook/1Ghz. CPU times are given in seconds.

## 5.1 The Gough-Stewart Platform Benchmark

The first benchmark [11] comes from robotics and describes the kinematics of a Gough-Stewart platform.

*Problem 1 (Gough-Stewart [11]).*

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

Table 1 shows the results of the different filtering algorithms for some initial 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 2 shows that the different algorithms yield almost the same pruned domains for some initial domains containing several solutions.

Table 3 provides the execution timing for finding all the solutions on the domains of 2. `Quad` outperforms `Numerica` and "interval Newton" on this problem. `Numerica` was run with *Box*–consistency, the default consistency; results where worst with *Bound*–consistency.

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

The results of "interval Newton" are those published by [11]; the computations where done on a Pentium 90 computer, more than 10 time slower than our computer. Removing multiple occurrences of the variables, enables [11] to solve the problem in about three hours. Didrit [11] solves in 24 minutes an other formulation of that problem which is far from being obvious and where three

**Table 1.** Filtering results on the Gough-Stewart problem for some domain containing only one solution

| | Initial domains | $2B$ | $Box$ | $3B$ | `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 2.** Filtering results on the Gough-Stewart problem for some domain containing many solutions

| | Initial domains | $2B$ | $Box$ | $3B$ | `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] |

variables are removed. Note that `Quad` requires very few splittings to find the four solutions of this problem.

## 5.2 The Kinematics Benchmark

Now, let us consider a kinematics application, named “ kin2” [18, 23]. This second benchmark describes the inverse position for a six-revolute-joint problem. The results obtained with the different algorithms are similar to the previous one.

Table 4 shows the results of the different filtering algorithms for some initial domains containing only one solution. `Quad` is again the only algorithm who isolates the unique solution.

Once again, the pruning achieved by $2B/Box/3B$-filtering and `Quad` is not significant when the initial domains contain several solutions (see table 5).

Table 6 shows that the performances of `Quad` and `Numerica` are comparable but that `Quad` requires much less splittings than `Numerica` to find the solutions. The speed of `Quad` could probably be improved by using a more efficient simplex algorithm like CPLEX .

**Table 3.** Filtering performance for Gough-Stewart problem

| | Interval Newton | Numerica | `Quad` |
|---|---|---|---|
| CPU time | 14400 | 4480 | 183 |
| splittings | ? | 122469 | 24 |
| narrowings | $10^6$ | 10470159 | 27255 |

**Table 4.** Filtering results on the kinematics problem for some domain containing a single solution

| | Initial domains | $2B$ | $Box$ | $3B$ | `Quad` |
|---|---|---|---|---|---|
| $x_1$ | [ 0.00, 1.00] | - | - | - | [ 0.97, 0.97] |
| $x_2$ | [ 0.00, 1.00] | - | - | - | [ 0.20, 0.20] |
| $x_3$ | [ 0.00, 1.00] | - | - | [ 0.00, 0.95] | [ 0.02, 0.02] |
| $x_4$ | [ 0.00, 1.00] | - | - | [ 0.29, 1.00] | [ 0.99, 0.99] |
| $x_5$ | [-1.00, 0.00] | - | - | - | [-0.09,-0.09] |
| $x_6$ | [ 0.00, 1.00] | - | - | - | [ 0.99, 0.99] |
| $x_7$ | [ 0.00,1.00] | - | - | [ 0.00, 0.96] | [ 0.07, 0.07] |
| $x_8$ | [-1.00, 0.00] | - | - | [-1.00,-0.27] | [-0.99,-0.99] |

**Table 5.** Filtering results on the kinematics problem for some domain containing many solutions

| | Initial domains | $2B$ | $Box$ | $3B$ | `Quad` |
|---|---|---|---|---|---|
| $x_1$ | [-1.00, 1.00] | - | - | - | - |
| $x_2$ | [-1.00, 1.00] | - | - | - | - |
| $x_3$ | [-1.00, 1.00] | - | - | - | - |
| $x_4$ | [-1.00, 1.00] | - | - | - | - |
| $x_5$ | [-1.00, 1.00] | - | - | - | - |
| $x_6$ | [-1.00, 1.00] | - | - | - | - |
| $x_7$ | [-1.00, 1.00] | - | - | - | [-0.99, 1.00] |
| $x_8$ | [-1.00, 1.00] | - | - | - | [-1.00, 0.98] |

## 6 Conclusion

This paper 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. First experimentations are very promising and show the capabilities of this framework.

Further works concern the integration of `Quad` in a general interval solver to tackle non-polynomial problems with a significant subset of quadratic constraints. So, `Quad` could play the role of global constraint in many geometric or robotics applications where numerous distance constraints often occur.

**Table 6.** Filtering performance for the kinematics problem

| | Numerica | *quad* |
|---|---|---|
| CPU time | 169 | 61 |
| splittings | 5421 | 33 |
| narrowings | 105704 | 12267 |

## Acknowledgments

Thanks to Prof. Arnold Neumaier who suggested us to explore the capabilities of linearization techniques for a global handling of distance relations. Thanks also to Bertrand Neveu for his careful reading of an earlier draft of this paper.

## References

[1] F. A. Al-Khayyal. Jointly constrained biconvex programming and related problems: An overview. *Computers and Mathematics with Applications*, pages Vol.19, No.11, 53–62, 1990. 113, 116, 117

[2] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages Vol.8, No.2, 273–286, 1983. 115, 116, 117

[3] G. Alefeld and J. Hezberger, editors. *Introduction to Interval Computations*. Academic press, 1983. 112

[4] 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. 112, 117

[5] O. P. Babichev, A. B.and 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. 112

[6] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming : Theory and Algorithms*. John Wiley & Sons, 1993. 114

[7] F. Benhamou, D. McAllester, and P. Van-Hentenryck. Clp(intervals) revisited. In *Proceedings of the International Symposium on Logic Programming*, pages 124–138, 1994. 110, 113

[8] F. Benhamou and W. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, pages 32(1):1–24, 1997. 112

[9] A. Colmerauer. Spécifications de prolog iv. Technical report, GIA, Faculté des Sciences de Luminy,163, Avenue de Luminy 13288 Marseille cedex 9 (France), 1994. 112

[10] E. Davis. Constraint propagation with interval labels. *Journal of Artificial Intelligence*, pages 32:281–331, 1987. 113

[11] O. Didrit. *Analyse par intervalles pour l'automatique : résolution globale et garantie de problèmes non linéaires en robotique et en commande robuste*. PhD thesis, Université Parix XI Orsay, 1997. 110, 118, 119

[12] M. Rueher H.Collavizza, F.Delobel. Comparing partial consistencies. *Reliable Computing*, pages Vol.5(3),213–228, 1999. 110, 113

[13] Ilog, editor. *ILOG Solver 4.0, Reference Manual.* Ilog, 1997. 112

[14] Y. Lebbah. *Contribution à la résolution de contraintes par consistance forte.* PhD thesis, Ecole des Mines de Nantes, France, 1999. 119

[15] Y. Lebbah and O. Lhomme. Accelerating filtering techniques for numeric csps. *Journal of Artificial Intelligence*, page Forthcoming, 2002. 119

[16] O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of IJCAI'93*, pages 232–238, 1993. 110, 113

[17] R. Moore. *Interval Analysis.* Prentice Hall, 1966. 112, 115, 117

[18] A. P. Morgan. Computing all solutions to polynomial systems using homotopy continuation. *Appl. Math. Comput.*, pages 24:115–138, 1987. 118, 120

[19] A. Neumaier. *Introduction to Numerical Analysis.* Cambridge Univ. Press, Cambridge, 2001. 112

[20] 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. 112

[21] H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishing, 1999. 110, 113, 117

[22] H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial using a reformulation-linearization technique. *Journal of Global Optimization*, pages 7, 1–31, 1992. 110, 112

[23] 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. 118, 120

[24] P. Van-Hentenryck, L. Michel, and Y. Deville. *Numerica : a Modeling Languge for Global Optimization.* MIT press, 1997. 110, 112, 118

[25] R. Wunderlings. *Paralleler und Objektorientierter Simplex-Algorithmus (in German).* PhD thesis, Berlin, 1996. 119