Combining local consistencies with a new global filtering algorithm on linear relaxations

Yahia LEBBAH[2], Claude MICHEL[1], Michel RUEHER[1]

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

Content Areas: constraint satisfaction, robotics, application

Abstract. Numeric systems of constraints are widely used to model problems in numerous application areas ranging from robotics to chemistry. This paper introduces a new filtering algorithm (GFLR) to prune the domains of the variables in such numeric applications. Roughly speaking, GFLR combines classical local consistencies and a new global filtering algorithm that works on a linear relaxation of numeric constraints. We introduce a safe rounding procedure to preserve the completeness of the linear constraint filtering algorithm. Different combinations of these filtering techniques have been investigated and the most significant results are reported. Experimental results on difficult robotic problems show that a solver based on the GFLR filtering algorithm outperforms classical CSP approaches.

1 Introduction

We have recently introduced in [11] a global filtering algorithm (named Quad) for handling systems of quadratic equations and inequations over the real numbers. Such *quadratic continuous constraints*[1] can be formulated as follows :

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

where $C_{i,j}^k, C_i^k, d_i^k \in \mathbb{R}$ for all $(i,j) \in M$ and $k \in 1..K$; M and N being sets of indices.

The drawback of classical approaches (e.g. 2B–consistency [13] or Box–consistency [4]) on equations (1) comes from the fact that the constraints are handled independently and in a blind way; that's to say, these approaches do not take advantage of the very specific semantic of quadratic constraints to reduce the domains of the variables.
do not exploit efficiently quadratic terms semantics. $3B$–consistency and kB–consistency are partial consistencies which can achieve a better pruning since they are "less local" [9]. However, they require numerous splitting steps to find the solutions of a system of quadratic constraints; so, they may become rather slow.

[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.

The `Quad` *global* filtering algorithm [11] works on a tight linear relaxation of the quadratic constraints. This relaxation is adapted from a classical linearization method, the "Reformulation-Linearization Technique (RLT)" [18, 17]. 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. 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. We have demonstrated [11] that the `Quad` algorithm yields a more effective pruning of the domains than local consistency filtering algorithms.

In this paper, we generalize and extend our work in the following way :

1. We provide a safe rounding procedure that ensures the completeness of the `Quad` filtering algorithm.
2. We extend the scope of the `Quad` filtering algorithm to general systems of non linear equations and inequations.
3. We define a new filtering algorithm (named `GFLR`) that combines the generalization of `Quad` and classic local consistencies.

Experimental results on difficult robotic problems show that a solver based on the new `GFLR` filtering technique outperforms both solvers based on `Quad` and solvers based on classic local consistencies (e.g. Box–consistency [4]).

1.1 A brief summary of our framework

a) `Quad` filtering algorithm Classical local consistencies do not exploit the semantic of quadratic terms. Linear programming techniques [2, 18, 3] allow us to capture most of the semantics of quadratic terms (e.g., convex and concave envelopes of these particular terms). Section 2.2 (and 2.3) summarizes the `Quad` algorithm that use tight linear relaxation to capture the semantics of quadratic terms.

b) An extension of `Quad` for handling non-linear constraints The extension of `Quad` to polynomial constraints requires to replace non-quadratic terms by new variables and to add the corresponding identities to the constraint system. However, a complete quadrification [20] will generate a huge number of linear constraints. We introduce here a heuristic based on a good tradeoff between a tight approximation of the non linear terms and the size of the generated constraint system. This extension of the `Quad` algorithm is denoted by `Ext_Quad` in the rest of this paper (see section 2).

Section 4 shows that this heuristic works well on classical benchmarks.

c) A safe rounding for `Quad` `Quad` is a *global* filtering algorithm that works on a tight linear relaxation of the quadratic constraints. 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.

Two problems occur in the `Quad` -filtering process:

1. The coefficients of the generated linear constraints are real numbers but when they are computed on floating point numbers the whole linearization may be incorrect due to the rounding errors;

2. Most linear programming solvers are implemented with floating point numbers, and thus, are unsafe.

We propose in this paper a safe procedure that prevents the simplex algorithm from removing any solution while filtering the generated linear constraint system.

The second problem has recently been addressed by Neumaier [16]. He proposes a simple and cheap procedure to get a rigorous upper bound of the objective function. The incorporation of this procedure in the `Quad` filtering process allows us to call the simplex algorithm without worrying about safety.

Thus, this paper shows that linear programming can be used to handle continuous CSPs *without losing any solution.*

d) `GFLR`, a new global filtering algorithm `Ext_Quad` and `Quad` are global filtering algorithms that work on a linear relaxation of the initial constraint system. Arc–consistency [14] is a famous local consistency that has been widely used in finite domains. Box–consistency is the most successful adaptation of arc–consistency to constraints over the real numbers. The `interval newton method` [8] plays a key role in numerical analysis. It computes a local consistency that is weaker than Box–consistency [2] but the algorithm is very fast. So, we propose to combine `interval newton method` filtering, Box–consistency filtering and `Ext_Quad` filtering to prune the domain of the variables of numeric constraint system. More precisely, we call `GFLR` filtering the sequential processing of:

1. `interval newton method`,
2. `Ext_Quad` filtering,
3. Box–consistency filtering,
4. `interval newton method`.

Experimental results on difficult robotic problems show that a `GFLR_`solver[3] outperforms classical approaches.

Before going into the details, let us outline the advantage of our approach on three examples.

1.2 Motivating examples

a) `Quad` -filtering 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 $S = \{r : \underline{x} \leq r \wedge r \leq \overline{x}\}$.

[2] The last implementation of Box-consistency [21] works on the Taylor interval extension of the whole constraint system and performs a conditioning step. That's why Box–consistency is stronger than the `interval newton method` .

[3] A `GFLR_`solver (resp.`Ext_Quad_`solver) is just a process that performs a search and prune strategy, that's to say as soon as the `GFLR` filtering algorithm (resp. `Ext_Quad` filtering algorithm) reaches a fixed point, a bisection of the domain of some variable is done and the process is restarted. The solver stops when all solutions are found or when the size of the domains becomes smaller than some fixed value.

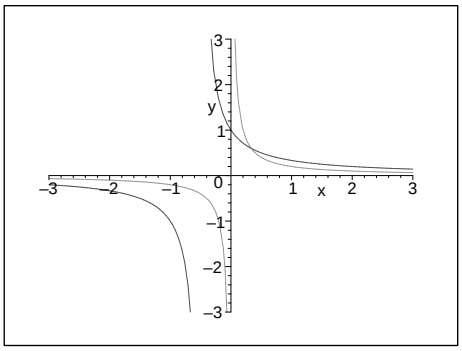

Fig. 1. Geometrical representation of $\{2xy + y = 1, xy = 0.2\}$

The reformulation-linearization technique (see section 2.2) 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 obtain a new linear constraint system. Two more minimizing (resp. maximizing) steps of x, y and xy are required to obtain the solution bounds. 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.

b) A safe rounding procedure Consider the constraint system

$$\mathcal{C} = \begin{cases} w_1 + w_2 = 1; w_1 x_1 + w_2 x_2 = 0; \\ w_1 x_1 x_1 + w_2 x_2 x_2 = 1; w_1 x_1 x_1 x_1 + w_2 x_2 x_2 x_2 = 0; \end{cases}$$

which represents the Gaussian quadrature formula to compute integrals [5]. Suppose that $D_{x_1} = D_{x_2} = D_{w_1} = D_{w_2} = [-1, +1]$. This system has two solutions :

- $x_1 = -1, x_2 = 1, w_1 = 0.5, w_2 = 0.5$,
- $x_1 = 1, x_2 = -1, w_1 = 0.5, w_2 = 0.5$.

A `Quad_`solver finds only the following unsafe solution (the value of x_2 is incorrect):

$$\begin{array}{llll} x_1 \in [-1.000000..., & -0.99999999...] & x_2 \in [+0.9999...944, & +0.9999...989] \\ w_1 \in [+0.49999999..., & +0.5000000...] & w_2 \in [+0.49999999..., & +0.5000000...] \end{array}$$

Indeed, when we examine the Quad-filtering process, we can find some LPs where the Simplex algorithm steps to the wrong side of the objective.

With the corrections we propose in section 3, we obtain a tight approximation of the two correct solutions :

$$\begin{array}{llll} x_1 \in [0.999999..., & 1.00000...1] & x_2 \in [-1.000000..., & -0.999999...] \\ w_1 \in [0.499999..., & 0.5000000...] & w_2 \in [0.499999..., & 0.5000000...] \end{array}$$

$$\begin{array}{llll} x_1 \in [-1.000000..., & -0.99...9811] & x_2 \in [0.999999..., & 1.000000...] \\ w_1 \in [0.499999..., & 0.5000000...] & w_2 \in [0.499999..., & 0.5000000...] \end{array}$$

c) Capabilities of GFLR : The benefit of combining global filtering on linear relaxations with local consistencies is illustrated by the performances on the Gough-Stewart platform problem.

A general Gough-Stewart platform consist of two rigid bodies connected by six rods attached via spherical joints. Dietmaier [7] showed that this problem has 40 effective assembly modes over the real numbers. The GFLR _solver managed to find the forty solutions of the Dietmaier benchmark in about 22 minutes whereas the Ext_Quad_solver required about 74 hours and Ilog solver needed more than 800 hours !

These results show that combining global algorithms and good propagation techniques is a key issue when solving numeric CSP.

1.3 Outline of the paper

Next subsection defines the notation. Section 2 details the linearization process and recalls the principle of the Quad algorithm. Section 3 introduces the proposed rounding process while section 4 reports the experimental results.

1.4 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. $\mathbb{R}$ denotes the set of real numbers whereas $\mathbb{F}$ stands for the set of floating point numbers.

We also use the "reformulation-linearization technique" notations introduced in [18, 3] with slight modifications.

2 Constraints filtering based on linear relaxations

The filtering process is achieved in two steps : first, we transform the non-linear terms into quadratic terms, second we use a tight linear relaxations of the generated quadratic terms, with the Quad principal, to prune the original non-linear constraints.

2.1 Transformation of non-linear constraints into quadratic constraints

In this section, we show how to transform a nonlinear constraint system into an equivalent quadratic constraint system.

Let E be some non-linear expression. $[E]_q$ denotes the quadrification of E, that's to say a new expression where the nonlinear terms have been replaced by equivalent quadratic terms. $[E]_{qI}$ denotes the set of equalities that keep the link between the new variables and the non-linear terms.

For example, consider $E = \{x_2x_3x_4^2 + 3x_6x_7 + \sin(x_1) = 0\}$, the proposed transformation yields:

- $[E]_q = \{y_1y_2 + 3y_2 + s_1 = 0, y_1 = x_2x_3, y_2 = x_4x_4, y_3 = x_6x_7\}$
- $[E]_{qI} = \{y_1 = x_2x_3, y_2 = x_4^2, y_3 = x_6x_7, s_1 = \sin(x_1)\}$.

$[E]_{qI}$ contains those non-polynomial terms that cannot be further quadrified. Such a transformation is one of the possible quadrifications. It is called a *single* quadrification.

We could generate all possible single quadrifications, or all quadrifying identities, and perform a so called *complete* quadrification. For example, the complete quadrification $[E]_Q$ of $E = \{x_2x_3x_4^2 + 3x_6x_7 + \sin(x_1) = 0\}$ is:

$$[E]_Q \equiv \begin{cases} y_1y_2 + 3y_2 + s_1 = 0, & y_1 = x_2x_3, & y_2 = x_4x_4, & y_3 = x_6x_7, \\ y_4y_5 + 3y_2 + s_1 = 0, & y_4 = x_2x_4, & y_4 = x_3x_4, & \\ x_4y_6 + 3y_2 + s_1 = 0, x_2y_7 + 3y_2 + s_1 = 0, x_3y_8 + 3y_2 + s_1 = 0, & & & \end{cases}$$
$$\cup [y_6 = x_2x_3x_4]_Q \cup [y_7 = x_3x_4^2]_Q \cup [y_8 = x_2x_4^2]_Q$$

where Equality $s_1 = \sin(x_1)$ is contained in $[E]_{QI}$

A complete quadrification for polynomial problems was introduced by Shor [20]. Sherali and Tuncbilek [19] have proposed a direct reformulation/linearization of the whole polynomial constraints without quadrifying the constraints. They did prove the dominance of their direct reformulation/linearization technique over Shor's quadrification.

Complete quadrification and direct reformulation/linearization yields a tighter linearization than the single quadrification but the number of generated linearizations grows in an exponential ways for non trivial polynomial constraint systems. Thus, we decide to generate only a single quadrification.

Since many single quadrifications exist, an essential point is the choice of a good heuristic that captures most of the semantics of the polynomial constraints. We use a "middle" heuristic to obtain balanced degrees on the generated terms. For instance, consider $T \equiv x_1x_2 \dots x_n$, a monomial of degree n, the middle heuristic will identify two monomials T_1 and T_2 of highest degree such that $T = T_1T_2$. It follows that $T_1 = x_1x_2 \dots x_{n \div 2}$ and $T_1 = x_{n \div 2+1} \dots x_n$.

Next subsection recalls the principle of the `Quad` algorithm which will be used to linearize the quadratic terms.

2.2 Linearization of quadratic terms

Quadratic constraints are approximated by linear constraints in the following way. `Quad` creates a new variable for each quadratic term : y for x^2, $y_{i,j}$ for $x_i * x_j$. The produced system is denoted:

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

; $[E]_l$ 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 of the quadratic terms over the constrained region, is built by generating new linear inequalities.

`Quad` uses 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.

Linearization of x^2 The function $f(x) = x^2$ with $\underline{x} \leq x \leq \overline{x}$ is approximated by the following relations:

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

$$L2(y) \equiv (\underline{x} + \overline{x})x - y - \underline{x} * \overline{x} \geq 0 \tag{3}$$

Note that $[(x - \alpha_i)^2 = 0]_l$ generates the tangent line to the curve $y = x^2$ at the point $x = \alpha_i$. Actually, `Quad` computes only $L1(y, \overline{x})$ and $L1(y, \underline{x})$. Consider for instance the quadratic term x^2 with $x \in [-4, 5]$. Figure 2 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 $L1(y, \overline{x})$ and $L1(y, \underline{x})$ are an underestimations of y whereas $L2(y)$ is an overestimation. $L2$ is also the concave envelope, which means that it is the optimal concave overestimation.

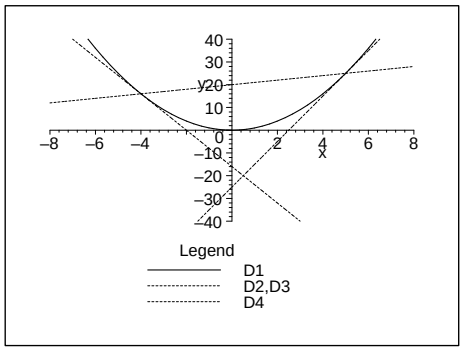

Fig. 2. Illustration of class I relaxations

Linearization of $x_i * x_j$ The function $g(x_i, x_j) = x_i x_j$ over $[\underline{x_i}, \overline{x_i}] \times [\underline{x_j}, \overline{x_j}]$ is approximated by the following relations:

$$L3(y_{i,j}) \equiv [(x_i - \underline{x_i})(x_j - \underline{x_j}) \geq 0]_l \tag{4}$$

$$L4(y_{i,j}) \equiv [(x_i - \underline{x_i})(\overline{x_j} - x_j) \geq 0]_l \tag{5}$$

$$L5(y_{i,j}) \equiv [(\overline{x_i} - x_i)(x_j - \underline{x_j}) \geq 0]_l \tag{6}$$

$$L6(y_{i,j}) \equiv [(\overline{x_i} - x_i)(\overline{x_j} - x_j) \geq 0]_l \tag{7}$$

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 3 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 $L6(z) \equiv z - 5x - 5y + 25 \geq 0$.

$L3(y_{i,j})$ 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 (see [2, 1]). Consequently these relaxations are the optimal convex/concave outer-estimations of $z = x * y$.

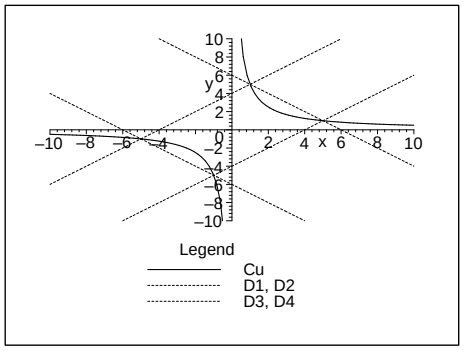

Fig. 3. Illustration of class II relaxations

2.3 `Ext_Quad` , the new filtering algorithm

We are now in position to introduce the whole filtering procedure we propose for handling general constraints. The `Ext_Quad` filtering algorithm (see Algorithm 1) starts with a quadrification step on the whole nonlinear constraints, and after that, applies the `Quad` filtering on the generated quadratic constraints. `Quad` algorithm [11] consists of three main steps : reformulation, liearization and filtering.

The reformulation step generates $\mathcal{L}$, the set of implied linear constraints. In other words, $\mathcal{L}$, contains $L1(y, \underline{x_i})$, $L1(y, \overline{x_i})$, $L2(y)$ for each variable x_i such that x_i^2 occurs in $[\mathcal{C}]_q$, and $L3(y_{i,j})$, $L4(y_{i,j})$, $L5(y_{i,j})$, $L6(y_{i,j})$ for each pair (i, j) such that the term $x_i xj$ occurs in $[\mathcal{C}]_q$.

The linearization step generates the linear inequalities and equalities $[[\mathcal{C}]_q]_l$. So, $LR = [[\mathcal{C}]_q]_l \cup \mathcal{L}$ is a linear relaxation of the original constraints $\mathcal{C}$. The filtering step is just a fixed point algorithm that calls iteratively a linear programming solver to reduce the upper and the lower bound of every original variable. The algorithm converges and terminates if ϵ is greater than zero.

3 A safe rounding procedure for the `Quad` algorithm

This section details the rounding procedure we propose to ensure the completeness of the `Quad` algorithm. First, we show how to compute safe coefficients for the generated linear constraints. In the second

Algorithm 1 The `Ext_Quad` and `Quad` algorithms

Function `Ext_Quad_filtering`(`IN`: $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, ϵ) **return** $\mathcal{D}'$
% $\mathcal{X}$: original variables; $\mathcal{D}$: input domains; $\mathcal{C}$: constraints; ϵ: minimal reduction

1. *Quadrification* : transforms the constraint system $\mathcal{C}$ into a quadratic system $[\mathcal{C}]_q$ and into a set of nonlinear identities $[\mathcal{C}]_{qI}$.
2. **return** `Quad`_filtering($\mathcal{X}$, $\mathcal{D}$, $[\mathcal{C}]_q$)

Function `Quad`_filtering(`IN`: $\mathcal{X}$, $\mathcal{D}$, $[\mathcal{C}]_q$, ϵ) **return** $\mathcal{D}'$
% $\mathcal{X}$: original variables ; $\mathcal{D}$: input domains; $[\mathcal{C}]_q$: quadratic constraints; ϵ: minimal reduction

1. *Reformulation* : generates implied inequalities $\mathcal{L}$ for the quadratic terms in $[\mathcal{C}]_q$.
2. *Linearization* : Linearization of the whole system $[\mathcal{C}]_q$. We obtain a linear system $LR = [[\mathcal{C}]_q]_l \cup \mathcal{L}$.
3. $\mathcal{D}' := \mathcal{D}$
4. **While** the reduction amount of some bound is greater than ϵ **and** $\emptyset \notin \mathcal{D}'$ **Do**
   (a) $\mathcal{D} \leftarrow \mathcal{D}'$
   (b) Update the coefficients of the linearizations $\underline{\mathcal{L}}$ according to the domain $\mathcal{D}'$
   (c) Reduce the lower and upper bounds $\underline{D'_i}$ and $\overline{D'_i}$ of each *original* variable $x_i \in \mathcal{X}$ by computing *min* and *max* of x_i subject to LR with a linear programming solver.

subsection we explain how a recent result from Neumaier allows us to use the simplex algorithm in a safe way.

3.1 Computing safe coefficients

a) Approximation of L_1 The linear constraint $L_1(y, \alpha) \equiv y - 2\alpha x + \alpha^2 \geq 0$ approximates a term x^2 with $x \in [\underline{x}, \overline{x}]$ and $\alpha = \underline{x}$ or $\overline{x}$. $L_1(y, \underline{x})$ (resp.$L_1(y, \overline{x})$) corresponds to the tangent line to the curve $y = x^2$ at the point $(\underline{\mathbf{x}}, \underline{\mathbf{x}}^2)$ (resp. $(\overline{\mathbf{x}}, \overline{\mathbf{x}}^2)$).

Thus, the computation over the floats of the coefficients of $L_1(y, \underline{x})$ and $L_1(y, \overline{x})$ may change the slope of these tangent lines as well as the intersection points with the curve $y = x^2$. Consider the case where α is negative : the solutions are above the tangent line and we have to decrease the slope to be sure to keep all of the solutions. It follows that we have to use a rounding mode towards $+\infty$. Likewise, when α is positive, we have to set the rounding mode towards $-\infty$. More formally, we have:

- $L_{1_{I\!F}}(y, \alpha) \equiv y - \lfloor 2\alpha \rfloor x + \lceil \alpha^2 \rceil \geq 0$ if $\alpha \geq 0$
- $L_{1_{I\!F}}(y, \alpha) \equiv y - \lceil 2\alpha \rceil x + \lceil \alpha^2 \rceil \geq 0$ if $\alpha < 0$

where $\lfloor x \rfloor$ (resp. $\lceil x \rceil$) denotes a rounding mode of x towards $-\infty$ (resp. $+\infty$).

b) Approximation of L_2 The case of L_2 is a bit more tricky since the "rotation axis" of the line defined by L_2 is between the extremum values of x^2 ($L2(y)$ is an overestimation of y). Thus, to keep all solutions we have to strengthen the slope of this line at its smallest extremum. It follows

that :

$$
\begin{aligned}
L_{2I\!F} &\equiv \lceil \underline{\mathbf{x}} + \overline{\mathbf{x}} \rceil x - y - \lfloor \underline{\mathbf{x}}\overline{\mathbf{x}} \rfloor \geq 0, \quad \text{if } \underline{\mathbf{x}} \geq 0 \\
L_{2I\!F} &\equiv \lfloor \underline{\mathbf{x}} + \overline{\mathbf{x}} \rfloor x - y - \lfloor \underline{\mathbf{x}}\overline{\mathbf{x}} \rfloor \geq 0 \quad \text{if } , \overline{\mathbf{x}} < 0 \\
L_{2I\!F} &\equiv \lceil \underline{\mathbf{x}} + \overline{\mathbf{x}} \rceil x - y - \lfloor \underline{\mathbf{x}}\overline{\mathbf{x}} + Ulp(\lceil \underline{\mathbf{x}} + \overline{\mathbf{x}} \rceil)\underline{\mathbf{x}} \rfloor \geq 0, \\
&\quad \text{if } \overline{\mathbf{x}} > 0,\ \underline{\mathbf{x}} < 0,\ |\underline{\mathbf{x}}| \leq |\overline{\mathbf{x}}| \\
L_{2I\!F} &\equiv \lfloor \underline{\mathbf{x}} + \overline{\mathbf{x}} \rfloor x - y - \lfloor \underline{\mathbf{x}}\overline{\mathbf{x}} - \lceil Ulp(\lceil \underline{\mathbf{x}} + \overline{\mathbf{x}} \rceil)\overline{\mathbf{x}} \rceil \rfloor \geq 0, \\
&\quad \text{if } \overline{\mathbf{x}} > 0,\ \underline{\mathbf{x}} < 0,\ |\underline{\mathbf{x}}| > |\overline{\mathbf{x}}|
\end{aligned}
$$

where $Ulp(x)$ computes the distance between x and the float following x.

c) Approximation of L_3, L_4, L_5 and L_6 The general form of L_3, L_4, L_5 and L_6 is $x_i x_j + s_1 b_1 x_i + s_2 b_2 x_j + s_3 b_1 b_2 \geq 0$ where b_1 and b_2 are floating point numbers corresponding to bounds of x_i and x_j whereas $s_i \in \{-1, 1\}$.

The term $s_3 b_1 b_2$ is the only term which corresponds to a computation result. Thus, these linear constraints can be rewritten in the following form : $Y + s_3 b_1 b_2$.

A rounding of $s_3 b_1 b_2$ towards $+\infty$ enlarges the solution space, and thus ensures that all these linear constraints are safe approximations of x^2.

It follows that $L_{3..6I\!F} \equiv Y + \lceil s_3 b_1 b_2 \rceil \geq 0$.

d) Approximation of the initial constant values The initial constant values are real numbers that may not have an exact floating point number representation. Thus, a safe approximation is required.

Constant values in inequalities have to be rounded toward $-\infty$ or $+\infty$ according to the orientation of the inequality. Equalities must be transformed into inequalities when their constant values have to be approximated.

3.2 Computation of safe bounds with the simplex algorithm

The Simplex method can solve problems of the following form :

$$\min\ c^T x \tag{8}$$

$$\text{such that}\ \ \underline{b} \leq Ax \leq \overline{b} \tag{9}$$

$$\text{and}\ \ \underline{x} \leq x \leq \overline{x} \tag{10}$$

The solution of such a problem is a vector $xr \in I\!R^n$. However, the solution computed by solvers like `CPLEX` or `SOPLEX` is a vector $xf \in I\!F^n$ that may be different from xr due to the rounding errors. More precisely, xf is safe for the objective only if $c^T xr \geq c^T xf$.

A recent work of Neumaier [16] provides a cheap method to obtain a rigorous bound of the objective and certificates of infeasibility. The essential observation is that the dual of (8) is

$$\max \underline{b}^T z' + \overline{b}^T z'' \tag{11}$$

$$\text{such that}\ \ A^T(z' - z'') = c \tag{12}$$

Let $y = z' - z''$, and the residue $r = A^T y - c$. It follows that

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

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

$$\mu = inf(y^T[\underline{b}, \overline{b}] - r^T[\underline{x}, \overline{x}]) = \lfloor y^T[\underline{b}, \overline{b}] - r^T[\underline{x}, \overline{x}] \rfloor \quad (13)$$

Formula (13) is trivially correct by construction. Note that the precision of such a safe bound depends on the width of the interval $[\underline{x}, \overline{x}]$.

So, we have just to apply this correction before updating the lower bound and the upper bound of each variable.

4 Experimental results

To evaluate the contribution of the `GFLR_solver` we have compared its performances with the computational results of the following solvers:

1. `Ilog solver` : a commercial implementation of `Numerica`[22] the filtering algorithm of which is based on Box–consistency .
2. `Quad_solver`, the solver introduced in [11]
3. `Box_Newton_solver`, a solver the filtering algorithm of which performs `interval Newton` filtering, Box–consistency filtering and `interval Newton` filtering in a sequential way ; in other words, the `Box_Newton_solver` corresponds to a `GFLR_solver` without `Ext_Quad` filtering.
4. `Ext_Quad_solver`, a solver that only performs `Ext_Quad` filtering; contrary to the `GFLR_solver` it achieves neither Box–consistency filtering nor `interval Newton` filtering.

The `GFLR_solver`, `Quad_solver`, `Box_Newton_solver` and `Ext_Quad_solver` have been implemented with our own constraint tool-kit. `Quad` and `Ext_Quad` filtering systems use the `CPLEX`[10] linear programming system. All the tests have been run on PC-Notebook computer at 1.2Ghz.

4.1 Benchmarks

The first benchmark, denoted by `kin2`, is from [15] and describes the inverse position problem for a six-revolute-joint problem in mechanics.

The other benchmarks describe a Gough-Stewart platform with variations on the initial position of the robot as well as on its geometry. The constraints systems consist of 9 equations with 9 variables. They express the length of the rods as well as the distances between the connection points. The maximum degree of the equations is 2, except for the Dietmaier problem where the degree is 4. Further informations on these problems can be found in [6] for the benchmark denoted by `Didrit`, in [12] for the benchmark denoted by `Lee`, and in [7] for the benchmark denoted by `Dietmaier`.

4.2 Computational results

The experimental results are reported in Tables 1–5.

`Kin2` and `Didrit` are easy problems that have been solved in a couple of seconds by all solvers but the `Box_Newton_`solver. The difference between `Ilog solver` and the `Box_Newton_`solver is probably due to some optimizations in the code of `Ilog solver` [4].

The `Lee` benchmark is more difficult : `Ilog solver` requires about 74 hours to find the four solutions whereas `Ext_Quad_`solver and `GFLR _`solver achieved the job in a bit more than one minute. The `Quad _`solver did find the four solutions in about 6 minutes whereas the `Box_Newton_`solver has been stopped after one hour.

The `GFLR _`solver managed to find forty solutions of the `Dietmaier` benchmark in about 22 minutes whereas the `Ext_Quad_`solver required about 74 hours and `Ilog solver` needed more than 800 hours ! The `Ext_Quad_`solver has been stopped after one hour. The `Quad _`solver could not tackle this problem since it is not quadratic.

The results on the `Lee` and `Dietmaier` benchmarks clearly shows that combining classical local consistencies and a global filtering algorithm pays offs on difficult problems. The essential observation is that these two solvers spend more time in the filtering step but they perform much less splitting than classical solvers.

Name	Splits	Nb Sols	T (s)
kin2	3485	10	31.45
Didrit	50739	4	157.09
Lee	30904886	4	266764.95
Dietmaier	100294739	40	3134026.38

Table 1. Results with `Ilog solver`

Name	Splits	Nb Sols	T (s)
kin2	32	10	13.01
Didrit	64	4	29.45
Lee	138	4	78.80
Dietmaier	1201	40	1349.11

Table 2. Results with the `GFLR_`solver

5 Conclusion

We have introduced in this paper a new filtering algorithm that combines local consistencies and a global filtering algorithm that work on a relaxation of the initial constraints system. First experimental results are very promising.

[4] No technical documentation on this part of `Ilog solver` is available.

Name	Splits	Nb Sols	T (s)
kin2	3585	8	$> 300^{\dagger}$
Didrit	8105	1	$> 300^{\dagger}$
Lee	66369	0	$> 3600^{\dagger}$
Dietmaier	22453	0	$> 3600^{\dagger}$

† These tests have been interrupted by a timer

Table 3. Results with the `Box_Newton` _solver

Name	Splits	Nb Sols	T (s)
kin2	28	10	11.20
Didrit	67	4	29.91
Lee	146	11	74.36
Dietmaier	8281	3511	$> 3600^{\dagger}$

† These tests have been interrupted by a timer

Table 4. Results with the `Ext_Quad` _solver

The proposed rounding procedure ensures the completeness of the `Quad` algorithm. More generally, it allows to tackle linear relaxations of continuous CSP without losing any solution.

Further works concerns the improvement of the searching strategy as well as the improvement of the quadrification process (the single quadrification does not capture some interesting quadratic identities in $[\mathcal{C}]_{qI}$, and does not apply any linearization on the non-polynomial part in $[\mathcal{C}]_{qI}$).

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.
2. F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages 8:2:273–286, 1983.
3. 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.
4. F. Benhamou, D. McAllester, and P. Van-Hentenryck. Clp(intervals) revisited. In *Proceedings of the International Symposium on Logic Programming*, pages 124–138, 1994.
5. R. L. Burden and J. D. Faires. *Numerical Analysis*. PWS-KENT, Boston, MA, 5^{th} edition, 1993.
6. 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.
7. Peter Dietmaier. The stewart-gough platform of general geometry can have 40 real postures. In *Advances in Robot Kinematics: Analysis and Control*, pages 1–10, 1998.
8. Eldon R. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 1992.
9. M. Rueher H.Collavizza, F.Delobel. Comparing partial consistencies. *Reliable Computing*, pages Vol.5(3),213–228, 1999.
10. Ilog, editor. *ILOG CPLEX 7.0, Reference Manual*. Ilog, 2000.
11. Yahia Lebbah, Michel Rueher, and Claude Michel. A global filtering algorithm for handling systems of quadratic equations and inequations. *Lecture Notes in Computer Science*, 2470:109–123, 2002.
12. T-Y Lee and J-K Shim. Elimination-based solution method for the forward kinematics of the general stewart-gough platform. In *In F.C. Park C.C. Iurascu, editor, Computational Kinematics, pages 259-267. 20-22 Mai*, 2001.

Name	Splits	Nb Sols	T (s)
kin2	11	10	69.30
Didrit	24	4	185.44
Lee	52	4	461.11

† These tests have been interrupted by a timer

Table 5. Results with the `Quad` _solver

13. O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of IJCAI'93*, pages 232–238, 1993.
14. A. Mackworth. Consistency in networks of relations. *Journal of Artificial Intelligence*, pages 8(1):99–118, 1977.
15. A.P. Morgan. Computing all solutions to polynomial systems using homotopy continuation. *Appl. Math. Comput.*, pages 24:115–138, 1987.
16. Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer programming. *http://www.mat.univie.ac.at/ neum/papers.html#mip*, page Submitted, 2002.
17. H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishing, 1999.
18. 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.
19. H.D. Sherali and C.H. Tuncbilek. Comparison of two reformulation-linearization technique based linear programming relaxations for polynomial programming problems. *Journal of Global Optimization*, pages 10:381–390, 1997.
20. N.Z. Shor. Dual quadratic estimates in polynomial and boolean programming. *Annals of Operations Research*, pages 25:163–168, 1990.
21. 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.
22. P. Van-Hentenryck, L. Michel, and Y. Deville. *Numerica : a Modeling Languge for Global Optimization*. MIT press, 1997.