EFFICIENT AND SAFE GLOBAL CONSTRAINTS FOR HANDLING NUMERICAL CONSTRAINT SYSTEMS

YAHIA LEBBAH[†][¶], CLAUDE MICHEL[‡][¶], MICHEL RUEHER[‡][¶], DAVID DANEY[§][¶], AND JEAN-PIERRE MERLET[§][¶]

– accepted for publication in the "SIAM Journal on Numerical Analysis"–

Abstract. Numerical constraint systems are often handled by branch and prune algorithms that combine splitting techniques, local consistencies and interval methods. This paper first recalls the principles of `Quad`, a global constraint that works on a tight and safe linear relaxation of quadratic subsystems of constraints. Then, it introduces a generalisation of `Quad` to polynomial constraint systems. It also introduces a method to get safe linear relaxations and shows how to compute safe bounds of the variables of the linear constraint system. Different linearisation techniques are investigated to limit the number of generated constraints. `QuadSolver`, a new branch and prune algorithm that combines `Quad`, local consistencies and interval methods is introduced. `QuadSolver` has been evaluated on a variety of benchmarks from kinematics, mechanics and robotics. On these benchmarks, it outperforms classical interval methods as well as CSP solvers and it compares well with state-of-the-art optimisation solvers.

Key words. systems of equations and inequalities, constraint programming, reformulation linearisation technique (RLT), global constraint, interval arithmetic, safe rounding.

AMS subject classifications. 65H10, 65G40, 65H20, 93B18, 65G20.

1. Introduction.

Many applications in engineering sciences require finding all isolated solutions to systems of constraints over real numbers. These systems may be non-polynomial and are difficult to solve: the inherent computational complexity is NP-hard and numerical issues are critical in practice (e.g., it is far from being obvious to guarantee correctness and completeness as well as to ensure termination). These systems, called numerical CSP[1] in the rest of this paper, have been approached in the past by different interesting methods[2] : interval methods [35, 24, 38, 20, 40], continuation methods [37, 2, 62] and constraint satisfaction methods [30, 6, 11, 61]. Of particular interest is the mathematical and programming simplicity of the latter approach: the general framework is a branch and prune algorithm that only requires specifying the constraints and the initial range of the variables.

The purpose of this paper is to introduce and to study a new branch and bound algorithm called `QuadSolver`. The essential feature of this algorithm is a global constraint –called `Quad`– that works on a tight and safe linear relaxation of the polynomial relations of the constraint systems. More precisely, `QuadSolver` is a branch and prune algorithm that combines `Quad`, local consistencies and interval methods. That is to say, `QuadSolver` is an attempt to merge the best interval and constraint programming techniques. `QuadSolver` has been evaluated on a variety of benchmarks from kinematics, mechanics and robotics. On these benchmarks, it outperforms classical interval methods as well as CSP solvers and it compares well with state-of-the-art optimisation solvers.

The `Quad`-filtering algorithm [27] has first been defined for quadratic constraints. The relaxation of quadratic terms is adapted from a classical linearisation method, the "Reformulation-Linearisation Technique (RLT)" [54, 53]. The simplex algorithm is used to narrow the domain of

[†]Département Informatique, Faculté des Sciences, Université d'Oran Es-Senia, B.P. 1524, El-M'Naouar Oran, Algeria

[‡]Université de Nice–Sophia Antipolis, I3S–CNRS, 930 route des Colles, B.P. 145, 06903 Sophia Antipolis Cedex, France

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

[¶]COPRIN Project INRIA–I3S–CNRS

[1]CSP stands for Constraint Satisfaction Problem.

[2]Alternative methods have been proposed for solving nonlinear systems. For instance, algebraic constraints can be handled with symbolic methods [13] (e.g. Groebner Basis, Resultant). However, these methods can neither handle non-polynomial systems nor deal with inequalities.

each variable with respect to the subset of the linear set of constraints generated by the relaxation process. 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. We have demonstrated [27] that the Quad algorithm yields a more effective pruning of the domains than local consistency filtering algorithms (e.g., 2b–consistency [30] or box–consistency [6]). Indeed, the drawback of classical local consistencies comes from the fact that the constraints are handled independently and in a blind way[3]. That is to say, classical local consistencies do not exploit the semantic of quadratic terms; in other words, these approaches do not take advantage of the very specific semantic of quadratic constraints to reduce the domains of the variables. Conversely, linear programming techniques [1, 54, 3] do capture most of the semantics of quadratic terms[4] e.g., convex and concave envelopes of these particular terms.

The extension of Quad for handling any polynomial constraint system requires to replace non-quadratic terms by new variables and to add the corresponding identities to the initial constraint system. However, a complete quadrification [58] would generate a huge number of linear constraints. Thus, we introduce here an heuristic based on a good tradeoff between a tight approximation of the nonlinear terms and the size of the generated constraint system. This heuristic works well on classical benchmarks (see § 8).

A safe rounding process is a key issue for the Quad framework. Let's recall that 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 point is that most implementations of the simplex algorithm are unsafe. Moreover, the coefficients of the generated linear constraints are computed with floating point numbers. So, two problems may occur in the Quad-filtering process:

1. The whole linearisation may become incorrect due to rounding errors when computing the coefficients of the generated linear constraints ;
2. Some solutions may be lost when computing the bounds of the domains of the variables with the simplex algorithm.

We propose in this paper a safe procedure for computing the coefficients of the generated linear constraints. Neumaier and Shcherbina [42] have addressed the second problem[5]. They have proposed a simple and cheap procedure to get a rigorous upper bound of the objective function. The incorporation of these procedures in the Quad-filtering process allows us to call the simplex algorithm without worrying about safety. So, with these two procedures, linear programming techniques can be used to tackle continuous CSPs *without losing any solution*.

The rest of this paper is organised as follows. Section 2 gives an overview of the approach whereas Section 3 contains the notation. Sections 4 and 5 recall the basic of interval programming and constraint programming. Section 6 details the principle of the Quad algorithm, the linearisation process and the extension to polynomial constraints. Section 7 introduces the rounding process we propose to ensure the safe relaxations. Section 8 describes the experimental results and discusses related work.

[3] 3b–consistency and kb–consistency are partial consistencies that can achieve a better pruning since they are "less local" [11]. However, they require numerous splitting steps to find the solutions of a system of quadratic constraints; so, they may become rather slow.

[4] Sherali and Tuncbilek [55] have also proposed four different filtering techniques for solving quadratic problems. Roughly speaking, the first filtering strategy performs a feasibility check on inequality constraints to discard subintervals of the domains of the variables. This strategy is very close to *Box*-consistency filtering (see [60]). The three other techniques are based on specific properties of optimisation problems with a quadratic objective function: the eigenstructure of the quadratic objective function, fathoming node and Lagrangian dual problem. Thus, these techniques can be considered as local consistencies for optimisation problems (see also [59] and Neumaier's survey [41]).

[5] They also suggest a solution to the first problem though their solution is dedicated to MIP problems.

2. Overview of the approach. As mentioned, `QuadSolver` is a branch and prune algorithm that combines `Quad` and a box-consistency.
Box-consistency is the most successful adaptation of arc–consistency [31] to constraints over the real numbers. The box–consistency implementation of P. Van Hentenryck et al. [60] is computed on three interval extensions of the initial constraints : the natural interval extension, the distributed interval extension, and the Taylor interval extension with a conditioning step. The leftmost and the rightmost zero are computed using a variation of the univariate interval Newton method.
The `QuadSolver` we propose here combines `Quad`-filtering and box–consistency filtering to prune the domain of the variables of numerical constraint systems. Operationally, `QuadSolver` performs the following filtering process:

1. box–consistency filtering,
2. `Quad`-filtering.

The box–consistency is first used to detect some inconsistencies before starting the `Quad` filtering algorithm which is more costly. These two steps are wrapped into a classical fixed point algorithm which stops when the domains of the variables can not be further reduced[6].

To isolate the different solutions, `Quad` uses classical branching techniques.

Before going into the details, let us outline the advantages of our approach on a couple of small examples.

2.1. `Quad`-filtering.
Consider the constraint system $\mathcal{C} = \{2xy + y = 1, xy = 0.2\}$ which represents two intersecting curves (see figure 2.1). Suppose that $\mathbf{x} = [-10, +10]$ and $\mathbf{y} = [-10, +10]$ are the domains of the variables x and y. An interval $\mathbf{x} = [\underline{x}, \overline{x}]$ denotes the set of reals $\{r | \underline{x} \le r \le \overline{x}\}$.

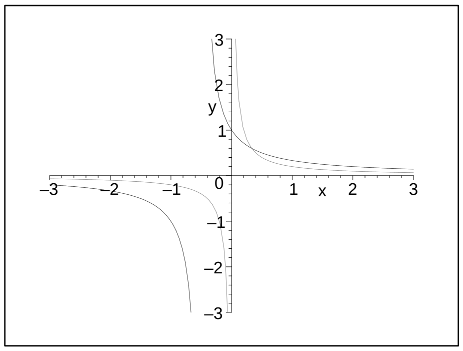

FIG. 2.1. *Geometrical representation of* $\{2xy + y = 1, xy = 0.2\}$

The reformulation-linearisation technique (see § 6.2) yields the following constraint system:

$$(a) \begin{cases} y + 2w = 1, \quad w = 0.2, \\ \underline{y}x + \underline{x}y - w \le \underline{x}\underline{y}, \quad \overline{y}x + \underline{x}y - w \ge \underline{x}\overline{y}, \\ \underline{y}x + \overline{x}y - w \ge \overline{x}\underline{y}, \quad \overline{y}x + \overline{x}y - w \le \overline{x}\overline{y}, \\ x \ge \underline{x}, \; x \le \overline{x}, \quad y \ge \underline{y}, \; y \le \overline{y} \end{cases}$$

where w is a new variable that stands for the product xy. Note that constraint system (a) implies that $w \in [\underline{x}, \overline{x}] * [\underline{y}, \overline{y}]$.
Substituting $\underline{x}, \underline{y}, \overline{x}$ and $\overline{y}$ by their values and minimising (resp. maximising) x, y and w with the

[6]In practice, the loop stops when the domain reduction is lower than a given ϵ.

simplex algorithm yields the following new bounds:

$$\mathbf{x} = [-9.38, 9.42], \quad \mathbf{y} = [0.6, 0.6], \quad \mathbf{w} = [0.2, 0.2].$$

By substituting the new bounds of x, y and w in the constraint system (a), we obtain a new linear constraint system. One more minimising (resp. maximising) step is required to obtain tight bounds of $\mathbf{x}$. Note that numerous 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 6 linear constraints and with 8 calls to the simplex algorithm. It finds the same solution than a solver based on 3b-consistency but without splitting and in less time.

2.2. A safe rounding procedure.

Consider the constraint system

$$\mathcal{C} = \begin{cases} w_1 + w_2 = 1, \quad w_1x_1 + w_2x_2 = 0, \\ w_1x_1x_1 + w_2x_2x_2 = 1, \quad w_1x_1x_1x_1 + w_2x_2x_2x_2 = 0 \end{cases}$$

which represents a simple Gaussian quadrature formula to compute integrals [9]. Suppose that the domains of variables x_1, x_2, w_1 and w_2 are all equal to $[-1, +1]$. This system has two solutions:

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

A straightforward implementation of `Quad` would only find one unsafe solution with:

$$x_2 \in [+0.9999...944, +0.9999...989]$$

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

With the corrections we propose in § 7, we obtain a tight approximation of the two correct solutions (with $x_2 \in [-1.000000..., -0.999999...]$ and $x_2 \in [0.999999..., 1.000000...]$).

3. Notation and basic definitions.

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

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

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

We use the “reformulation-linearisation technique” (RLT) notation introduced in [54, 3] with slight modifications. More precisely, we will use the following notations: $[c]_L$ is the set of linear constraints generated by replacing the nonlinear terms by new variables in constraint c and $[c]_{LI}$ denotes the set of equations that keep the link between the new variables and the nonlinear terms while $[c]_R$ contains linear inequalities that approximate the semantics of nonlinear terms of constraint c. These notations will be used indifferently whether c is a constraint or $\mathcal{C}$ is a set of constraints.

Rounding is necessary to close the operations over $\mathbb{F}$ [18]. A rounding function maps the result of the evaluation of an expression to available floating-point numbers. Rounding x towards $+\infty$ maps x to the least floating point number x_f such that $x \leq x_f$. $\bigtriangledown(x)$ (resp. $\bigtriangleup(x)$) denotes a rounding mode of x towards $-\infty$ (resp. $+\infty$).

4. Interval Programming. This section recalls the basic concepts of interval arithmetic that are required to understand the rest of the paper. Readers familiar with interval arithmetic may skip this section.

4.1. Interval Arithmetic. *Interval arithmetic* has been introduced by Moore [35]. It is based on the representation of variables as intervals.

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

$$\underline{y} \leq f(\mathbf{X}) \leq \overline{y}, \quad \text{for all} \;\; X = (x_1, \dots, x_n) \in \mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \tag{4.1}$$

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

There are numerous ways to calculate an interval evaluation of a function [20, 46]. The simplest is the *natural evaluation* in which all the mathematical operators in f are substituted by their interval equivalents. Interval equivalents exist for all classical mathematical operators. Hence interval arithmetic allows to calculate an interval evaluation for all nonlinear expressions, whether algebraic or not. For example if $f(x) = x + \sin(x)$, then the interval evaluation of f for $x \in [1.1, 2]$ can be calculated as follows:

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

Interval arithmetic can be implemented with directed rounding to take into account round-off errors. There are numerous interval arithmetic packages implementing this property: one of the most famous library is BIAS/Profil[7] but a promising new package — based on the multi-precision software MPFR[8] — is MPFI [47].

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

- the so-called *wrapping effect* [35, 39], which overestimates by a unique vector the image of an interval vector (which is in general not a vector). That is to say, $\{f(X) | X \in \mathbf{X}\}$ is contained in $f(\mathbf{X})$ but is usually not equal to $f(\mathbf{X})$.
- the so-called *dependency problem* [20], which is due to the independence of the different occurrences of some variables during the interval evaluation of an expression. In other words, during the interval evaluation process there is no correlation between the different occurrences of a same variable in an equation. For instance, consider $\mathbf{x} = [0, 10]$. $\mathbf{x} - \mathbf{x} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-10, 10]$ instead of $[0, 0]$ as one could expect.

In general, it is not possible to compute the exact enclosure of the range for an arbitrary function over the real numbers [25]. Thus, Moore introduced the concept of *interval extension*: the interval extension of a function is an interval function that computes outer approximations on the range of the function over a domain [20, 36]. Two main extensions have been introduced: the natural extension and the Taylor extension[9] [46, 20, 38]. Due to the properties of interval arithmetic, the evaluation of a function may yield different results according to the literal form of the equations. Thus, many literal forms may be used as, for example, factorised form (Horner for polynomial system) or distributed form [60].

Nevertheless, in general, neither the natural form nor the Taylor expansion allow to compute the exact range of a function f. For instance, consider $f(x) = 1 - x + x^2$, and $\mathbf{x} = [0, 2]$, we have:

$$\begin{aligned} &f_{\text{tay}}([0,2]) = f(x) + (2\mathbf{x} - 1)(\mathbf{x} - x) = f(1) + (2[0,2] - 1)([0,2] - 1) = [-2, 4], \\ &f([0,2]) = 1 - \mathbf{x} + \mathbf{x}^2 = 1 - [0,2] + [0,2]^2 = [-1, 5], \\ &f_{\text{factor}}([0,2]) = 1 + \mathbf{x}(\mathbf{x} - 1) = 1 + [0,2]([0,2] - 1) = [-1, 3] \end{aligned} \tag{4.2}$$

[7]http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

[8]http://www.mpfr.org

[9]$f_{\text{tay}}(\mathbf{X}) = f(X) + \mathbf{A}(\mathbf{X} - X)$ where $\mathbf{A}$ is the Jacobian or the interval slope matrix.

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

4.2. Interval analysis methods. This section provides a short introduction on interval analysis methods (see [35, 20, 38, 40] for a more detailed introduction). We limit this overview to interval Newton-like methods for solving a multivariate system of nonlinear equations. Their use is complementary to methods provided by the Constraint Programming community.

The aim is to determine the zeros of a system of n equations $f_i(x_1, \dots, x_n)$ in n unknowns x_i inside the interval vector $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with $x_i \in \mathbf{x}_i$ for $i = 1, \dots, n$.

First, consider solving an interval linear system of equations defined as follows:

$$AX = b, A \in \mathbf{A}, b \in \mathbf{b} \tag{4.3}$$

where $\mathbf{A}$ is an interval matrix and $\mathbf{b}$ is an interval vector. Solving this linear interval system requires to determine an interval vector $\mathbf{X}$ containing all solutions of all scalar linear systems noted $AX = b$ such that $A \in \mathbf{A}$ and $b \in \mathbf{b}$. Finding the exact value of $\mathbf{X}$ is a difficult problem but three basic interval methods exist: Gaussian elimination, Gauss-Seidel iterative method or Krawczyk method (see [24, 38, 20, 40]). They may provide an overestimated interval vector $\mathbf{X}_1$ including $\mathbf{X}$. However, in general the computed interval are too wide and a preconditioning is required. That is to say, a multiplication of both side of equation (4.3) by the inverse of a midpoint of $\mathbf{A}$. The matrix $m(\mathbf{A})^{-1}\mathbf{A}$ is then "closer" to the identity matrix and the width of $\mathbf{X}_1$ is smaller [20].

To solve nonlinear systems, an interval Newton algorithm is often used – see [20] or [38]. The basic idea is to solve iteratively a linear approximation of the nonlinear system obtained by a Taylor expansion. Many improvements [24, 19], based on variations of the resolution of the linear subsystem or the preconditioning have been proposed. Note that many interesting properties are provided by Newton-like methods: existence and/or uniqueness of a root, convergence area/rate, ...

5. Constraint programming. This section recalls the basics of constraint programming techniques which are required to understand the rest of this paper. A detailed discussion of these concepts and techniques can be found in [6, 26].

5.1. The general framework. The constraint programming framework is based on a branch and prune scheme which was inspired by the traditional branch and bound approach used in optimisation problems. That is to say, it is best viewed as an iteration of two steps [60]:

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

The pruning step ensures that some local consistency holds. In other words, the pruning step reduces an interval when it can prove that the upper bound or the lower bound does not satisfy some constraint. Informally speaking, a constraint system $\mathcal{C}$ satisfies a partial consistency property if a relaxation of $\mathcal{C}$ is consistent. For instance consider $\mathbf{x} = [\underline{x}, \overline{x}]$ and $c(x, x_1, \dots, x_n) \in \mathcal{C}$. Whenever $c(x, x_1, \dots, x_n)$ does not hold for any values $a \in \mathbf{x} = [\underline{x}, x']$, then $\mathbf{x}$ may be shrunk to $\mathbf{x} = [x', \overline{x}]$. Local consistencies are detailed in the next subsection. Roughly speaking, they are relaxations of arc-consistency, a notion well known in artificial intelligence [31, 34].

The branching step usually splits the interval associated to some variable in two intervals with the same width. However, the splitting process may generate more than two sub-problems and one may split an interval at a point different from its midpoint. The choice of the variable to split is a critical issue in difficult problems. Sophisticated splitting strategies have been developed for finite domains but few results [23] are available for continuous domains.

5.2. Local consistencies [11, 26]. Local consistencies are conditions that filtering algorithms must satisfy. A filtering algorithm can be seen as a fixed point algorithm defined by the sequence $\{\mathcal{D}_k\}$ of domains generated by the iterative application of an operator $Op : \mathbb{IR}^n \longrightarrow \mathbb{IR}^n$ (see Figure 5.1).

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

FIG. 5.1. *Filtering algorithms as fixed point algorithms*

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

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

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

Consistencies used in numerical CSPs solvers can be categorised in two main classes: *arc-consistency*-like consistencies and strong consistencies. Strong consistencies will not be discussed in this paper (see [30, 26] for a detailed introduction).

Most of the numerical CSPs systems (e.g., BNR-prolog [43], Interlog [8], CLP(BNR) [7], PrologIV [12], UniCalc [4], Ilog Solver [22], Numerica [61] and RealPaver [5]) compute an approximation of arc-consistency [31] which will be named ac-like-consistency in this paper. An *ac*-like-consistency states a local property on a constraint and on the bounds of the domains of its variables. Roughly speaking, a constraint c_j is ac-like-consistent if for any variable x_i in $var(c_j)$, the bounds $\underline{x}_i$ and $\overline{x}_i$ have a support in the domains of all other variables of c_j.

The most famous ac-like consistencies are 2b–consistency and box–consistency.

2b–consistency (also known as hull consistency) [10, 7, 28, 30] only requires to check the arc–consistency property for each bound of the intervals. The key point is that this relaxation is more easily verifiable than arc–consistency itself. Informally speaking, variable x is 2b–consistent for constraint $''f(x, x_1, \ldots, x_n) = 0''$ if the lower (resp. upper) bound of the domain of x is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$. The box–consistency [6, 21] is a coarser relaxation (i.e., it allows less stringent pruning) of arc–consistency than 2b–consistency. Variable x is box–consistent for constraint $''f(x, x_1, \ldots, x_n) = 0''$ if the bounds of the domain of x correspond to the leftmost and the rightmost zero of the optimal interval extension of $f(x, x_1, \ldots, x_n)$. 2b–consistency algorithms actually achieve a weaker filtering (i.e., a filtering that yields bigger intervals) than box–consistency, more precisely when a variable occurs more than once in some constraint (see proposition 6 in [11]). This is due to the fact that 2b–consistency algorithms require a decomposition of the constraints with multiple occurrences of the same variable.

2b–consistency [30] states a local property on the bounds of the domains of a variable at a single constraint level. A constraint c is 2b–consistent if, for any variable x, there exist values in the domains of all other variables which satisfy c when x is fixed to $\underline{x}$ and $\overline{x}$.

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

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

Filtering by 2b–consistency of P always exists and is unique [30], that is to say it is a closure.

The box–consistency [6, 21] 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. Thus, box–consistency generates a system of univariate interval functions which can be tackled by numerical methods such as interval Newton. In contrast to 2b–consistency, box–consistency does not require any constraint decomposition and thus does not amplify the locality problem. Moreover, box–consistency can tackle some dependency problems when each constraint of a CSP contains only one variable which has multiple occurrences. More formally:

Definition 5.1 (box–consistency).
Let $(\mathcal{D},\mathcal{C})$ be a CSP and $c \in \mathcal{C}$ a k-ary constraint over the variables $(x_1,\ldots,x_k)$. c is box–consistent if, for all x_i the following relations hold :
1. $c(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1},[\underline{x}_i,\underline{x}_i^+),\mathbf{x}_{i+1},\ldots,\mathbf{x}_k)$,
2. $c(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1},(\overline{x}_i^-,\overline{x}_i],\mathbf{x}_{i+1},\ldots,\mathbf{x}_k)$.

Closure by box–consistency of P is defined similarly to closure by 2b–consistency of P.

Benhamou et al. have introduced hc4 [5], an ac-like-consistency that merges 2b–consistency and box–consistency and which optimises the computation process.

6. `Quad` basics and extensions.

This section first introduces `Quad`, a global constraint that works on a tight and safe linear relaxation of quadratic subsystems of constraints. Then, it generalises `Quad` to the polynomial part of numerical constraint systems. Different linearisation techniques are investigated to limit the number of generated constraints.

6.1. The `Quad` algorithm.

The `Quad`-filtering algorithm (see Algorithm 1) consists of three main steps: reformulation, linearisation and pruning.

The reformulation step generates $[\mathcal{C}]_R$, the set of implied linear constraints. More precisely, $[\mathcal{C}]_R$ contains linear inequalities that approximate the semantics of nonlinear terms of $\mathcal{C}$.

The linearisation process first decomposes each nonlinear term in sums and products of univariate terms, then, it replaces nonlinear terms with their associated new variables. For example, consider constraint $c : x_2x_3x_4^2(x_6+x_7)+\sin(x_1)(x_2x_6-x_3)=0$, a simple linearisation transformation may yield the following sets:

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

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

Finally, the linearisation step computes the set of final linear inequalities and equations $LR = [\mathcal{C}]_L \cup [\mathcal{C}]_R$, the linear relaxation of the original constraints $\mathcal{C}$.

The pruning 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.

Now we are in position to introduce the reformulation of nonlinear terms. Subsection 6.2 first introduces the handling of quadratic constraints while subsection 6.3 extends the previous results to polynomial constraints.

6.2. Handling quadratic constraints.

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_ix_j. The produced system is denoted:

$$[\sum_{(i,j)\in M} a_{k,i,j}x_ix_j + \sum_{i\in N} b_{k,i}x_i^2 + \sum_{i\in N} d_{k,i}x_i = b_k]_L$$

```
Function Quad_filtering(IN: $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, $\epsilon$) return $\mathcal{D}'$
% $\mathcal{X}$: initial variables ; $\mathcal{D}$: input domains; $\mathcal{C}$: constraints; $\epsilon$: minimal reduction
% $\mathcal{D}'$: output domains
    1. Reformulation: generation of linear inequalities $[\mathcal{C}]_R$ for the nonlinear terms in $\mathcal{C}$.
    2. Linearisation: linearisation of the whole system $[\mathcal{C}]_L$.
       We obtain a linear system $LR = [\mathcal{C}]_L \cup [\mathcal{C}]_R$.
    3. $\mathcal{D}' := \mathcal{D}$
    4. Pruning:
       While the amount of reduction of some bound is greater than $\epsilon$ and $\emptyset \notin \mathcal{D}'$ Do
         (a) $\mathcal{D} \leftarrow \mathcal{D}'$
         (b) Update the coefficients of the linearisations $[\mathcal{C}]_R$ according to the domains $\mathcal{D}'$
         (c) Reduce the lower and upper bounds $\underline{x}'_i$ and $\overline{x}'_i$ of each initial variable $x_i \in \mathcal{X}$ by
             computing min and max of $x_i$ subject to $LR$ with a linear programming solver.
```

ALGORITHM 1
The Quad-*algorithm*

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 preserve equations $y = x^2$ and $y_{i,j} = x_i x_j$ and that provide a better approximation than interval arithmetic [27].

6.2.1. Linearisation of x^2. The term x^2 with $\underline{x} \le x \le \overline{x}$ is approximated by the following relations:

$$[x^2]_R = \begin{cases} L1(\alpha) & \equiv & [(x-\alpha)^2 \ge 0]_L \text{ where } \alpha \in [\underline{x}, \overline{x}] \\ L2 & \equiv & [(\underline{x}+\overline{x})x - y - \underline{x}\overline{x} \ge 0]_L \end{cases} \tag{6.1}$$

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 $L1(\overline{x})$ and $L1(\underline{x})$. Consider for instance the quadratic term x^2 with $x \in [-4, 5]$. Figure 6.1 displays the initial curve (i.e., D_1), and the lines corresponding to the equations generated by the relaxations: D_2 for $L1(-4) \equiv y + 8x + 16 \ge 0$, D_3 for $L1(5) \equiv y - 10x + 25 \ge 0$, and D_4 for $L2 \equiv -y + x + 20 \ge 0$.

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.

6.2.2. Bilinear terms. In the case of bilinear terms xy, McCormick [32] proposed the following relaxations of xy over the box $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$, stated in the equivalent RLT form [54]:

$$[xy]_R = \begin{cases} BIL1 & \equiv & [(x-\underline{x})(y-\underline{y}) \ge 0]_L \\ BIL2 & \equiv & [(x-\underline{x})(\overline{y}-y) \ge 0]_L \\ BIL3 & \equiv & [(\overline{x}-x)(y-\underline{y}) \ge 0]_L \\ BIL4 & \equiv & [(\overline{x}-x)(\overline{y}-y) \ge 0]_L \end{cases} \tag{6.2}$$

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 [1] 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 6.2 displays the 2D shape, for the level $z = 5$, of the initial curve (i.e., Cu), and the lines corresponding

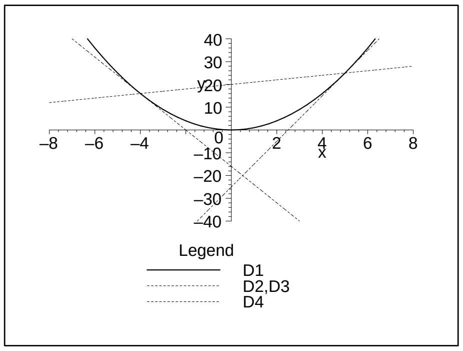

FIG. 6.1. *Approximation of x^2*

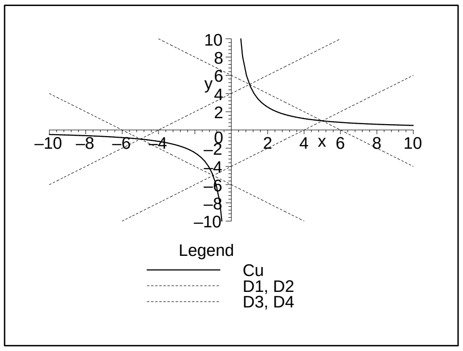

FIG. 6.2. *Illustration of xy relaxations*

to the equations generated by the relaxations (where $z = 5$): D_1 for $BIL1 \equiv z + 5x + 5y + 25 \geq 0$, D_2 for $BIL2 \equiv -z + 5x - 5y + 25 \geq 0$, D_3 for $BIL3 \equiv -z - 5x + 5y + 25 \geq 0$, and D_4 for $BIL4 \equiv z - 5x - 5y + 25 \geq 0$.

6.3. Extension to polynomial constraints. In this section, we show how to extend the linearisation process to polynomial constraints. We first discuss the quadrification process and compare it with RLT. Then, we present the linearisations of product and power terms.

6.3.1. Transformation of nonlinear constraints into quadratic constraints. In this section, we show how to transform a polynomial constraint system into an equivalent quadratic constraint system, a process called *quadrification* [58].

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

$$\{y_1y_2 + 3y_2 + s_1 = 0, \quad y_1 = x_2x_3, \quad y_2 = x_4x_4, \quad y_3 = x_6x_7\}$$

and the set $\{y_1 = x_2x_3, \quad y_2 = x_4^2, \quad y_3 = x_6x_7, \quad s_1 = \sin(x_1)\}$ of equations that keep the link between the new variables and the nonlinear 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 of $E = \{x_2x_3x_4^2 + 3x_6x_7 + \sin(x_1) = 0\}$ is

$$\left\{\begin{array}{llll}
y_1 + 3y_2 + s_1 = 0, & y_2 = x_6x_7, & & \\
y_1 = y_3y_4, & y_3 = x_2x_3, & y_4 = x_4^2, & \\
y_1 = y_5y_6, & y_5 = x_2x_4, & y_6 = x_3x_4, & \\
y_1 = x_2y_7, & y_7 = x_3y_4, & y_7 = x_4y_6, & \\
y_1 = x_3y_8, & y_8 = x_2y_4, & y_8 = x_4y_5, & \\
y_1 = x_4y_9, & y_9 = x_2y_6, & y_9 = x_3y_5, & y_9 = x_4y_3
\end{array}\right.$$

where $s_1 = \sin(x_1)$.

A quadrification for polynomial problems was introduced by Shor [58]. Sherali and Tuncbilek [57] have proposed a direct reformulation/linearisation (RLT) of the whole polynomial constraints without quadrifying the constraints. They did prove the dominance of their direct reformulation/linearisation technique over Shor's quadrification [56].

A complete quadrification generates as many new variables as the direct RLT. Linearisations proposed in RLT are built on every non-ordered combination of δ variables, where δ is the highest polynomial degree of the constraint system.

The complete quadrification generates linearisations on every couple of non-ordered combined variables $[v_i, v_j]$ where v_i (resp. v_j) is the variable that has been introduced for linearising the non-ordered combination of variables.

Complete quadrification and direct RLT yields a tighter linearisation than the single quadrification but the number of generated linearisations grows in an exponential way for non trivial polynomial constraint systems. More precisely, the number of linearisations depends directly on the number of generated new variables.

To sum up, the linearisation of polynomial systems offers two main possibilities : the transformation of the initial problem into an equivalent quadratic constraint system through a process called *quadrification*, or the direct linearisation of polynomial terms by means of RLT. Theoretical considerations, as well as experimentations, have been conducted to exclude as practical a complete quadrification which produces a huge amount of linear inequalities for non trivial polynomial systems. The next two subsections presents our choices for the linearisation of product and power terms.

6.3.2. Product terms.

For the product term

$$x_1 x_2 ... x_n \tag{6.3}$$

we use a two steps procedure: quadrification and bilinear relaxations.

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 $T \equiv x_1 x_2 \dots x_n$, a monomial of degree n, the middle heuristic will identify two monomials T_1 and T_2 of highest degree such that $T = T_1 T_2$. It follows that $T_1 = x_1 x_2 \dots x_{n \div 2}$ and $T_1 = x_{n \div 2+1} \dots x_n$.

The quadrification is performed by recursively decomposing each product $x_i ... x_j$ into two products $x_i ... x_d$ and $x_{d+1} ... x_j$. Of course, there are many ways to choose the position of d. Sahnidis et al. [49, 51] use what they call `rAI`, “recursive interval arithmetic”, which is a recursive quadrification where $d = j - 1$. We use the middle heuristic `Qmid`, where $d = (i + j)/2$, to obtain balanced degrees on the generated terms. Let us denote $[E]_{RI}$ the set of equations that transforms a product terms into a set of quadratic identities.

The second step consists in a *bilinear relaxation* $[[\mathcal{C}]_{RI}]_R$ of all the quadratic identities in $[\mathcal{C}]_{RI}$ with the bilinear relaxations introduced in § 6.2.2.

Sherali and Tuncbilek [57] have proposed a promising direct reformulation/linearisation technique (RLT) of the whole polynomial constraints without quadrifying the constraints. Applying RLT on the product term $x_1 x_2 ... x_n$ generates the following n-ary inequalities [10]:

$$\prod_{i \in J_1} (x_i - \underline{x}_i) \prod_{i \in J_2} (\overline{x}_i - x_i) \geq 0, \quad \text{for all } J_1, J_2 \subseteq \{1, \dots, n\} \colon |J_1 \cup J_2| = n \tag{6.4}$$

where $\{1, \dots, n\}$ is to be understood as a multiset and where J_1 and J_2 are multisets.

We now introduce proposition 6.1 that states the number of new variables and relaxations respectively generated by the quadrification and RLT process on the product term (6.3).

Proposition 6.1. *Let $T \equiv x_1 x_2 \dots x_n$ be some product of degree $n \geq 1$ with n distinct variables. The RLT of T will generate up to $(2^n - n - 1)$ new variables and 2^n inequalities whereas the quadrification of T will only generate $(n - 1)$ new variables and $4(n - 1)$ inequalities.*

[10] Linearisations proposed in RLT on the whole polynomial problem are built on every non-ordered combination of δ variables, where δ is the highest polynomial degree of the constraint system.

Proof. The number of terms of length i is clearly the number of combinations of i elements within n elements, that is to say C_n^i. In the RLT relaxations (6.4), we generate new variables for all these combinations. Thus, the number of variables is bounded by $\sum_{i=2...n} \mathrm{C}_n^i = \sum_{i=0...n} \mathrm{C}_n^i - n - 1$, that is to say $2^n - n - 1$ since $\sum_{i=0...n} \mathrm{C}_n^i = 2^n$. In (6.4), for each variable we consider alternatively lower and upper bound, thus there are 2^n new inequalities.

For the quadrification process, the proof can be done by induction. For $n = 1$, the formula is true. Now suppose that for length i (with $1 \leq i < n$), $(i - 1)$ new variables are generated. For $i = n$, we can split the term at the position d with $1 \leq d < n$. It results from the induction hypothesis that we have $d - 1$ new variables for the first part, and $n - d - 1$ new variables for the second part, plus one more new variable for the whole term. So, $n - 1$ new variables are generated. Bilinear terms require four relaxations, thus we get $4(n - 1)$ new inequalities. □

Proposition 6.2 states that quadrification with bilinear relaxations provides convex and concave envelopes with any d. This property results from the proof given in [49] for the `rAI` heuristic.

PROPOSITION 6.2. $[[x_1 x_2 ... x_n]_{RI}]_R$ *provides convex and concave envelopes of the product term* $x_1 x_2 ... x_n$.

Generalisation for sums of products, the so-called multi-linear terms $\sum_{i=1...t} a_i \prod_{j \in J_i} x_j$, have been studied recently [14, 52, 48, 49]. It is well known that finding the convex or concave envelope of a multi-linear term is a NP-hard problem [14]. The most common method of linear relaxation of multi-linear terms is based on the simple product term. However, it is also well known that this approach leads to a poor approximation of the linear bounding of the multi-linear terms. Sherali [52] has introduced formulae for computing convex envelopes of the multi-linear terms. It is based on an enumeration of vertices of a pre-specified polyhedra which is of exponential nature. Rikun [48] has given necessary and sufficient conditions for the polyhedrality of convex envelopes. He has also provided formulae of some faces of the convex envelope of a multi-linear function. To summarise, it is difficult to characterise convex and concave envelopes for general multi-linear terms. Conversely, the approximation of "product of variables" is an effective approach; moreover, it is easy to implement [51, 50].

6.3.3. Power terms. A power term of the form x^n can be approximated by $n + 1$ inequalities with a procedure proposed by Sherali and Tuncbilek [57], called "bound-factor product RLT constraints". It is defined by the following formula:

$$[x^n]_R \;=\; \{[(x - \underline{x})^i (\overline{x} - x)^{n-i} \geq 0]_L, i = 0...n\} \tag{6.5}$$

The essential observation is that this relaxation generates a tight relations between variables on their upper and lower bounds. More precisely, suppose that some original variable takes a value equal to either of its bounds. Then all the corresponding new RLT linearisation variables that involve this original variable take a relative values that conform with actually fixing this original variable at its particular bound in the nonlinear expressions represented by these new RLT variables [57].
Note that relaxations (6.5) of the power term x^n are expressed with x^i for all $i \leq n$, and thus provide a fruitful relationship on problems containing many power terms involving some variable.

The univariate term x^n is convex when n is even, or when n is odd and the value of x is negative; it is concave when n is odd and the value of x is positive. Sahinidis and Twarmalani [50] have introduced the convex and concave envelopes when n is odd by taking the point where the power term x^n and its under-estimator have the same slope. These convex/concave relaxations on x^n are expressed with only $[x^n]_L$ and x. In other words, they do not generate any relations with x^i for $1 < i < n$.

That is why we suggest to implement the approximations defined by formulae (6.5). Note that for the case $n = 2$, (6.5) provides the concave envelope.

7. A safe rounding procedure for the `Quad`-algorithm.

This section details the rounding procedure we propose to ensure the completeness of the `Quad` algorithm [33]. First, we show how to compute safe coefficients for the generated linear constraints. In the second subsection we explain how a recent result from Neumaier and Shcherbina allows us to use the simplex algorithm in a safe way.

7.1. Computing safe coefficients.

a) Approximation of $L1$. The linear constraint $L1(y, \alpha) \equiv y - 2\alpha x + \alpha^2 \geq 0$ approximates a term x^2 with $\alpha \in [\underline{x}, \overline{x}]$. $L1(y, \alpha)$ corresponds to the tangent lines to the curve $y = x^2$ at the point (α, α^2).

Thus, the computation over the floats of the coefficients of $L1(y, \alpha)$ may change the slope of the tangent line 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; thus 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:

$$L1_{\mathbb{F}}(y, \alpha) \; \equiv \; \begin{cases} y - \bigtriangledown(2\alpha)x + \bigtriangleup(\alpha^2) \geq 0, & \text{if } \alpha \geq 0 \\ y - \bigtriangleup(2\alpha)x + \bigtriangleup(\alpha^2) \geq 0, & \text{if } \alpha < 0 \end{cases}$$

where $\bigtriangledown(x)$ (resp. $\bigtriangleup(x)$) denotes a rounding mode of x towards $-\infty$ (resp. $+\infty$).

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

$$L2_{\mathbb{F}} \; \equiv \; \begin{cases} \bigtriangleup(\underline{x} + \overline{x})x - y - \bigtriangledown(\underline{x}\overline{x}) \geq 0, & \text{if } \underline{x} \geq 0 \\ \bigtriangledown(\underline{x} + \overline{x})x - y - \bigtriangledown(\underline{x}\overline{x}) \geq 0, & \text{if } \overline{x} < 0 \\ \bigtriangleup(\underline{x} + \overline{x})x - y - \bigtriangledown(\underline{x}\overline{x} + Ulp(\bigtriangleup(\underline{x} + \overline{x}))\underline{x}) \geq 0, & \text{if } \overline{x} > 0,\, \underline{x} < 0,\, |\underline{x}| \leq |\overline{x}| \\ \bigtriangledown(\underline{x} + \overline{x})x - y - \bigtriangledown(\underline{x}\overline{x} - \bigtriangleup(Ulp(\bigtriangleup(\underline{x} + \overline{x}))\overline{x})) \geq 0, & \text{if } \overline{x} > 0,\, \underline{x} < 0,\, |\underline{x}| > |\overline{x}| \end{cases}$$

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

c) Approximation of $BIL1, BIL2, BIL3$ *and* $BIL4$. The general form of $BIL1, BIL2, BIL3$ and $BIL4$ 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 results from a computation: all the other terms use constants which are not subject to round off errors. 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 $BIL\{1..4\}_{\mathbb{F}} \equiv Y + \bigtriangleup(s_3 b_1 b_2) \geq 0$.

d) Approximation of multivariate linearisations. We are now in position to introduce the corrections of multivariate linearisations as introduced for the power of x. Such linearisations could be rewritten in the following form :

$$\sum_{i=1}^{n} a_i x_i + b \geq 0$$

where a_i denotes the expression used to compute the coefficient of variable x_i and b the expression used to compute the constant value. Proposition 7.1 takes advantage of interval arithmetic to compute a safe linearisation with coefficients over the floating point numbers:

PROPOSITION 7.1.

$$\sum_{i=1}^{n} \overline{a}_i x_i + \sup(\overline{b} + \sum_{i=1}^{n} \sup(\sup(\mathbf{a}_i \underline{x}_i) - \overline{a}_i \underline{x}_i)) \geq \sum_{i=1}^{n} a_i x_i + b \geq 0, \forall x_i \in \mathbf{x}_i.$$

Proof. $\forall x_i \in \mathbf{x}_i$, we have

$$\sum_{i=1}^{n} \overline{a}_i x_i + \sup(\overline{b} + \sum_{i=1}^{n} \sup(\sup(\mathbf{a}_i \underline{x}_i) - \overline{a}_i \underline{x}_i)) \geq \sum_{i=1}^{n} \overline{a}_i x_i + b + \sum_{i=1}^{n} (\sup(\mathbf{a}_i \underline{x}_i) - \overline{a}_i \underline{x}_i)$$

and

$$\sum_{i=1}^{n} \overline{a}_i x_i + b + \sum_{i=1}^{n} (\sup(\mathbf{a}_i \underline{x}_i) - \overline{a}_i \underline{x}_i) = \sum_{i=1}^{n} (\overline{a}_i (x_i - \underline{x}_i) + \sup(\mathbf{a}_i \underline{x}_i)) + b$$

As $\forall i \in \{1, ..., n\}$, we have $\overline{a}_i \geq a_i$, $\sup(\mathbf{a}_i \underline{x}_i) \geq a_i \underline{x}_i$, and $\forall x_i \in \mathbf{x}_i$, $x_i - \underline{x}_i \geq 0$. Therefore :

$$\sum_{i=1}^{n} (\overline{a}_i (x_i - \underline{x}_i) + \sup(\mathbf{a}_i \underline{x}_i)) + b \geq \sum_{i=1}^{n} (a_i (x_i - \underline{x}_i) + a_i \underline{x}_i) + b = \sum_{i=1}^{n} a_i x_i + b$$

$\square$

This proposition provides a safe approximation of a multivariate linearisation which holds for any a_i, x_i and b. This result could be refine by means of the previous approximations. For instance, whenever $\underline{x}_i \geq 0$, $\overline{a}_i x_i \geq a_i x_i$. In this case, there is no need for an additional correction.

e) Approximation of initial constant values. Initial constant values are real numbers that may not have an exact representation within the set of floating point numbers. Thus, a safe approximation is required.

Constant values in inequalities have to be correctly rounded according to the orientation of the inequality. The result presented in the previous paragraph sets the rounding directions which have to be used.

Equations must be transformed into inequalities when their constant values have to be approximated.

7.2. Computation of safe bounds with linear programming algorithm. Linear programming methods can solve problems of the following form:

$$\begin{array}{rl} \min & C^T X \\ \text{such that} & \underline{B} \leq AX \leq \overline{B} \\ \text{and} & \underline{X} \leq X \leq \overline{X} \end{array} \tag{7.1}$$

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

Neumaier and Shcherbina [42] provide a cheap method to obtain a rigorous bound of the objective and certificates of infeasibility. The essential observation is that the dual of (7.1) is

$$\begin{array}{rl} \max & \underline{B}^T Z' + \overline{B}^T Z'' \\ \text{such that} & A^T (Z' - Z'') = C \end{array} \tag{7.2}$$

Let $Y = Z' - Z''$, and the residue $R = A^T Y - C \in \mathbf{R} = [\underline{R}, \overline{R}]$. It follows that

$$C^T X = (A^T Y - R)^T X = Y^T AX - R^T X \in Y^T [\underline{B}, \overline{B}] - \mathbf{R}^T [\underline{X}, \overline{X}]$$

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

$$\mu = \inf(Y^T \mathbf{B} - \mathbf{R}^T \mathbf{X}) = \bigtriangledown(Y^T \mathbf{B} - \mathbf{R}^T \mathbf{X}) \tag{7.3}$$

Formula (7.3) is trivially correct by construction. Note that the precision of such a safe bound depends on the width of the intervals $[\underline{X}, \overline{X}]$.

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

However, the linear program (7.1) may be infeasible. In that case, Neumaier and Shcherbina show that whenever $d = \inf(\mathbf{R'}^T \mathbf{X} - Y^T \mathbf{B}) > 0$ where $R' = A^T Y \in \mathbf{R}'$, then it is certain that no feasible point exist. However, the precision of interval arithmetic does not always allow to get a positive value for d while the linear program is actually infeasible. In the latter case, we consider it as feasible. Note that box-consistency may be able to reject most, if not all, of the domains of such variables.

8. Experimental results.

This section reports experimental results of `Quad` on a variety of twenty standard benchmarks. Benchmarks `eco6`, `katsura5`, `katsura6`, `katsura7`, `tangets2`, `ipp`, `assur44`, `cyclic5`, `tangents0`, `chemequ`, `noon5`, `geneig`, `kinema`, `reimer5` and `camera1s` were taken from Verschelde's web site[11], `kin2` from [60], `didrit` from [15], `lee` from [29], and finally `yama194`, `yama195` and `yama196` from [63]. The most challenging benchmark is `stewgou40` [16]. It describes the 40 possible positions of a Gough-Stewart platform as a function of the values of the actuators. The proposed modelling of this problem consists of 9 equations with 9 variables.

The experimental results are reported in Tables 8.1 and 8.2. Column n (resp. δ) shows the number of variables (resp. the maximum polynomial degree). `BP(Φ)` stands for a *Branch and Prune* solver based on the Φ filtering algorithm, that is to say, a search-tree exploration where a filtering technique Φ is applied at each node. `quad(H)` denotes the `Quad` algorithm where bilinear terms are relaxed with formulae (6.2), power terms with formulae (6.5) and product terms with the quadrification method; `H` stands for the heuristic used for decomposing terms in the quadrification process.

The performances of the following five solvers have been investigated:

1. `RealPaver`: a free *Branch and Prune* solver[12] that dynamically combines optimised implementations of box–consistency filtering and 2b–consistency filtering algorithms [5]
2. `BP(box)`: a *Branch and Prune* solver based on the ILOG[13] commercial implementation of box–consistency
3. `BP(box+simplex)`: a *Branch and Prune* solver based on `box` and a simple linearisation of the whole system without introducing linear relaxations of the nonlinear terms
4. `BP(box+quad(Qmid))`: a *Branch and Prune* solver which combines `box` and the `Quad` algorithm where product terms are relaxed with the `Qmid` heuristic
5. `BP(box+quad(rAI))`: a *Branch and Prune* solver which combines `box` and the `Quad` algorithm where product terms are relaxed with the `rAI` heuristic

Note that the `BP(box+simplex)` solver implements a strategy that is slightly different to Yamamura's approach [63].

All the solvers have been parameterised to get solutions or boxes with precision of 10^{-8}. That is to say, the width of the computed intervals is smaller than 10^{-8}. A solution is said to be *safe*

[11]The database of polynomial systems available at http://www.math.uic.edu/~jan/Demo/
[12]See http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver/main.html
[13]See http://www.ilog.com/products/jsolver

			`BP(box+quad(Qmid))`			`BP(box)`			*Realpaver*	
Name	n	δ	*Sols*	*Ksplits*	$T(s)$	*Sols*	*Ksplits*	$T(s)$	*Sols*	$T(s)$
cyclic5	5	5	10(10)	0.6	45.8	10(10)	13.4	26.3	10	291.6
eco6	6	3	4(4)	0.4	15.3	4(4)	1.7	3.7	4	1.3
assur44	8	3	10(10)	0.1	49.5	10(10)	15.8	72.5	10	72.6
ipp	8	2	10(10)	0.0	5.7	10(10)	4.6	14.0	10	16.8
katsura5	6	2	16(11)	0.1	9.9	41(11)	8.2	12.7	12	6.7
katsura6	7	2	60(24)	0.5	121.9	182(24)	136.6	281.4	32	191.8
kin2	8	2	10(10)	0.0	6.2	10(10)	3.5	19.3	10	2.6
noon5	5	3	11(11)	0.1	17.9	11(11)	50.2	58.7	11	39.0
tangents2	6	2	24(24)	0.1	17.5	24(24)	14.1	27.9	24	16.5
camera1s	6	2	16(16)	1.0	28.9	2(2)	11820.3	–	0	–
didrit	9	2	4(4)	0.1	14.7	4(4)	51.3	132.9	4	94.6
geneig	6	3	10(10)	0.8	39.1	10(10)	290.7	868.6	10	475.6
kinema	9	2	8(8)	0.2	19.9	15(7)	244.0	572.4	8	268.4
katsura7	8	2	58(42)	1.7	686.9	231(42)	1858.5	11104.1	44	4671.1
lee	9	2	4(4)	0.5	43.3	0(0)	8286.3	–	0	–
reimer5	5	6	24(24)	0.1	53.0	24(24)	2230.2	2892.5	24	733.9
stewgou40	9	4	40(40)	1.6	924.0	11(11)	3128.6	–	8	–
yama194	16	3	9(9)	0.0	11.1	9(8)	1842.1	–	0	–
yama195	60	3	3(3)	0.0	106.1	0(0)	19.6	–	0	–
yama196	30	1	2(1)	0.0	6.7	0(0)	816.7	–	0	–

TABLE 8.1
Experimental results: comparing `Quad` *and Constraint solvers*

if we can prove its uniqueness within the considered box. This proof is based on the well known Brouwer fixed point theorem (see [20]) and just requires a single test.
Sols, *Ksplit* and $T(s)$) are respectively the number of solutions, the number of thousands of branchings (or splittings) and the execution time in seconds. The number of solutions is followed with a number of *safe* solutions between brackets. A "-" in the column $T(s)$ means that the solver was unable to find all the solutions within eight hours. All the computations have been performed on a PC with Pentium IV processor at 2.66GHz running Linux. The compiler was GCC 2.9.6 used with the -O6 optimisation flag.

Table 8.1 displays the performances of `RealPaver`, `BP(box)` and `BP(box+quad(Qmid))`. The benchmarks have been grouped into three sets. The first group contains problems where the `QuadSolver` does not behave very well. These problems are quite easy to solve and the overhead of the relaxation and the calls to a linear solver does not pay off. The second group contains a set of benchmarks for which the `QuadSolver` compares well with the two other constraint solvers: the `QuadSolver` requires always much less splitting and often less time than the other solvers. In the third group, which contains difficult problems, the `QuadSolver` outperforms the two other constraint solvers. The latter were unable to solve most of these problems within eight hours whereas the `QuadSolver` managed to find all the solutions for all but two of them in less than 8 minutes. For instance, `BP(box)` requires about 74 hours to find the four solutions of the `Lee` benchmark whereas the `QuadSolver` managed to do the job in a couple of minutes. Likewise, the `QuadSolver` did find the forty safe solutions of the `stewgou40` benchmark in about 15 minutes whereas `BP(box)` required about 400 hours. The essential observation is that the `QuadSolver` spends more time in the filtering step but it performs much less splitting than classical solvers. This strategy pays off for difficult problems.

	`BP(box+simplex)`			`BP(box+quad(Qmid))`			`BP(box+quad(rAI))`		
Name	*Sols*	*Ksplits*	*T(s)*	*Sols*	*Ksplits*	*T(s)*	*Sols*	*Ksplits*	*T(s)*
cyclic5	10(10)	15.6	60.6	10(10)	0.6	45.8	10(10)	0.8	76.1
eco6	4(4)	1.1	7.2	4(4)	0.4	15.3	4(4)	0.4	15.3
assur44	10(10)	15.5	261.9	10(10)	0.1	49.5	10(10)	0.1	50.0
ipp	10(10)	3.2	39.7	10(10)	0.0	5.7	10(10)	0.0	5.7
katsura5	41(11)	7.7	47.8	16(11)	0.1	9.9	16(11)	0.1	9.9
katsura6	182(24)	135.2	1156.7	60(24)	0.5	121.9	60(24)	0.5	122.7
kin2	10(10)	3.4	42.5	10(10)	0.0	6.2	10(10)	0.0	6.2
noon5	11(11)	49.6	226.7	11(11)	0.1	17.9	11(11)	0.1	17.8
tangents2	24(24)	11.4	77.7	24(24)	0.1	17.5	24(24)	0.1	17.5
camera1s	4(4)	3298.6	–	16(16)	1.0	28.9	16(16)	1.0	29.9
didrit	4(4)	5.3	93.2	4(4)	0.1	14.7	4(4)	0.1	14.7
geneig	10(10)	202.8	2036.8	10(10)	0.8	39.1	10(10)	0.8	39.2
kinema	13(7)	87.0	1135.1	8(8)	0.2	19.9	8(8)	0.2	20.0
katsura7	231(42)	1867.2	21679.6	58(42)	1.7	686.9	58(42)	1.7	684.0
lee	2(2)	78.1	1791.8	2(2)	0.3	27.1	2(2)	0.3	26.5
lee2	4(4)	117.6	2687.2	4(4)	0.5	43.3	4(4)	0.5	43.3
reimer5	24(24)	2208.7	10433.5	24(24)	0.1	53.0	24(24)	0.1	53.1
stewgou40	13(13)	716.3	–	40(40)	1.6	924.0	40(40)	1.5	914.1
yama194	9(7)	442.0	–	9(9)	0.0	11.1	9(9)	0.0	11.2
yama195	3(2)	0.0	37.7	3(3)	0.0	106.1	3(3)	0.0	106.7
yama196	2(1)	0.0	6.6	2(1)	0.0	6.7	2(1)	0.0	6.7

TABLE 8.2
Experimental results: comparing `Quad` *based on different relaxations*

All the problems, except `cyclic5` and `reimer5`, contain many quadratic terms and some product and power terms. `cyclic5` is a pure multi-linear problem that contains only sums of products of variables. The `Quad` algorithm has not been very efficient for handling this problem. Of course, one could not expect an outstanding performance on this bench since product term relaxation is a poor approximation of multi-linear terms. `reimer5` is a pure power problem of degree 6 that has been well solved by the `Quad` algorithm.

Table 8.2 displays the performances of solvers combining box–consistency and three different relaxation techniques. There is no significant difference between the solver based on the `Qmid` heuristics and the solver based on the `rAI` heuristics. Indeed, both heuristics provide convex and concave envelopes of the product terms. The `QuadSolver` with relaxations outperforms the `BP(box+simplex)` approach for all benchmarks but `yama195`, which is a quasi-linear problem. These performances on difficult problems illustrate well the capabilities of the relaxations.

Note that Verschelde's homotopy continuation system, `PHCpack` [62], required 115s to solve `lee` and 1047s to solve `stewgou40` on our computer. `PHCpack` is a state of the art system in the solving of polynomial systems of equations. Unfortunately, it is limited to polynomial systems and does not handle inequalities. `PHCpack` searches for all the roots of the equations, whether real or complex, and it does not restrict its search to a given subspace. The homotopy continuation approach also suffers from an exponential growing computation time which depends on the number of nonlinear terms (`PHCpack` failed to solve `yama195` which contains 3600 nonlinear terms). In contrast to homotopy continuation methods, `QuadSolver` can easily be extended to non-polynomial systems.

Thanks to Arnold Neumaier and Oleg Shcherbina, we had the opportunity to test BARON [50] with some of our benchmarks. `QuadSolver` compares well with this system. For example, BARON

6.0[14] and `QuadSolver` require more or less the same time to solve `camera1s`, `didrit`, `kinema` and `lee`. BARON needs only $1.59s$ to find all the solutions of `yama196` but it requires $859.6s$ to solve `yama195`. Moreover, BARON loses some solutions on `reimer5` (22 solutions found) and `stewgou40` (14 solutions found) whereas it generates numerous wrong solutions for these two problems. We must also underline that BARON is a global optimisation problem solver and that it has not been built to find all the solutions of a problem.

9. Conclusion.

In this paper, we have exploited an RLT schema to take into account specific semantics of nonlinear terms. This relaxation process is incorporated in the *Branch and Prune* process [60] that exploits interval analysis and constraint satisfaction techniques to find rigorously all solutions in a given box. The reported experimental results show that this approach outperforms the classical constraint solvers.

G. Pesant and M. Boyer [44, 45] first introduced linear relaxations in a CLP language to handle geometrical constraints. However, the approximation of the constraints was rather weak. The approach introduced in this paper is also related to recent work that has been done in the interval analysis community as well as to some work achieved in the optimisation community.

In the interval analysis community, Yamamura et al. [63] used a simple linear relaxation procedure where nonlinear terms are replaced by new variables to prove that some box does not contain solutions. No convex/concave outer-estimations are proposed to obtain a better approximation of the nonlinear terms. As pointed out by Yamamura, this approach is well adapted to quasi-linear problems: "*This test is much more powerful than the conventional test if the system of nonlinear equations consists of many linear terms and a relatively small number of nonlinear terms*" [63].

The global optimisation community worked also on solving nonlinear equation problems by transforming them into an optimisation problem (see for example chapter 23 in [17]). The optimisation approach has the capability to take into account specific semantics of nonlinear terms by generating a tight outer-estimation of these terms. The pure optimisation methods are usually not rigorous since they do not take into account rounding errors and do not prove the uniqueness of the solutions found.

Acknowledgements. We thank Arnold Neumaier for his fruitful comments on an early version of this paper. We are also grateful to Arnold Neumaier and Oleg Shcherbina for their help in testing BARON.

REFERENCES

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

[2] E. Allgower and K. Georg. *Numerical continuation methods : an introduction.* Springer Verlag, 1990.

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

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

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

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

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

[14] The tests was performed on an Athlon XP 1800 computer.

[9] R. L. Burden and J. D. Faires. *Numerical Analysis.* PWS-KENT, Boston, MA, 5^{th} edition, 1993.

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

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

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

[13] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Springer-Verlag, New York, 2nd edition, 1997.

[14] Y. Crama. Recognition problems for polynomial in 0-1 variables. *Mathematical Programming*, pages 44:139–155, 1989.

[15] 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.

[16] P. 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.

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

[18] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[19] E. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *Bit*, 21:203–221, 1981.

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

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

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

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

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

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

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

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

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

[29] 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.

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

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

[32] G.P. McCormick. Computability of global solutions to factorable nonconvex programs – part i – convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[33] C. Michel, Y. Lebbah, and M. Rueher. Safe embedding of the simplex algorithm in a CSP framework. In *Proc. of 5th Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems CPAIOR 2003, CRT, Université de Montréal*, pages 210–220, 2003.

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

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

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

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

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

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

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

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

[42] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *in Math. Programming A., DOI 10.1007/s10107-003-0433-3 (also as http://www.mat.univie.ac.at/∼neum/papers.html#mip)*, 2003.

[43] 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.

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

[45] G. Pesant and M. Boyer. Reasonning about solids using constraint logic programming. *Journal of Automated Reasoning*, 22:241–262, 1999.

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

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

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

[49] H.S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, pages 19:403–424, 2001.

[50] N. V. Sahinidis and M. Twarmalani. Baron 5.0 : Global optimisation of mixed-integer nonlinear programs. Technical report, University of Illinois at Urbana-Champaign, Department of Chemical and Biomolecular Engeneering, 2002.

[51] N. V. Sahinidis and M. Twarmalani. Global optimization of mixed-integer programs : A theoretical and computational study. *Mathematical Programming*, page Forthcoming, 2003.

[52] H.D. Sherali. Convex envelopes of multilinear functions over a unit hypercube and over special discrete sets. *Acta mathematica vietnamica*, pages 22(1):245–270, 1997.

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

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

[55] H.D. Sherali and C.H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Journal of Global Optimization*, pages 7, 1–31, 1995.

[56] H.D. Sherali and C.H. Tuncbilek. A comparison of two reformulation-linearization technique based on linear programming relaxations for polynomial porgramming problems. *Journal of Global Optimization*, pages 10:381–390, 1997.

[57] H.D. Sherali and C.H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, pages 21:1–9, 1997.

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

[59] M. Tawarmalani and N. V. Sahinidis, editors. *Convexification and Global Optimization in Continuous and Mixed-Integer Non-Linear Programming.* Kluwer Academic Publishers, 2002.

[60] 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.

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

[62] J. Verschelde. Algorithm 795: Phcpack: a general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software (TOMS)*, 25(2):251–276, june 1999.

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