A Reliable Affine Relaxation Method for Global Optimization

Jordan Ninin† Frédéric Messine‡ Pierre Hansen§¶

October 23, 2012

Abstract

An automatic method for constructing linear relaxations of constrained global optimization problems is proposed. Such a construction is based on affine and interval arithmetics and uses operator overloading. These linear programs have exactly the same numbers of variables and of inequality constraints as the given problems. Each equality constraint is replaced by two inequalities. This new procedure for computing reliable bounds and certificates of infeasibility is inserted into a classical Branch and Bound algorithm based on interval analysis. Extensive computation experiments were made on a sample of 74 problems from the COCONUT database with up to 24 variables or 17 constraints; 61 of these were solved, and 30 of them for the first time, with a guaranteed upper bound on the relative error equal to 10^{-8}. Moreover, this sample comprises 39 examples to which the GlobSol algorithm was recently applied finding reliable solutions in 32 cases. The proposed method allows solving 31 of these, and 5 more with a CPU-time not exceeding 2 minutes.

1 Introduction

For about thirty years, interval Branch and Bound algorithms are increasingly used to solve global optimization problems in a deterministic way [12, 15, 23, 36]. Such algorithms are *reliable*, i.e., they provide an optimal solution and its value with a guaranteed bound on the error, or a proof that the problem under study is infeasible. Other approaches of global optimization (e.g. [1, 2, 13, 20, 21, 22, 29, 37, 39]), while useful and often less time-consuming than interval methods, do not provide such a guarantee. Recently, the second author adapted and improved standard interval branch and bound algorithms to solve design problems of electromechanical actuators [8, 9, 26, 27]. This work showed that interval propagation techniques based on constructions of computation trees [25, 40, 41] and on linearization techniques [12, 15, 21] improved considerably the speed of convergence of the algorithm.

Another way to solve global optimization problems, initially outside of the interval branch and bound framework, is the Reformulation-Linearization Technique developed by Adams and

†OSM, IHSEV, Lab-STICC, ENSTA-Bretagne, 2 rue F. Verny, 29806 Brest, France, `jordan.ninin@ensta-bretagne.fr`

‡ENSEEIHT-IRIT, 2 rue Charles Camichel, 31071, Toulouse, France, `Frederic.Messine@enseeiht.fr`

§Groupe d'Études et de Recherches en Analyse des Décisions, HEC Montréal, Canada and LIX, École Polytechnique, Palaiseau, France, `Pierre.Hansen@gerad.ca`

¶The research of Pierre Hansen was supported by an NSERC Operating Grant as well as by the DIGITEO Fondation. The work of Jordan Ninin has been supported by French National Research Agency (ANR) through COSINUS program (project ID4CS nANR-09-COSI-005).

Sherali [37], see also [2, 35] for methods dedicated to quadratic non-convex problems. The main idea is to reformulate a global optimization problem as a larger linear one, by adding new variables as powers or products of the given variables and linear constraints on their values.

Kearfott [16], Kearfott and Hongthong [18] and Lebbah, Rueher and Michel [21] embedded this technique in interval branch and bound algorithms, showing their efficiency on some numerical examples. However, it is not uncommon that the relaxed linear programs are time-consuming to solve exactly at each iteration owing to their large size. Indeed, if the problem has highly nonlinear terms, fractional exponents or many quadratic terms, these methods will require many new variables and constraints.

In this paper, the main idea is to use affine arithmetic [4, 5, 6, 38] which can be considered as an extension of the classical interval arithmetic [30] obtained by converting intervals into affine forms and performing the computations using definitions of an proper arithmetic such that the results remain affine forms. This has several advantages: (i) keeping affine information on the dependencies among the variables during the computations reduces the dependency problem which occurs when the same variable has many occurrences in the expression of a function; (ii) as for interval arithmetic, affine arithmetic can be implemented in an automated way by using computation trees and operator overloading [29] which are available in some languages such as C++, Fortran90/95/2000 and Java; (iii) the linear programs have exactly the same numbers of variables and of inequality constraints as the given constrained global optimization problem. The equality constraints are replaced by two inequality ones. This is due to the use of two affine forms introduced by the second author in a previous work [24]. The linear relaxations have to be solved by specialized codes such as C-PLEX. Techniques for obtaining reliable results with such non reliable codes have been proposed by Neumaier and Shcherbina [32], and are used in some of the algorithms proposed below; (iv) compared with previous, often specialized, works on the interval branch and bound approach [16, 21], or which could be embedded into such an approach [2, 37], the proposed method is fairly general and can deal with many usual functions such as logarithm, exponential, inverse, square root, etc.

The paper is organized as follows. Section 2 specifies notations and recalls basic definitions about affine arithmetic and affine forms. Section 3 is dedicated to the proposed reformulation methods and their properties. Section 4 describes the reliable version of these methods. In Section 5, their embedding in an interval Branch and Bound algorithm is discussed. Section 6 validates the efficiency of this approach by performing extensive numerical experiments on a sample of 74 test problems from the COCONUT website. Section 7 concludes.

2 Affine Arithmetic and Affine Forms

Interval arithmetic, developed by Moore [30], extends usual functions of arithmetic to intervals. The set of intervals will be denoted by $\mathbb{I}$, and the set of n-dimensional interval vectors, also called *boxes*, will be denoted by $\mathbb{I}^n$. The four standard operations of arithmetic are defined by the following equations, where $\mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ and $\mathbf{y} = [\underline{\mathbf{y}}, \overline{\mathbf{y}}]$ are intervals:

$$\begin{array}{llll} \mathbf{x}+\mathbf{y} & = [\underline{\mathbf{x}}+\underline{\mathbf{y}}, \overline{\mathbf{x}}+\overline{\mathbf{y}}], & \mathbf{x}\times\mathbf{y} & = [\min(\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}}), \max(\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}})], \\ \mathbf{x}-\mathbf{y} & = [\underline{\mathbf{x}}-\overline{\mathbf{y}}, \overline{\mathbf{x}}-\underline{\mathbf{y}}], & \mathbf{x}\ /\ \mathbf{y} & = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \times [1/\overline{\mathbf{y}}, 1/\underline{\mathbf{y}}], \text{ if } 0 \notin \mathbf{y}. \end{array}$$

These operators are the basis of interval arithmetic, and its principle can be extended to many unary functions, such as cos, sin, exp, log, $\sqrt{\ }$, etc. [15, 38].

We denote the middle of the interval $\mathbf{x}$ by $\mathrm{mid}(\mathbf{x}) = \frac{\underline{\mathbf{x}}+\overline{\mathbf{x}}}{2}$.

If $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{I}^n$, $\mathrm{mid}(\mathbf{X}) = (\frac{\underline{\mathbf{x}_1}+\overline{\mathbf{x}_1}}{2}, \frac{\underline{\mathbf{x}_2}+\overline{\mathbf{x}_2}}{2}, \ldots, \frac{\underline{\mathbf{x}_n}+\overline{\mathbf{x}_n}}{2})$.

Given a function f of one or several variables $x_1, \dots, x_n$ and the corresponding intervals for the variables $\mathbf{x}_1, \dots, \mathbf{x}_n$, the *natural interval extension* F of f is an interval obtained by substituting variables by their corresponding intervals and applying the interval arithmetic operations. This provides an *inclusion function*, i.e., $F : \mathbb{D} \subseteq \mathbb{I}^n \to \mathbb{I}$ such that $\forall \mathbf{X} \in \mathbb{D}, \{f(x) : \forall x \in \mathbf{X}\} \subseteq F(\mathbf{X})$, for details see [36, section 2.6].

Example 2.1 *Using the rounded interval arithmetic in Fortran double precision, such as define in [30, Chap. 3],*

$$\mathbf{X} = [1,2] \times [2,6], \quad f(x) = x_1 \times x_2^2 - \exp(x_1 + x_2),$$
$$F(\mathbf{X}) = [1,2] \times [2,6]^2 - \exp([1,2] + [2,6]) = [-2976.957987041728, 51.914463076812],$$
$$\textit{We obtain that: } \forall x \in \mathbf{X}, f(x) \in [-2976.957987041728, 51.914463076812].$$

Affine arithmetic was introduced in 1993 by Comba and Stolfi [4] and developed by De Figueiredo and Stolfi in [5, 6, 38]. This technique is an extension of interval arithmetic obtained by replacing *intervals* with *affine forms*. The main idea is to keep linear dependency information during the computations. This makes it possible to efficiently deal with a difficulty of interval arithmetic: the *dependency problem*, which occurs when the same variable appears several times in an expression of a function (each occurrence of the same variable is treated as an independent variable). To illustrate, the natural interval extension of $f(x) = x - x$, where $x \in \mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$, is equal to $[\underline{\mathbf{x}} - \overline{\mathbf{x}}, \overline{\mathbf{x}} - \underline{\mathbf{x}}]$ instead of 0.

A standard affine form is written as follows, where x is a partially unknown quantity, the coefficients x_i are finite floating-point numbers (we note this with a slight abuse of notation $\mathbb{R}$) and ϵ_i are real variables whose values are unknown but lie in $[-1,1]$ [38]:

$$\widehat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i, \qquad \text{with } \forall i \in \{0,1,\dots,n\}, x_i \in \mathbb{R} \text{ and } \forall i \in \{1,2,\dots,n\}, \epsilon_i = [-1,1]. \tag{1}$$

As in interval arithmetic, usual operations and functions are extended to deal with affine forms. For example, the addition between two affine forms, latter denoted by $\widehat{x}$ and $\widehat{y}$, is simply the term-wise addition of their coefficients x_i and y_i. The algorithm for the other operations and some transcendental functions, such as the square root, the logarithm, the inverse and the exponential, can be found in [38]. Conversions between affine forms and intervals are done as follows:

Interval $\longrightarrow$ Affine Form

$$\begin{aligned} \mathbf{x} = &\ [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \longrightarrow \\ \widehat{x} = &\ \frac{\underline{\mathbf{x}} + \overline{\mathbf{x}}}{2} + \frac{\underline{\mathbf{x}} - \overline{\mathbf{x}}}{2}\epsilon_k, \end{aligned} \tag{2}$$
where ϵ_k is a new variable.

Affine Form $\longrightarrow$ Interval

$$\begin{aligned} \widehat{x} = &\ x_0 + \textstyle\sum_{i=1}^{n} x_i \epsilon_i \longrightarrow \\ \mathbf{x} = &\ x_0 + \left(\sum_{i=1}^{n} |x_i|\right) \times [-1,1]. \end{aligned} \tag{3}$$

Indeed, using these conversions, it is possible to construct an *affine inclusion function*; all the intervals are converted into affine forms, the computations are performed using affine arithmetic and the resulting affine form is then converted into an interval; this generates bounds on values of a function over a box [24, 38]. These inclusion functions cannot be proved to be equivalent or better than natural interval extensions. However, empirical studies done by De Figueiredo et al. [38], Messine [24] and Messine and Touhami [28] show that when applied to global optimization problems, affine arithmetic is, in general, significantly more efficient for computing bounds than the direct use of interval arithmetic.

Nevertheless, standard affine arithmetic such as described in [38] introduces a new variable each time a non-affine operation is done. Thus, the size of the affine forms is not fixed and its growth may slow down the solution process. To cope with this problem, one of the authors proposed two extensions of the standard affine form which are denoted by AF1 and AF2 [24]. These extended affine forms make it possible to fix the number of variables and to keep track of errors generated by approximation of non-affine operations or functions.

- The first form AF1 is based on the same principle as the standard affine arithmetic but all the new symbolic terms generated by approximations are added up in a single term. Therefore the number of variables does not increase. Thus:

$$\widehat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1}\epsilon_{\pm}, \tag{4}$$

  with $\forall i \in \{0, 1, \ldots, n\}, x_i \in \mathbb{R}$, $x_{n+1} \in \mathbb{R}^+$, $\epsilon_i = [-1, 1]$ and $\epsilon_{\pm} = [-1, 1]$.

- The second form AF2 is based on AF1. Again the number of variables is fixed but the errors are stacked in three terms, separating the positive, negative and unsigned errors. Thus:

$$\widehat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1}\epsilon_{\pm} + x_{n+2}\epsilon_{+} + x_{n+3}\epsilon_{-}, \tag{5}$$

  with $\forall i \in \{0, 1, \ldots, n\}, x_i \in \mathbb{R}$ and $\forall i \in \{1, 2, \ldots, n\}$, $\epsilon_i = [-1, 1]$, and $(x_{n+1}, x_{n+2}, x_{n+3}) \in \mathbb{R}^3_+$, $\epsilon_{\pm} = [-1, 1], \epsilon_{+} = [0, 1], \epsilon_{-} = [-1, 0]$.

In this paper, we use mainly the affine form AF2. Note that a small mistake was recently found in the computation of the error of the multiplication between two AF2 forms in [24], see [41].

Usual operations and functions are defined by extension of affine arithmetic, see [24] for details. For example, the multiplication between two affine forms of type AF1 is performed as follows:

$$\widehat{x} \times \widehat{y} = x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\epsilon_i + \left(x_0 y_{n+1} + x_{n+1} y_0 + \left(\sum_{i=1}^{n+1} |x_i| \times \sum_{i=1}^{n+1} |y_i| \right) \right) \epsilon_{\pm}.$$

For computing unary functions in affine arithmetic, De Figueiredo and Stolfi [38] proposed two linear approximations: the Chebyshev and the min-range approximations, see Figure 1. The Chebyshev approximation is the reformulation which minimizes the maximal absolute error. The min-range approximation is that one which minimizes the range of the approximation. This affine approximation is denoted as follows:

$$\begin{array}{c} \widehat{f}(\widehat{x}) = \zeta + \alpha\widehat{x} + \delta\epsilon_{\pm}, \\ \text{with } \widehat{x} \text{ given by Equation (4) or (5) and } \zeta \in \mathbb{R}, \alpha \in \mathbb{R}^+, \delta \in \mathbb{R}^+. \end{array} \tag{6}$$

Thus on the one hand, the Chebyshev linearization gives the affine approximation which minimizes the error δ but the lower bound is worse than the actual minimum of the range, see Figure 1. On the other hand, the min-range linearization is less efficient in estimating linear dependency among variables, while the lower bound is equal to the actual minimum of the range. In our affine arithmetic code, as in De Figueiredo et al.'s one [38], we choose to implement the min-range linearization. Indeed, in experiments with monotonic functions, bounds were found to be better than those calculated by the Chebyshev approximation when basic functions were combined (because the Chebyshev approximations increase the range).

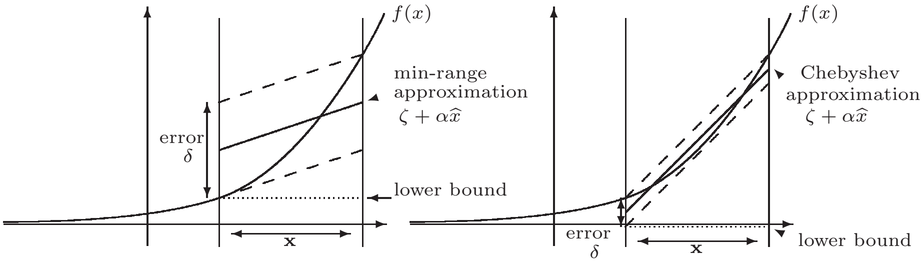

Figure 1: Affine approximations by min-range methods and Chebyshev.

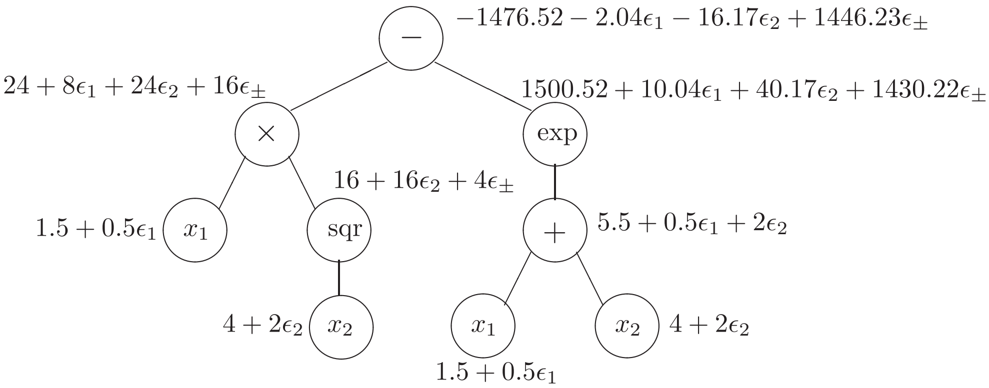

Figure 2: Visualization of AF1 by computation tree: $f(x) = x_1 x_2^2 - \exp(x_1 + x_2)$ in $[1, 2] \times [2, 6]$.

A representation of the computation of AF1 is shown in Figure 2 (numbers are truncated with 6 digits after comma). In our implementation, the computation tree is implicitly built by operator overloading [29]. Hence, its form depends on how the equation is written. The leaves contain constants or variables which are initialized with the affine form generated by the conversion of the initial interval. Then, the affine form of each internal node is computed from the affine form of its sons by applying the corresponding operation of AF1. The root gives the affine form for the entire expression. Lower and upper bounds are obtained by replacing the ϵ variables by $[-1, 1]$ and applying interval arithmetic.

Example 2.2 *Consider the following function:*

$$f(x) = x_1 x_2^2 - \exp(x_1 + x_2) \text{ in } [1, 2] \times [2, 6],$$

First, using Equation (2), we transform the intervals $[1, 2]$ *and* $[2, 6]$ *into the following affine forms (at this stage it is equivalent to use AF1 or AF2):*

$$\mathbf{x}_1 = [1, 2] \rightarrow \widehat{x}_1 = 1.5 + 0.5\epsilon_1 \quad \text{and} \quad \mathbf{x}_2 = [2, 6] \rightarrow \widehat{x}_2 = 4 + 2\epsilon_2.$$

Computing with the extended operators of AF1 and of AF2, we obtain the following affine forms:

$$\widehat{f}_{AF1}(x) = -1476.521761 - 2.042768\epsilon_1 - 16.171073\epsilon_2 + 1446.222382\epsilon_\pm,$$
$$\widehat{f}_{AF2}(x) = -1476.521761 - 2.042768\epsilon_1 - 16.171073\epsilon_2 + 1440.222382\epsilon_\pm + 6\epsilon_+ + 0\epsilon_-.$$

The details of the computation by AF1 are represented in Figure 2. The variable ϵ_1 corresponds to x_1, ϵ_2 to x_2 and using AF1, $\epsilon_\pm$ contains all the errors generated by non-affine operations. Using AF2, $\epsilon_\pm$ contains the errors generated by the multiplication and the exponential, and ϵ_+ the errors generated by the power of 2.

To conclude, using Equation (3), we convert these affine forms into intervals to have the following bounds:
Using directly interval arithmetic, we obtain

$$\forall x \in [1,2] \times [2,6], f(x) \in [-2976.9579870417284, 51.91446307681234],$$

using AF1: $\forall x \in [1,2] \times [2,6], f(x) \in [-2940.9579870417297, -12.085536923186737],$

using AF2: $\forall x \in [1,2] \times [2,6], f(x) \in [-2934.9579870417297, -12.085536923186737],$

and the exact range is

$$\forall x \in [1,2] \times [2,6],\ f(x) \in [-2908.957987041728, -16.085536923187668].$$

In this example, the positive upper bound computed by interval arithmetic introduces an ambiguity on the sign of f over $[1,2] \times [2,6]$ while it is clearly negative. This example shows the interest of AF1 and AF2 and how the corresponding arithmetics are used to compute an enclosure of f over $[1,2] \times [2,6]$.

An empirical comparison among interval arithmetic, AF1 and AF2 affine forms has been done on several randomly generated polynomial functions [24] and the proof of the following proposition is given there.

Proposition 2.3 *Consider a polynomial function f of $\mathbf{X} \subset \mathbb{R}^n$ to $\mathbb{R}$ and f_{AF}, f_{AF1} and f_{AF2} the reformulations of f respectively with AF, AF1 and AF2. Then one has:*

$$\left[\min_{x \in \mathbf{X}} f(x), \max_{x \in \mathbf{X}} f(x)\right] \subseteq \widehat{f}_{AF2}(\mathbf{X}) \subseteq \widehat{f}_{AF1}(\mathbf{X}) = \widehat{f}_{AF}(\mathbf{X}).$$

To compute bounds, the approach using affine arithmetic is completely different from the classical technique using Taylor's expansion which requires the computation of the gradient. In affine arithmetic, an approximation of the gradient is done node by node on the computation tree but the expression of the gradient is not used. This form of computation could be compared to automatic differentiation. The relations between affine arithmetic and interval arithmetic are similar to those between the constraint propagation by computation tree [25] and by linear relaxation [12].

3 Affine Reformulation Technique based on Affine Arithmetic

During many year, reformulation techniques have been used for global optimization [1, 2, 13, 18, 21, 22, 35, 37, 39]. In most cases, the main idea is to approximate a mathematical program by a linear relaxation. Thus, solving this linear program yields bounds on this optimal value or a certificate of infeasibility of the original problem. The originality of our approach lies in how the linear relaxation is made.

In our approach, named *Affine Reformulation Technique* (ART_{AF}), we have kept the computation tree and relied on the extended affine arithmetics (AF1 and AF2). Indeed, the extended affine arithmetics handle affine forms on the computation tree. But until now, this technique has been only used to compute bounds. Now, our approach uses the extended affine arithmetics not only as a simple way to compute bounds but also as a way to linearize automatically every factorable function [39]. This becomes possible by fixing the number of ϵ_i variables in the extended affine arithmetics. Thus, an affine transformation $\mathcal{T}$ between the original set $\mathbf{X} \subset \mathbb{R}^n$ and $\epsilon = [-1,1]^n$ appears, see Equation (2). Now, we can identify the linear part of AF1 and AF2 as a linear approximation of the original function.

Denote the affine form AF1 of f on $\mathbf{X}$ by $\widehat{f}(x)$. Here the components f_i in the formulation depend also on $\mathbf{X}$:

$$\begin{gathered}\widehat{f}(x) = f_0 + \sum_{i=1}^{n} f_i \epsilon_i + f_{n+1}\epsilon_{\pm}, \\ \text{with } \forall i \in \{0,1,\ldots,n\}, f_i \in \mathbb{R}\ ,\ f_{n+1} \in \mathbb{R}^+, \\ \forall i \in \{1,2,\ldots,n\},\ \epsilon_i = [-1,1] \text{ and } \epsilon_{\pm} = [-1,1].\end{gathered}$$

By definition, the affine form AF1 is an inclusion function:

$$\forall x \in \mathbf{X},\ f(x) \in f_0 + \sum_{i=1}^{n} f_i \epsilon_i + f_{n+1}\epsilon_{\pm}.$$

But $\forall y \in [-1,1]^n, \exists x \in \mathbf{X}, y = \mathcal{T}(x)$ where $\mathcal{T}$ is an affine function, then:

$$\begin{gathered}\forall x \in \mathbf{X},\ f(x) \in \left(\sum_{i=1}^{n} f_i \mathcal{T}_i(x_i) + f_0 + f_{n+1}[-1,1] \right), \\ \forall x \in \mathbf{X},\ f(x) - \sum_{i=1}^{n} f_i \mathcal{T}_i(x_i) \in [f_0 - f_{n+1}, f_0 + f_{n+1}], \\ \text{where } \mathcal{T}_i \text{ are the components of } \mathcal{T}.\end{gathered} \tag{7}$$

Thus, this leads to the following propositions:

Proposition 3.1 *Consider $(f_0, \ldots, f_{n+1})$, the reformulation of f on $\mathbf{X}$ using AF1. If $\forall x \in \mathbf{X}, f(x) \leq 0$, then $\forall y \in [-1,1]^n, \sum_{i=1}^n f_i y_i \leq f_{n+1} - f_0$ and if $\forall x \in \mathbf{X}, f(x) = 0$, then $\forall y \in [-1,1]^n$,* $\begin{cases} \sum_{i=1}^n f_i y_i \leq f_{n+1} - f_0, \\ -\sum_{i=1}^n f_i y_i \leq f_{n+1} + f_0. \end{cases}$

Proposition 3.2 *Consider $(f_0, \ldots, f_{n+1}, f_{n+2}, f_{n+3})$, the reformulation of f on $\mathbf{X}$ using AF2. If $\forall x \in \mathbf{X}, f(x) \leq 0$ then $\forall z \in [-1,1]^n, \sum_{i=1}^n f_i z_i \leq f_{n+1} + f_{n+3} - f_0$ and if $\forall x \in \mathbf{X}, f(x) = 0$ then $\forall z \in [-1,1]^n$,* $\begin{cases} \sum_{i=1}^n f_i z_i \leq f_{n+1} + f_{n+3} - f_0, \\ -\sum_{i=1}^n f_i z_i \leq f_{n+1} + f_{n+2} + f_0. \end{cases}$

Proof. If we replace AF1 with AF2 in Equations (7), we have the following inclusion:

$$\forall x \in \mathbf{X},\ f(x) - \sum_{i=1}^{n} f_i \mathcal{T}_i(x_i) \in [f_0 - f_{n+1} - f_{n+3}, f_0 + f_{n+1} + f_{n+2}]. \qquad \square$$

Consider a constrained global optimization problem (8), defined below, a linear approximation of each of the expression for f, g_i and h_j is obtained using AF1 or AF2. Each inequality constraint is relaxed by one linear equation and each equality constraint by two linear equations. Thus, the linear program (9) is automatically generated.

$$\begin{cases} \min\limits_{x \in \mathbf{X} \subset \mathbb{R}^n} & f(x) \\ \text{s.t.} & g_k(x) \leq 0\ , \quad \forall k \in \{1,\ldots,p\}, \\ & h_l(x) = 0\ , \quad \forall l \in \{1,\ldots,q\}. \end{cases} \tag{8}$$

$$\begin{cases} \min\limits_{y \in [-1,1]^n} & c^T y \\ \text{s.t.} & Ay \leq b. \end{cases} \tag{9}$$

Denote by $(F_0, \dots, F_{n+1})$ the resulting affine form AF1 of F such that $\widehat{F}(x) = F_0 + \sum_{i=1}^n F_i \epsilon_i + F_{n+1}\epsilon_\pm$. Then, the linear program (9) is constructed as follows:

$$c = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \qquad A = \begin{pmatrix} (g_1)_1 & \dots & (g_1)_n \\ & \vdots & \\ (g_p)_1 & \dots & (g_p)_n \\ (h_1)_1 & \dots & (h_1)_n \\ -(h_1)_1 & \dots & -(h_1)_n \\ & \vdots & \\ (h_q)_1 & \dots & (h_q)_n \\ -(h_q)_1 & \dots & -(h_q)_n \end{pmatrix}, \qquad b = \begin{pmatrix} (g_1)_{n+1} - (g_1)_0 \\ \vdots \\ (g_p)_{n+1} - (g_p)_0 \\ (h_1)_{n+1} - (h_1)_0 \\ (h_1)_{n+1} + (h_1)_0 \\ \vdots \\ (h_q)_{n+1} - (h_q)_0 \\ (h_q)_{n+1} + (h_q)_0 \end{pmatrix}.$$

Denote by $(F_0, \dots, F_{n+3})$ the resulting affine form AF2 of F such that $\widehat{F}(x) = F_0 + \sum_{i=1}^n F_i \epsilon_i + F_{n+1}\epsilon_\pm + F_{n+2}\epsilon_+ + F_{n+3}\epsilon_-$. Then, the linear program (9) is constructed as follows:

$$c = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \quad A = \begin{pmatrix} (g_k)_1 & \dots & (g_k)_n \\ & \vdots & \\ (h_l)_1 & \dots & (h_l)_n \\ -(h_l)_1 & \dots & -(h_l)_n \\ & \vdots & \end{pmatrix}, \qquad b = \begin{pmatrix} (g_k)_{n+1} + (g_k)_{n+3} - (g_k)_0 \\ \vdots \\ (h_l)_{n+1} + (h_l)_{n+3} - (h_l)_0 \\ (h_l)_{n+1} + (h_l)_{n+2} + (h_l)_0 \\ \vdots \end{pmatrix}.$$

Remark 3.3 *The size of the linear program (9) still remains small. The number of variables is the same and the number of inequality constraints cannot exceed twice the number of constraints of the general problem (8).*

Let us denote by S_1 the set of feasible solutions of the initial problem (8), S_2 the set of feasible solutions of the linear program (9), $\mathcal{T}$ the affine transformation between $\mathbf{X}$ and $[-1,1]^n$, and E_f the lower bound of the error of the affine form of f. Using AF1, $E_f = \underline{f_0 + f_{n+1}\epsilon_\pm} = f_0 - f_{n+1}$ and using AF2, $E_f = \underline{f_0 + f_{n+1}\epsilon_\pm + f_{n+2}\epsilon_+ + f_{n+3}\epsilon_-} = f_0 - f_{n+1} - f_{n+3}$.

Proposition 3.4 *Assume that x is a feasible solution of the original problem (8), hence $y = \mathcal{T}(x)$ is a feasible solution of the linear program (9) and therefore, one has $\mathcal{T}(S_1) \subseteq S_2$.*

Proof. The proposition is a simple consequence of Propositions 3.1 and 3.2. $\square$

Corollary 3.5 *If the relaxed linear program (9) of a problem (8) does not have any feasible solution then the problem (8) does not have any feasible solution.*

Proof. Using directly Proposition 3.4, $S_2 = \emptyset$ implies $S_1 = \emptyset$ and the result follows. $\square$

Proposition 3.6 *If y_{sol} is a solution which minimizes the linear program (9), then*

$$\forall x \in S_1,\ f(x) \geq c^T y_{sol} + E_f,$$

with $E_f = f_0 - f_{n+1}$ if AF1 has been used to generate the linear program (9), and $E_f = f_0 - f_{n+1} - f_{n+3}$ if AF2 has been used.

Proof. Using Proposition 3.4, one has $\forall x \in S_1, y = \mathcal{T}(x) \in S_2$. Moreover, y_{sol} denotes by assumption the solution which minimizes the linear program (9), hence one obtains $\forall y \in S_2, c^T y \geq c^T y_{sol}$. Using Proposition 3.1 and Proposition 3.2, we have $\forall x \in S_1, \exists y \in [-1,1], f(x) - c^T y \geq E_f$ and therefore $\forall x \in S_1, f(x) \geq c^T y_{sol} + E_f$. $\square$

We remark that equality occurs when the problem (8) is linear, because, in this case, AF1 and AF2 are just a rewriting of program (8) on $[-1,1]^n$.

Proposition 3.7 *Let us consider a polynomial program; i.e., f and all g_i, h_j are polynomial functions. Denote a minimizer point of a relaxed linear program (9) using AF1 form by y_{AF1}, and one using AF2 form by y_{AF2}. Moreover, using the notations c_{AF1}, $E_{f_{AF1}}$ and c_{AF2}, $E_{f_{AF2}}$ for the reformulations of f using AF1 and AF2 forms respectively, we have:*

$$\forall x \in S_1, f(x) \geq c_{AF2}^T y_{AF2} + E_{f_{AF2}} \geq c_{AF1}^T y_{AF1} + E_{f_{AF1}}.$$

Proof. By construction of the arithmetics defined in [24] (with corrections as in [41]) and mentioned in Proposition 2.3, if $y \in S_2$, we have

$$c_{AF2}^T y + E_{f_{AF2}} \geq c_{AF1}^T y + E_{f_{AF1}},$$
$$c_{AF2}^T y \geq c_{AF2}^T y_{AF2} \text{ and } c_{AF1}^T y \geq c_{AF1}^T y_{AF1}.$$

But Proposition 3.4 yields $\forall x \in S_1$, $y = \mathcal{T}(x) \in S_2$ and then,

$$\forall x \in S_1, f(x) \geq c_{AF2}^T y_{AF2} + E_{f_{AF2}} \geq c_{AF1}^T y_{AF2} + E_{f_{AF1}} \geq c_{AF1}^T y_{AF1} + E_{f_{AF1}}. \qquad \square$$

Remark 3.8 *Proposition 3.7 could be generalized to factorable functions depending on the definition of transcendental functions in AF1 and AF2 corresponding arithmetic. In [24], only affine operations and the multiplication between two affine forms were taken into account.*

Proposition 3.9 *If a constraint of the problem (8) is proved to be satisfied by interval analysis, then the associated linear constraint can be removed from the linear program (9) and the solution does not change.*

Proof. If a constraint of the original problem (8) is satisfied on $\mathbf{X}$ (which is guaranteed by interval arithmetic based computation), then the corresponding linear constraint is always satisfied for all the values of $y \in [-1, 1]$ and it is not necessary to have it in (9). $\square$

Example 3.10 *Let us consider the following problem:*

$$\begin{cases} \min\limits_{x \in \mathbf{X}=[1,1.5]\times[4.5,5]\times[3.5,4]\times[1,1.5]} & x_3 + (x_1 + x_2 + x_3)x_1x_4 \\ s.t. & c_1(x) = x_1x_2x_3x_4 \geq 25, \\ & c_2(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40, \\ & c_3(x) = 5x_1^4 - 2x_2^3 + 11x_3^2 + 6e^{x_4} \leq 50. \end{cases}$$

First we verify by interval arithmetic, that $C_1(\mathbf{X}) = [15.75, 45.0]$, $C_2(\mathbf{X}) = [34.5, 45.5]$ and $C_3(\mathbf{X}) = [-93.940309, 45.952635]$. $c_3(x)$ is proved to be less than or equal to 50 for all $x \in \mathbf{X}$, thus we do not need to linearize it.

This example is constructed numerically by using the double precision floating point representation. To simplify the notations, the floating point numbers are rounded by rational ones with two decimals. The aim is to illustrate how the technique ART is used.

By using the affine form AF1, the linear reformulations of the above equations provide:

$$\begin{aligned} x_3 + (x_1 + x_2 + x_3)x_1x_4 &\longrightarrow 18.98 + 3.43\epsilon_1 + 0.39\epsilon_2 + 0.64\epsilon_3 + 3.04\epsilon_4 + 1.12\epsilon_\pm, \\ 25 - x_1x_2x_3x_4 &\longrightarrow -2.83 - 5.56\epsilon_1 - 1.46\epsilon_2 - 1.85\epsilon_3 - 5.56\epsilon_4 + 2.71\epsilon_\pm, \\ x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40 &\longrightarrow -0.25 + 0.62\epsilon_1 + 2.37\epsilon_2 + 1.87\epsilon_3 + 0.62\epsilon_4 + 0.25\epsilon_\pm. \end{aligned}$$

We have now to consider the following linear program:

$$\begin{cases} \min\limits_{y \in [-1,1]^4} & 3.43y_1 + 0.39y_2 + 0.64y_3 + 3.04y_4 \\ s.t. & -5.56y_1 - 1.46y_2 - 1.85y_3 - 5.56y_4 \leq 5.54, \\ & 0.62y_1 + 2.37y_2 + 1.87y_3 + 0.62y_4 \leq 0.5, \\ & -0.62y_1 - 2.37y_2 - 1.87y_3 - 0.62y_4 \leq 0. \end{cases}$$

After having solved the linear program, we obtain the following optimal solution:

$$y_{sol} = (-1, -0.24, 1, -0.26), \quad c^T y_{sol} = -3.70, \quad c^T y_{sol} + E_f = 14.15.$$

Hence, using Proposition 3.6, we obtain a lower bound 14.15*. By comparison, the lower bound computed directly with interval arithmetic is* 12.5 *and* 10.34 *using directly only AF1, respectively. This is due to the fact that we do not consider only the objective function to find a lower bound but we use the constraints and the set of feasible solutions as well.*

Remark 3.11 *This section defines a methodology for constructing relaxed linear programs using different affine forms and their corresponding arithmetic evaluations. These results could be extended to the use of other forms such as those defined in [24] and in [28], which are based on quadratic formulations.*

Remark 3.12 *The expression of the linear program (9) depends on the box* $\mathbf{X}$*. Thus, if* $\mathbf{X}$ *changes, the linear program (9) must be generated again to have a better approximation of the original problem (8).*

4 Reliable Affine Reformulation Technique: $rART_{\mathbf{rAF}}$

The methodology explained in the previous section has some interests in itself; (i) the constraints are applied to compute a better lower bound on the objective function, (ii) the size of the linear program is not much larger than the original and (iii) the dependence links among the variables are exploited. But the method is not reliable in the presence of numerical errors due to the approximations provided by using some floating point representations and computations. In the present section, we explain how to make our methodology completely reliable.

First, we need to use a reliable affine arithmetic. The first version of affine arithmetic defined by Comba and Stolfi [4], was not reliable. In [38], De Figueiredo and Stolfi proposed a self-validated version for *standard affine arithmetic*; i.e., all the basic operations are done three times (including the computations of the value of the scalar, the positive and the negative numerical errors). Another possibility is to use the *Reliable Affine Arithmetic* such as defined by Messine and Touhami in [28]. This affine arithmetic replaces all the floating numbers of the affine form by an interval, see Equation (10), where the variables in bold indicate the interval variables.

$$\begin{gathered}\widehat{\mathbf{x}} = \mathbf{x_0} + \sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \\ \text{with } \forall i \in \{0,1,\dots,n\}, \mathbf{x_i} = [\underline{\mathbf{x_i}}, \overline{\mathbf{x_i}}] \in \mathbb{I} \text{ and } \forall i \in \{1,2,\dots,n\}, \epsilon_i = [-1,1].\end{gathered} \tag{10}$$

The conversions between interval arithmetic, affine arithmetic and reliable affine arithmetic a re performed as follows:

Reliable Affine Form $\longrightarrow$ Interval

$$\begin{aligned}\widehat{\mathbf{x}} = &\ \mathbf{x_0} + \textstyle\sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \quad \longrightarrow \\ \mathbf{x} = &\ \mathbf{x_0} + \left(\sum_{i=1}^{n} (\mathbf{x_i} \times [-1,1])\right).\end{aligned}$$

Affine Form $\longrightarrow$ Reliable Affine Form

$$\begin{gathered}\widehat{x} = x_0 + \textstyle\sum_{i=1}^{n} x_i\epsilon_i, \quad \longrightarrow \\ \forall i \in \{1,2,\dots,n\},\ \mathbf{x_i} = x_i, \\ \widehat{\mathbf{x}} = \mathbf{x_0} + \sum_{i=1}^{n} \mathbf{x_i}\epsilon_i.\end{gathered} \tag{11}$$

Reliable Affine Form $\longrightarrow$ Affine Form

$$\begin{aligned}\widehat{\mathbf{x}} = &\ \mathbf{x_0} + \textstyle\sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \quad \longrightarrow \\ \forall i \in &\ \{1,2,\dots,n\}, x_i = \mathrm{mid}(\mathbf{x_i}), \\ \widehat{x} = &\ x_0 + \textstyle\sum_{i=1}^{n} x_i\epsilon_i + \\ &\ \left(\textstyle\sum_{i=1}^{n} \max(\overline{x_i - \mathbf{x_i}}, \overline{\mathbf{x_i} - x_i})\right)\epsilon_{\pm}.\end{aligned}$$

Interval $\longrightarrow$ Reliable Affine Form

$$\begin{aligned} & \mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}], \quad \longrightarrow \\ & \mathbf{x_0} = \mathrm{mid}(\mathbf{x}) \\ \widehat{\mathbf{x}} = &\ \mathbf{x_0} + \max(\overline{\mathbf{x_0} - \mathbf{x}}, \overline{\mathbf{x} - \underline{\mathbf{x_0}}})\epsilon_k, \\ & \text{where } \epsilon_k \text{ is a new variable.}\end{aligned} \tag{12}$$

In this Reliable Affine Arithmetic, all the affine operations are done as for the standard affine arithmetic but using properly rounded interval arithmetic [30] to ensure its reliability. In [28], the multiplication was explicitly given, and the same principle is used in this paper to define other nonlinear operations.

Algorithm 1 is a generalization of the min-range linearization introduced by De Figueiredo and Stolfi in [38], for finding that linearization, which minimizes the range of a monotonous continuous convex or concave function in reliable affine arithmetic. Such as in the algorithm of De Figueiredo and Stolfi, Algorithm 1 consists of finding, in a reliable way, the three scalars α, ζ and δ, see Equation (6) and Figure 1.

Algorithm 1 Min-range linearization of f for reliable affine form on $\widehat{\mathbf{x}}$

1: Set all local variables to be interval variables ($\mathbf{d}, \boldsymbol{\alpha}, \boldsymbol{\zeta}$ and $\boldsymbol{\delta}$),
2: F = natural interval extension of f, F' = natural interval extension of the first derivative of f,
3: $\widehat{\mathbf{x}}$ = the reliable affine form under study, $\mathbf{x}$ = the interval corresponding to $\widehat{\mathbf{x}}$ (see Equation (11)),
4: $\widehat{\mathbf{f}}(\widehat{\mathbf{x}})$ = the reliable affine form of f over $\widehat{\mathbf{x}}$.
5: $\begin{cases} \boldsymbol{\alpha} = \underline{F'(\mathbf{x})}, & \mathbf{d} = \left[\underline{F(\mathbf{x}) - \boldsymbol{\alpha} \times \underline{\mathbf{x}}}, \overline{F(\mathbf{x}) - \boldsymbol{\alpha} \times \overline{\mathbf{x}}}\right], \text{ if } \underline{F'(\mathbf{x})} \geq 0, \\ \boldsymbol{\alpha} = \overline{F'(\mathbf{x})}, & \mathbf{d} = \left[\underline{F(\mathbf{x}) - \boldsymbol{\alpha} \times \overline{\mathbf{x}}}, \overline{F(\mathbf{x}) - \boldsymbol{\alpha} \times \underline{\mathbf{x}}}\right], \text{ if } \overline{F'(\mathbf{x})} \leq 0, \\ \boldsymbol{\alpha} = 0, & \mathbf{d} = \mathbf{x}, \text{ if } f \text{ is constant,} \end{cases}$
6: $\boldsymbol{\zeta} = \text{mid}(\mathbf{d})$,
7: $\boldsymbol{\delta} = \max\left(\overline{\overline{\boldsymbol{\zeta}} - \mathbf{d}}, \overline{\mathbf{d} - \underline{\boldsymbol{\zeta}}}\right)$,
8: $\widehat{\mathbf{f}}(\widehat{\mathbf{x}}) = \boldsymbol{\zeta} + \boldsymbol{\alpha} \times \widehat{\mathbf{x}} + \boldsymbol{\delta}\epsilon_{\pm}$.

Remark 4.1 *Such an arithmetic is a particular case of* the generalized interval arithmetic *introduced by E. Hansen in [11]. This one is equivalent to an affine form with interval coefficients. The multiplication has the same definition as in reliable affine arithmetic. However, the division is not generalizable and the affine information is lost. Furthermore, for nonlinear functions, such as the logarithm, exponential, and square root, nothing is defined in [11]. In our particular case of a reliable affine arithmetic, these difficulties to compute the division and nonlinear functions are avoided.*

Indeed, using the principle of this reliable affine arithmetic, we obtain reliable versions for the affine forms AF1 and AF2, denoted by *rAF1* and *rAF2*. Moreover, as in Section 3, we apply Proposition 3.1 and 3.2 to rAF1 and rAF2 to provide a *reliable affine reformulation* for every factorable function; i.e., we obtain a linear relaxation in which all variables are intervals. Consequently, using the reformulation methodology described in Section 3 for rAF1 or rAF2, we produce automatically a reliable linear program, i.e. all the variables of the linear program (9) are intervals, and the feasibility of a solution x in Proposition 3.4 can exactly be verified.

When the reliable linear program is generated, two approaches can be used to solve it; (i) the first one relies on the use of an interval linear solver such as LURUPA [14, 19] to obtain a reliable lower bound on the objective function or a certificate of infeasibility. Thus, these properties are extended to the general problem (8) using Proposition 3.6 and Corollary 3.5; (ii) the second one is based on generating a reliable linear reformulation of each function of the general problem (8).

Then, we use the conversion between rAF1/AF1 or rAF2/AF2 (see Equation (12)) to obtain a linear reformulation in which all variables are scalar numbers, but, in this case, all numerical errors are taken into account as intervals, and moved to the error variables of the affine form by the conversion. Indeed, we have a linear program which satisfies the conditions of Proposition 3.4 in a reliable way. Then, we use a result from Neumaier and Shcherbina [32] to compute a reliable lower bound of our linear program or a reliable certificate of infeasibility. This method applied to our case yields:

$$\left\{\begin{array}{cl} \min\limits_{\lambda\in\mathbb{R}^m_+, u,l\in\mathbb{R}^n_+} & b^T\lambda + \sum\limits_{i=1}^{n}(l_i+u_i) \\ s.t. & A^T\lambda - l + u = -c. \end{array}\right. \tag{13}$$

The linear program (13) corresponds to the dual formulation of the linear program (9). Let (λ_S, l_S, u_S) be an approximate dual solution given by a linear solver, the bold variables indicate the interval variables and the solution $(\mathbf{\Lambda_S}, \mathbf{L_S}, \mathbf{U_S})$ is the extension of (λ_S, l_S, u_S) into interval arithmetic. This conversion allows performing all the computation by rounded interval arithmetic. Then, we can compute the residual of the dual (13) by interval arithmetic, such as:

$$r \in \mathbf{R} = c + A^T\mathbf{\Lambda_S} - \mathbf{L_S} + \mathbf{U_S}. \tag{14}$$

Hence, using the bounds calculated in [32], we have:

$$\begin{array}{c} \forall y \in S_2,\ c^T y \in \left(\mathbf{R}^T\varepsilon - \mathbf{\Lambda_S}^T[-\infty, b] + \mathbf{L_S}^T\varepsilon - \mathbf{U_S}^T\varepsilon\right), \\ \text{where } \varepsilon = ([-1,1],\ldots,[-1,1])^T \text{ and } [-\infty,b] = ([-\infty,b_1],\ldots,[-\infty,b_m])^T. \end{array} \tag{15}$$

Proposition 4.2 *Let $(\mathbf{\Lambda_S}, \mathbf{L_S}, \mathbf{U_S})$ be the extension to intervals of an approximate solution which minimizes (13) the dual of the linear program (9). Then,*

$$\forall x \in S_1\ ,\ f(x) \geq \underline{\left(\mathbf{R}^T\varepsilon - \mathbf{\Lambda_S}^T[-\infty, b] + \mathbf{L_S}^T\varepsilon - \mathbf{U_S}^T\varepsilon\right) + E_f}.$$

Proof. The result is obtained by applying Equation (14), Equation (15) and Proposition 3.6. □

When the bound cannot be computed, the dual program (13) can be unbounded or infeasible. However, the feasible set of the primal (9) is included in the bounded set $[-1,1]^n$. Thus, the dual (13) cannot be infeasible. Indeed, to prove that the dual (13) is bounded, we look for a feasible solution of the constraint satisfaction problem (16) (it is a well-known method directly adapted from [32]):

$$\left\{\begin{array}{l} b^T\lambda + \sum\limits_{i=1}^{n}(l_i+u_i) \neq 0 \\ A^T\lambda - l + u = 0, \\ \lambda\in\mathbb{R}^m_+, u,l\in\mathbb{R}^n_+. \end{array}\right. \tag{16}$$

Proposition 4.3 *Let $(\mathbf{\Lambda_c}, \mathbf{L_c}, \mathbf{U_c})$ the extension into intervals of a feasible solution of the constraint satisfaction problem (16),*
if $0 \notin \left(\left(A^T\mathbf{\Lambda_c} - \mathbf{L_c} + \mathbf{U_c}\right)^T\varepsilon - \mathbf{\Lambda_c}^T[-\infty, b] + \mathbf{L_c}^T\varepsilon - \mathbf{U_c}^T\varepsilon\right)$ then the general problem (8) is guaranteed to be infeasible.

Proof. By applying the previous calculation with the dual residual $r \in \mathbf{R} = A^T\mathbf{\Lambda_c} - \mathbf{L_c} + \mathbf{U_c}$, we obtain that:

if $0 \notin \left(\left(A^T \mathbf{\Lambda_c} - \mathbf{L_c} + \mathbf{U_c}\right)^T \varepsilon - \mathbf{\Lambda_c}^T [-\infty, b] + \mathbf{L_c}^T \varepsilon - \mathbf{U_c}^T \varepsilon \right)$, then the primal program (9) is guaranteed to be infeasible. Thus by applying Corollary 3.5, Proposition 4.3 is proven. □

Indeed, using Propositions 4.2 and 4.3, we have a reliable way to compute a lower bound on the objective function and a certificate of infeasibility by taking into account the constraints. In the next section, we will explain how to integrate this technique into an interval branch and bound algorithm.

5 Application within an Interval Branch and Bound Algorithm

In order to prove the efficiency of the reformulation method described previously, we apply it in an Interval Branch and Bound Algorithm named *IBBA*, previously developed by two of the authors [26, 34]. The general principle is described next, as Algorithm 2. Note that there exist other algorithms based on interval arithmetic such as for example *GlobSol*, developed by Kearfott [15], to which our method can be adapted. The fundamental principle is still the same, except that different acceleration techniques are used.

Algorithm 2 Interval Branch and Bound Algorithm: IBBA

1: $\mathbf{X}$ = initial hypercube in which the global minimum is searched, $\{\mathbf{X} \subseteq \mathbb{R}^n\}$
2: $\tilde{f} = \infty$, denotes the current upper bound on the global minimum value,
3: $\mathcal{L} = \{(\mathbf{X}, -\infty)\}$, initialization of the data structure of stored elements, {all elements in $\mathcal{L}$ have two components: a box $\mathbf{Z}$ and f_z ,a lower bound of $f(\mathbf{Z})$,}
4: **repeat**
5: Extract from $\mathcal{L}$ the element which has the smallest lower bound,
6: Choose the component which has the maximal width and bisect it by the middle, to get $\mathbf{Z}_1$ and $\mathbf{Z}_2$,
7: **for** $j = 1$ to 2 **do**
8: `Pruning of` $\mathbf{Z}_j$ `by a` **Constraint Propagation Technique** `[25],`
9: **if** $\mathbf{Z}_j$ is not empty **then**
10: Compute $f_{z\,j}$, a lower bound of $f(\mathbf{Z}_j)$, and all the lower and upper bounds of all the constraints over $\mathbf{Z}_j$,
11: **if** $\left(\tilde{f} - \epsilon_f \max(|\tilde{f}|, 1) \geq f_{z\,j}\right)$ and no constraint is unsatisfied **then**
12: Insert $(\mathbf{Z}_j, f_{z\,j})$ into $\mathcal{L}$,
13: $\tilde{f} = \min(\tilde{f}, f(\text{mid}(\mathbf{Z}_j)))$, if and only if $\text{mid}(\mathbf{Z}_j)$ satisfies all the constraints,
14: **if** $\tilde{f}$ is modified **then**
15: $\tilde{x} = \text{mid}(\mathbf{Z}_j)$,
16: Discard from $\mathcal{L}$ all the pairs $(\mathbf{Z}, f_z)$ which $\left(f_z > \tilde{f} - \epsilon_f \max(|\tilde{f}|, 1)\right)$ is checked,
17: **end if**
18: **end if**
19: **end if**
20: **end for**
21: **until** $\left(\tilde{f} - \min_{(\mathbf{Z}, f_z) \in \mathcal{L}} f_z \leq \epsilon_f \max(|\tilde{f}|, 1)\right)$ or $\mathcal{L} = \emptyset$

In Algorithm 2, at each iteration, the domain under study is choosen and bisected to improve the computation of bounds. In Line 11 of Algorithm 2, boxes are eliminated if and only if it is certified that at least one constraint cannot be satisfied by any point in such a box, or that no point in the box can produce a solution better than the current best solution minus the required relative accuracy. The criterion $\left(\tilde{f} - \epsilon_f \max(|\tilde{f}\,|, 1) \geq f_{z_j}\right)$ has the advantage to reduce the *cluster problem*, which happens when an infinity of equivalent solutions exists. At the end of the execution, Algorithm 2 is able to provide only one global minimizer $\tilde{x}$, if a feasible solution exists.

$\tilde{x}$ is reliably proven to be a global minimizer with a relative guaranteed error ϵ_f. If Algorithm 2 does not provide a solution, this proves that the problem is infeasible (case when $\tilde{f} = \infty$ and $\mathcal{L} = \emptyset$). For more details about this kind of interval branch and bound algorithms, please refer to [12, 15, 26, 34, 36].

One of the main advantages of Algorithm 2 is its modularity. Indeed, acceleration techniques can be inserted or removed from IBBA. For example at Line 8, an interval constraint propagation technique is included to reduce the width of boxes $\mathbf{Z}_j$, for more details refer to [25]. Another implementation of this method is included in the code *RealPaver* [10], the code *Couenne* of the project *COIN-OR* [3] and the code *GlobSol* [17]. This additional technique improves the speed of the convergence of such a Branch and Bound algorithm.

Affine reformulation techniques described in the previous sections can also be introduced in Algorithm 2. This routine must be inserted between Lines 8 and 9. At each iteration, as described in Section 3, for each $\mathbf{Z}_1$ and $\mathbf{Z}_2$, the associated linear program (9) is automatically generated and a linear solver (such as C-PLEX) is applied. If the linear program is infeasible, the element is eliminated. Otherwise the solution of the linear program is used to compute a lower bound of the general problem over the boxes $\mathbf{Z}_1$ and $\mathbf{Z}_2$.

Algorithm 3 Affine Reformulation Technique by Affine Arithmetic: ART_{AF}

Let $(\mathbf{Z}, f_z)$ be the current element and f_z a lower bound of f over $\mathbf{Z}$,
Initialize a linear program with the same number of variables as the general problem (8),
Generate the affine form of f using AF1 or AF2,
Define c the objective function of (9) and E_f the lower bound of the error term of the affine form of f, $c = (f_1, \ldots, f_n)$, $E_f = f_0 - f_{n+1}$ with AF1 or $E_f = f_0 - f_{n+1} - f_{n+3}$ with AF2,
for all constraints g of the general problem (8) **do**
  Calculate G($\mathbf{Z}$), the natural interval extension of g over $\mathbf{Z}$,
  if g is not satisfied over $\mathbf{Z}$ **then**
    Generate the affine form of g using AF1 or AF2,
    Add the associated linear constraint(s) into the linear program (9), such as described in Section 3,
  end if
end for
Solve the linear program (9) with a linear solver such as C-PLEX,
if the linear program has a solution y_{sol} **then**
  $f_z = \max(f_z, c^T y_{sol} + E_f)$,
else if the linear program is infeasible **then**
  Eliminate the element $(\mathbf{Z}, f_z)$
end if

Remark 5.1 *In order to take into account the value of the current minimum in the affine reformulation technique, the equation* $f(x) \leq \tilde{f}$ *is added to the constraints when* $\tilde{f} \neq \infty$.

Algorithm 3 describes all the steps of the affine reformulation technique ART_{AF}. The purpose of this method is to accelerate the resolution by reducing the number of iterations and the computation time of Algorithm 2. At Line 7 of Algorithm 3, Proposition 3.9 is used to reduce the number of constraints; this limits the size of the linear program without losing any information. The computation performed in Line 14 provides a lower bound for the general problem over a particular box by using Proposition 3.6. Corollary 3.5 involves the elimination part which corresponds to Line 16. If the linear solver cannot produce a solution in an imposed time or within a given number of iterations, the bound is not modified and Algorithm 2 continues.

Remark 5.2 *Affine arithmetic cannot be applied to functions which increase to infinity; for example, $f(x) = 1/x$, $x \in [-1, 1]$. In this case, it is impossible to construct a linearization of this function with our method. Therefore, if the objective function corresponds to this case, the bound is not modified and Algorithm 2 continues without using the affine reformulation technique at the current iteration. More generally, if it is impossible to linearize a constraint, the method continues without including this constraint into the linear program. Thus the linear program is more relaxed, and the computation of the lower bound and the elimination property are still correct.*

In Section 4, we have explained how the affine reformulation technique can be reliable. Algorithm 4 summarizes this method named $rART_{\text{rAF}}$ and adapts Algorithm 3. We first use rAF1 or rAF2 with the conversion between rAF1/AF1 or rAF2/AF2 to produce the linear program (9), using Equations (12), Propositions 3.1 and 3.2. Then, Proposition 3.9 is used in line 7 of Algorithm 4 to reduce the number of constraints. Thus, in most cases, the number of added constraints is small, and the dual resolution is improved. Moreover, we do not need to explicitly give the primal solution, thus we advise generating the dual (13) directly and solving it with a primal solver. If a dual solution is found, Proposition 4.2 guarantees a lower bound of the objective function, line 16 of Algorithm 4. Otherwise, if the solver returns that the dual is unbounded or infeasible, Proposition 4.3 produces a certificate of infeasibility for the original problem (8).

In this section, we have described two new acceleration methods, which can be added to an interval branch and bound algorithm. $rART_{\text{rAF}}$ (Algorithm 4) included in *IBBA* (Algorithm 2) allows us to take into account of rounding errors everywhere in the interval branch and bound codes. In the next section, this method will be tested to several numerical tests to prove its efficiency concerning CPU-times and the number of iterations.

6 Numerical Tests

In this section, 74 non-linear and non-convex constrained global optimization problems are considered. These test problems come from library 1 of the COCONUT website [31, 33]. We take into account all the problems with constraints, having less than 25 variables and without the cosine and sine functions which are not yet implemented in our affine arithmetic code [24, 34]; however square root, inverse, logarithm and exponential functions are included, using Algorithm 1. For all 74 test problems, the expressions of all equations are exactly as each one is defined in the COCONUT format. No modification has been done on the expressions of those functions and constraints, even when some of them are clearly unadapted to the computation of bounds with interval and affine arithmetic.

The code is written in Fortran 90/95 using the `f90` ORACLE compiler which includes a library for interval arithmetic. In order to solve the linear programming relaxation, C-PLEX version 11.0 is used. All tests are performed on a Intel-Xeon based 3 GHz computer with 2 GB of RAM and using a 64-bit Linux system (the standard time unit (STU) is 13 seconds which corresponds to 10^8 evaluations of the Shekel-5 function at the point $(4, 4, 4, 4)^T$). The termination condition is based on the precision of the value of the global minimum: $\tilde{f} - \min_{(\mathbf{Z}, f_z) \in \mathcal{L}} f_z \leq \epsilon_f \max(|\tilde{f}|, 1)$.

Algorithm 4 reliable Affine Reformulation Technique by reliable Affine Arithmetic: $rART_{\text{rAF}}$

Let $(\mathbf{Z}, f_z)$ be the current element and f_z a lower bound of $f(\mathbf{Z})$,
Initialize a linear program with the same number of variables as the general problem (8),
Generate the reliable affine form of f using rAF1 or rAF2,
Using the conversion rAF1/AF1 or rAF2/AF2, generate E_f and c of the linear program (9),

for all constraint g of the general problem (8) **do**
  Calculate $\mathrm{G}(\mathbf{Z})$, the natural interval extension of g over $\mathbf{Z}$,
  if g is not satisfied over $\mathbf{Z}$ **then**
    Generate the affine form of g using rAF1 or rAF2 forms and conversions rAF1/AF1 or rAF2/AF2,
    Add the associated linear constraints to the linear program (9) such as described in Section 3,
  end if
end for
Generate the dual program (13) of the corresponding linear program (9),
Solve the dual (13) with a primal linear solver,
if the dual program has a solution (λ_s, l_s, u_s) **then**
  $(\mathbf{\Lambda_s}, \mathbf{L_s}, \mathbf{U_s})$ = extension of (λ_s, l_s, u_s) to intervals,
  $f_z = \max\left(f_z, \underline{\left(\mathbf{R}^T\varepsilon - \mathbf{\Lambda_S}^T[-\infty, b] + \mathbf{L_S}^T\varepsilon - \mathbf{U_S}^T\varepsilon\right) + E_f}\right)$,
else if the dual program is infeasible **then**
  Solve the program (16) associated with the dual (13),
  if program (16) has a solution (λ_c, l_c, u_c) **then**
    $(\mathbf{\Lambda_c}, \mathbf{L_c}, \mathbf{U_c})$ = extension of (λ_c, l_c, u_c) to intervals,
    if $0 \notin \left(\left(A^T\mathbf{\Lambda_c} - \mathbf{L_c} + \mathbf{U_c}\right)^T\varepsilon - \mathbf{\Lambda_c}^T[-\infty, b] + \mathbf{L_c}^T\varepsilon - \mathbf{U_c}^T\varepsilon\right)$ **then**
      Eliminate the element $(\mathbf{Z}, f_z)$
    end if
  end if
end if

This relative accuracy is fixed to $\varepsilon_f = 10^{-8}$ for all the problems and the same value is taken into account for the satisfaction of the constraints, respectively. The accuracy to solve the linear program by C-PLEX is fixed to 10^{-8} and we limit the number of iterations of a run of C-PLEX to 15. Furthermore, two limits are imposed: (a) on the CPU-time which must be less than 60 minutes and (b) on the maximum number of elements in $\mathcal{L}$ which must be less than two million (corresponding approximately to the limit of the RAM of our computer for the largest problem). When the code terminates normally the values corresponding to (i) whether the problem is solved or not, (ii) the number of iterations of the main loop of Algorithm 2, and (iii) the CPU-time in seconds (s) or in minutes (min), are respectively given in columns 'ok?', 'iter' and 't' of Tables 1, 2 and 3.

The name of the COCONUT problems are in the first column of the tables; in the COCONUT website, all problems and best known solutions are given. Columns N and M represent the number of variables and the number of constraints for each problem. Test problem *hs071* from the library 2 of COCONUT corresponds to Example 3.10 when the box is $\mathbf{X} = [1, 5]^4$ and the constraints are only c_1 and c_2.

For all tables and performance profiles, *IBBA+CP* indicates results obtained with Algo-

rithm 2 (IBBA) and the constraint propagation technique (CP) described in [25].
IBBA+rART$_{\text{rAF2}}$ represents results obtained with Algorithm 2 and the reliable affine reformulation technique based on the rAF2 affine form (Algorithm 4) and the corresponding affine arithmetic [24, 34, 28]. *IBBA+rART*$_{\text{rAF2}}$ *+CP* represents results obtained with Algorithm 2 and both acceleration techniques. *GlobSol+LR* and *GlobSol* represent the results extracted from [16] and obtained using (or not) the linear relaxation based on RLT [18].

The performance profiles, defined by Dolan and Moré in [7], are visual tools to benchmark algorithms. Thus, Tables 1, 2 and 3 are summarized in Figure 3 accordingly. The percentage of solved problems is represented as a function of the performance ratio; the latter depending itself on the CPU-time. More precisely, for each test problem, one compares the ratio of the CPU-time of each algorithm to the minimum of those CPU-times. Then the performance profiles, i.e. the cumulative distribution function for the ratio, are computed.

Remark 6.1 *Algorithm 2 was also tested alone. The results are not in Table 1 because Algorithm 2 does not work efficiently without one of the two acceleration techniques. In this case, only 24 of the 74 test problems were solved.*

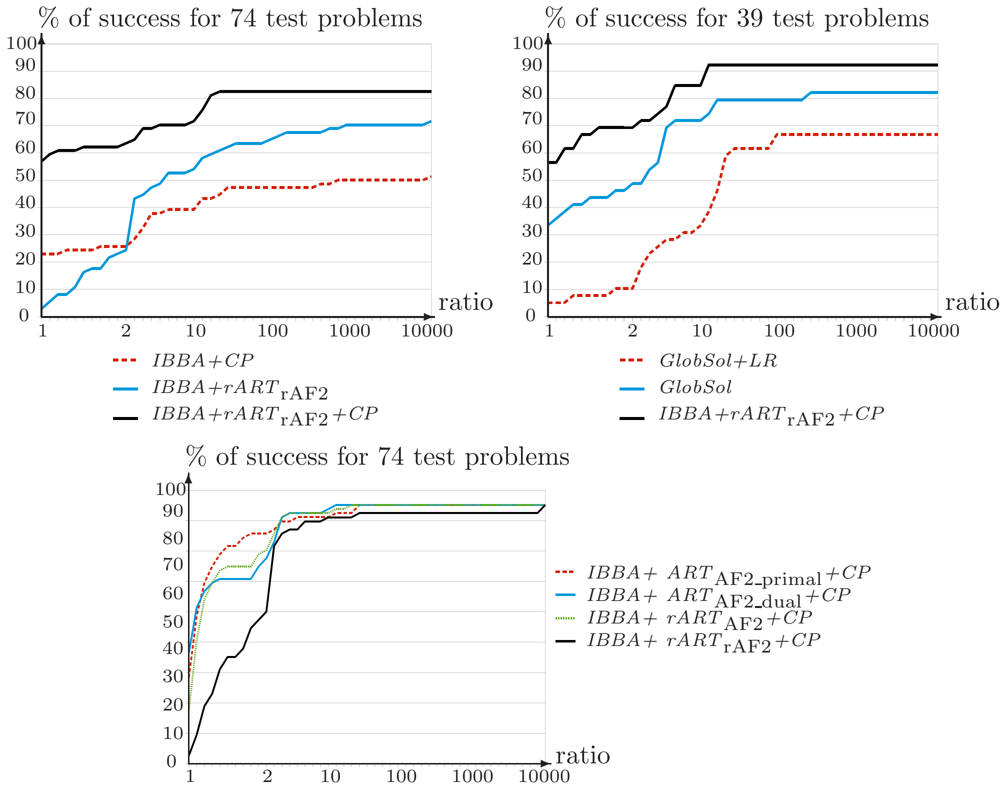

Figure 3: Performance Profile comparing the results of various versions of algorithms

6.1 Validation of the reliable approach

In Table 1, a comparison is made among the basic algorithm IBBA with constraint propagation CP, with the new relaxation technique rART and with both. It appears that:

- *IBBA+CP* solved 37 test problems, *IBBA+rART*$_{\text{rAF2}}$ 52 test problems and *IBBA+rART*$_{\text{rAF2}}$ *+CP* 61 test problems.

- The solved cases are not the same using the two distinct techniques (*CP* or *rART*$_{\text{rAF2}}$). Generally, *IBBA+CP* finished when the limit on the number of elements in the list is reached (corresponding to the limitation of the RAM). In contrast, the *IBBA+rART*$_{\text{rAF2}}$ code stopped when the limit on the CPU-time was reached.

- All problems solved with one of the acceleration techniques are solved also when both are combined. Moreover, this is achieved in a moderate computing time of about 1 min 09 s on average.

- Considering only the 33 cases solved by all three methods (in the tables), in the line 'Average when T for all' of Table 1, we obtain that average computing time of *IBBA+CP* is three times the one of *IBBA+rART*$_{\text{rAF2}}$, but is divided by a factor of about 10 when those two techniques are combined. Considering the number of iterations, the gain of *IBBA+rART*$_{\text{rAF2}}$ *+CP* is a factor of about 200 compared to *IBBA+CP*, and about 3.5 compared to *IBBA+rART*$_{\text{rAF2}}$.

The performance profiles of Figure 3 confirm that *IBBA+rART*$_{\text{rAF2}}$*+CP* is the most efficient and effective of the three first studied algorithms. Considering the curve of the algorithms *IBBA+rART*$_{\text{rAF2}}$ and *IBBA+CP* shows *IBBA+CP* is in general faster than the other but *IBBA+rART*$_{\text{rAF2}}$ solves more problems, which implies a crossing of the two curves.

Observing when the two techniques lead to fathoming a subproblem shows that the reformulation *rART*$_{\text{rAF2}}$ is more precise when the box under study is small. This technique is slow at the beginning and becomes very efficient after a while. In contrast, *CP* enhances the convergence when the box is large, but since it considers the constraints one by one, this technique is less useful at the end. That is why the combination of *CP* and *rART*$_{\text{rAF2}}$ is so efficient: *CP* reduces quickly the size of boxes and then *rART*$_{\text{rAF2}}$ improves considerably the lower bound on each box and eliminates boxes which do not contain the global minimum.

In Table 2, column 'our guaranteed UB' corresponds to the upper bound found by our algorithm and column 'UB of COCONUT' corresponds to the upper bound listed in [31] and found by the algorithm of the column 'Algorithm'. We remark that all of our bounds are close to those of COCONUT. These small differences appear to be due to the accuracy guaranteed on the constraint satisfactions.

6.2 Comparison with *GlobSol*

Kearfott and Hongthong in [18] have developed another technique based on the same principle such as Reformulation-Linearization Technique (RLT), by replacing each nonlinear term by linear overestimators and underestimators. This technique was well-known and already embedded without interval and affine arithmetics in the software package BARON [39]. Another paper by Kearfott [16] studies its integration into an interval branch and bound algorithm named *GlobSol*. In [16], the termination criteria of the branch and bound code are similar for *GlobSol* and *IBBA+rART*$_{\text{rAF2}}$*+CP*. The numerical results from [16] were inserted in Table 2 in order to compare them with the IBBA algorithm with both acceleration techniques. This empirical comparison between *GlobSol* and *IBBA+rART*$_{\text{rAF2}}$*+CP* should be considered as a first approximation. Indeed, (i) the CPU-times in Table 2 depend on the performances of the two different computers (Kearfott used GlobSol with the Compaq Visual Fortran version 6.6, on a Dell Inspiron 8200 notebook with a mobile Pentium 4 processor running at 1.60 GHz), (ii) the version of *GlobSol* used in [16] is not the last one, and (iii) it is the first version of *IBBA+rART*$_{\text{rAF2}}$*+CP*

Name	N	M	IBBA+CP ok?	IBBA+CP iter	IBBA+CP t (s)	IBBA+$rART_{\text{rAF2}}$ ok?	IBBA+$rART_{\text{rAF2}}$ iter	IBBA+$rART_{\text{rAF2}}$ t (s)	IBBA+$rART_{\text{rAF2}}$+CP ok?	IBBA+$rART_{\text{rAF2}}$+CP iter	IBBA+$rART_{\text{rAF2}}$+CP t (s)
hs071	4	2	**T**	9,558,537	722.34	**T**	1,580	2.44	**T**	804	1.04
ex2_1_1	5	1	**T**	26,208	1.17	**T**	151	0.32	**T**	151	0.23
ex2_1_2	6	2	**T**	105	0.00	**T**	289	0.46	**T**	105	0.18
ex2_1_3	13	9	F	2,004,691	106.02	**T**	352	0.74	**T**	266	0.52
ex2_1_4	6	5	**T**	5,123	0.25	**T**	641	0.74	**T**	250	0.27
ex2_1_5	10	11	**T**	172,195	32.70	**T**	844	1.98	**T**	263	0.66
ex2_1_6	10	5	**T**	5,109,625	565.22	**T**	286	0.77	**T**	285	0.69
ex2_1_7	20	10	F	7,075,425	1,735.15	**T**	1,569	16.26	**T**	1,574	16.75
ex2_1_8	24	10	F	2,005,897	280.95	**T**	3,908	53.38	**T**	1,916	26.78
ex2_1_9	10	1	F	1,999,999	93.57	**T**	66,180	160.10	**T**	60,007	154.02
ex2_1_10	20	10	F	1,999,999	635.95	**T**	938	8.81	**T**	636	5.91
ex3_1_1	8	6	F	38,000,000	3,604.46	**T**	81,818	137.02	**T**	131,195	115.92
ex3_1_2	5	6	**T**	6,571	0.44	**T**	144	0.36	**T**	111	0.19
ex3_1_3	6	6	**T**	4,321	0.21	**T**	243	0.55	**T**	182	0.24
ex3_1_4	3	3	**T**	21,096	1.06	**T**	171	0.37	**T**	187	0.25
ex4_1_8	2	1	**T**	78,417	2.31	**T**	137	0.32	**T**	128	0.11
ex4_1_9	2	2	**T**	49,678	6.38	**T**	171	0.19	**T**	157	0.17
ex5_2_2_case1	9	6	F	4,266,494	308.90	F	2,300,000	3,699.67	**T**	5,233	8.05
ex5_2_2_case2	9	6	F	7,027,892	529.41	F	2,200,000	3,646.14	**T**	9,180	14.73
ex5_2_2_case3	9	6	F	3,671,986	257.71	F	2,300,000	3,682.61	**T**	2,255	3.44
ex5_2_4	7	6	F	3,338,206	510.99	**T**	128,303	142.42	**T**	9,848	11.30
ex5_4_2	8	6	F	43,800,000	3,606.12	**T**	8,714	12.72	**T**	201,630	121.45
ex6_1_1	8	6	F	5,270,186	2,805.03	F	1,600,000	3,756.88	F	1,500,000	3,775.80
ex6_1_2	4	3	**T**	15,429	0.83	**T**	1,813	2.39	**T**	108	0.26
ex6_1_3	12	9	F	4,534,626	3,233.97	F	900,000	3,704.39	F	1,000,000	3,913.87
ex6_1_4	6	4	F	2,444,266	204.92	**T**	148,480	262.65	**T**	1,622	2.70
ex6_2_5	9	3	F	1,999,999	192.80	F	800,000	3,934.43	F	800,000	4,055.02
ex6_2_6	3	1	F	2,097,277	124.56	F	2,100,000	3,719.93	**T**	922,664	1,575.43
ex6_2_7	9	3	F	1,999,999	229.94	F	500,000	3,973.21	F	500,000	4,036.90
ex6_2_8	3	1	F	2,003,020	118.81	**T**	634,377	1,122.06	**T**	265,276	457.87
ex6_2_9	4	2	F	3,724,203	369.78	F	1,500,000	3,700.92	**T**	203,775	522.57
ex6_2_10	6	3	F	1,999,999	241.17	F	1,300,000	3,872.20	F	1,200,000	3,775.14
ex6_2_11	3	1	F	2,729,823	149.66	**T**	214,420	346.71	**T**	83,487	140.51
ex6_2_12	4	2	F	2,975,037	202.77	**T**	1,096,081	2,136.20	**T**	58,231	112.58
ex6_2_13	6	3	F	2,007,671	332.47	F	1,600,000	3,605.98	F	1,500,000	3,650.76
ex6_2_14	4	2	**T**	8,446,077	988.14	**T**	450,059	956.88	**T**	95,170	207.78
ex7_2_1	7	14	F	9,324,644	2,512.97	**T**	18,037	50.64	**T**	8,419	24.72
ex7_2_2	6	5	F	4,990,110	1,031.30	**T**	4,312	5.64	**T**	531	0.87
ex7_2_3	8	6	F	41,000,000	3,607.35	F	2,300,000	3,684.73	F	2,200,000	3,716.02
ex7_2_5	5	6	**T**	6,000	0.67	**T**	249	0.52	**T**	186	0.40
ex7_2_6	3	1	F	7,022,520	326.38	**T**	2,100	1.89	**T**	1,319	1.23
ex7_2_10	11	9	**T**	1,417	0.09	**T**	2,605	3.96	**T**	1,417	2.19
ex7_3_1	4	7	**T**	33,347	4.23	**T**	2,713	6.21	**T**	1,536	3.50
ex7_3_2	4	7	**T**	141	0.08	**T**	2,831	3.05	**T**	141	0.28
ex7_3_3	5	8	**T**	18,603	2.14	**T**	1,104	1.74	**T**	373	0.66
ex7_3_4	12	17	F	3,194,446	467.81	F	800,000	3,756.89	F	1,000,000	3,971.41
ex7_3_5	13	15	F	3,017,872	513.88	F	500,000	4,291.36	F	500,000	4,259.44
ex7_3_6	17	17	**T**	1	0.00	**T**	84	5.55	**T**	1	0.14
ex8_1_7	5	5	F	3,807,889	395.30	**T**	6,183	12.25	**T**	1,432	2.64
ex8_1_8	6	5	F	4,990,110	1,029.01	**T**	4,312	5.65	**T**	531	0.87
ex9_2_1	10	9	**T**	161	0.02	F	1,800,000	3,637.95	**T**	64	0.26
ex9_2_2	10	11	F	5,902,793	314.66	F	2,000,000	3,653.05	F	4,700,000	3,602.48
ex9_2_3	16	15	**T**	884	0.15	F	1,500,000	3,740.15	**T**	156	0.50
ex9_2_4	8	7	**T**	77	0.00	**T**	4,682	7.96	**T**	49	0.25
ex9_2_5	8	7	**T**	51,303	7.59	**T**	6,331	12.69	**T**	136	0.44
ex9_2_6	16	12	F	2,895,007	233.85	F	1,000,000	3,611.39	F	1,200,000	3,756.22
ex9_2_7	10	9	**T**	161	0.02	F	1,700,000	3,643.63	**T**	64	0.35
ex14_1_1	3	4	**T**	367	0.13	**T**	1,728	2.75	**T**	301	0.54
ex14_1_2	6	9	**T**	619,905	145.68	**T**	59,677	206.88	**T**	24,166	54.58
ex14_1_3	3	4	**T**	94	0.00	F	8,000,000	3,629.02	**T**	91	0.26
ex14_1_5	6	6	**T**	165,381	18.06	**T**	3,961	6.38	**T**	1,752	2.99
ex14_1_6	9	15	**T**	42,139	8.88	**T**	6,326	26.61	**T**	2,531	12.45
ex14_1_7	10	17	F	9,600,000	3,635.90	F	600,000	4,155.17	F	1,100,000	3,703.00
ex14_1_8	3	4	**T**	98	0.01	**T**	2,011	2.30	**T**	77	0.25
ex14_1_9	2	2	**T**	1,300	0.05	**T**	23,465	17.84	**T**	223	0.35
ex14_2_1	5	7	**T**	12,017,408	1,683.62	**T**	30,436	64.13	**T**	16,786	36.73
ex14_2_2	4	5	**T**	8,853	0.67	**T**	2,671	3.64	**T**	1,009	1.39
ex14_2_3	6	9	F	13,800,000	3,622.13	**T**	70,967	252.31	**T**	47,673	173.28
ex14_2_4	5	7	**T**	1,975,320	455.49	**T**	62,245	274.42	**T**	30,002	127.56
ex14_2_5	4	5	**T**	18,821	1.92	**T**	5,821	11.75	**T**	2,041	3.70
ex14_2_6	5	7	F	9,543,033	2,124.91	**T**	138,654	407.27	**T**	74,630	237.56
ex14_2_7	6	9	F	7,678,896	2,844.60	F	800,000	4,021.63	F	700,000	3,841.44
ex14_2_8	4	5	**T**	2,085,323	279.49	**T**	31,840	57.17	**T**	10,044	19.13
ex14_2_9	4	5	**T**	463,414	70.83	**T**	19,474	40.44	**T**	6,582	14.59
Average when '**T**'			37	1,108,213.51	135.16	52	64,547.85	131.89	61	37,556.70	69.3
Average when '**T**' for all			33	1,242,503.03	151.54	33	22,023.73	52.23	33	5,977.39	14.98

Table 1: Numerical results for reliable IBBA based methods

which does not include classical accelerating techniques as the use of a local solver to improve the upper bounds. However, this does not modify our conclusions. It appears that:

- *GlobSol+LR* solves 26 among the subset of 39 test problems attempted, *GlobSol* without *LR* solves 32 of them, and *IBBA+rART*$_{\mathrm{rAF2}}$*+CP* solves 36 of them.
- Kearfott limited his algorithm to problems with 10 variables at most. Indeed problems solved by *GlobSol* without *LR* have at most 8 variables and 9 constraints. Problems solved by *IBBA+rART*$_{\mathrm{rAF2}}$*+CP* have at most 24 variables and 17 constraints.
- *GlobSol* without *LR* solved 1 problem in 53 minutes that *IBBA+rART*$_{\mathrm{rAF2}}$ *+CP* does not solve in 60 minutes (ex6_1_1). *IBBA+rART*$_{\mathrm{rAF2}}$*+CP* solved 5 problems that *GlobSol* without *LR* does not solve and 10 that *GlobSol+LR* does not solve.

Turning now to the performance profile of Figure 3, we observe that: (i) the linear relaxation of Kearfott and Hongthong slows down their algorithm on this set of test problems; (ii) the performance of *IBBA+rART*$_{\mathrm{rAF2}}$*+CP* dominates those of *GlobSol* with and without *LR*, it still remains true if we multiply the computation time by 2 to overestimate the difference between the computers.

6.3 Comparison with the non-reliable methods

In Table 3, results for non-rigorous global optimization algorithms are presented to evaluate the cost of the reliability in our algorithm combining *CP* and *ART*$_{\mathrm{AF2}}$ techniques. Thus, we test for all three cases an IBBA algorithm associated with the *CP* and *ART*$_{\mathrm{AF2}}$ techniques (Algorithm 3) when the affine arithmetic corresponding to the affine form AF2 is not strictly reliable (the rounding errors are not taken into account). In the first and second main data column, Algorithm 3 is used and the associated linear programs for computing bounds are solved by using the primal formulation of program (9) for the column *IBBA+ART*$_{\mathrm{AF2\_primal}}$ *+CP* and the dual formulation for the column *IBBA+ART*$_{\mathrm{AF2\_dual}}$ *+CP*. In the third columns *IBBA+rART*$_{\mathrm{AF2}}$ *+CP*, we use Algorithm 4 but the linear program is generated directly with AF2 instead of rAF2, thus the linear program is not completely reliable. It appears that:

- Comparing to the reliable code *IBBA+rART*$_{\mathrm{rAF2}}$ *+CP*, two new test problems, ex6_1_1 and ex7_2_3 of COCONUT, are now solved by all the three non-reliable algorithms, see Table 3. However, this is only due to the stopping criterion on the CPU-time which is fixed to one hour.
- Analyzing the performance profiles on Figure 3, the primal formulation seems to be more efficient. Indeed until a ratio of about 2, we note that the largest part of the tests are most rapidly solved by the version using the primal formulation for solving the linear programs *IBBA+ART*$_{\mathrm{AF2\_primal}}$ *+CP*. Nevertheless, we also note that some cases are more difficult to solve using the primal formulation (see ex2_1_9 and ex7_3_1 of Table 3) than using the dual formulation. This provides the worst CPU-time average for *IBBA+ART*$_{\mathrm{AF2\_primal}}$ *+CP* even if it is generally the most efficient (see Figure 3). In fact, it appears that *IBBA+ART*$_{\mathrm{AF2\_primal}}$ *+CP* spends more time to reach the fixed precision of 10^{-8} than the dual versions; solutions with an accuracy of about 10^{-6} are rapidly obtained with the primal version, but sometimes, this code spends a huge part of the time to improve the precision until 10^{-8} is reached (as for example ex2_1_9 in Table 3).

Name	N	M	Globsol + LR		GlobSol		IBBA+ CP + rART$_{rAF2}$		our guaranteed UB	UB of COCONUT	Algorithm
			ok?	t(min)	ok?	t(min)	ok?	t(min)			
hs071	4	2					**T**	0.02	17.014017363	17.014	DONLP2
ex2_1_1	5	1	**T**	0.09	**T**	0.02	**T**	0	-16.999999870	-17	BARON7.2
ex2_1_2	6	2	**T**	0.08	**T**	0.06	**T**	0	-212.999999704	-213	MINOS
ex2_1_3	13	9					**T**	0.01	-14.999999864	-15	BARON7.2
ex2_1_4	6	5	**T**	0.16	**T**	0.09	**T**	0	-10.999999934	-11	DONLP2
ex2_1_5	10	11					**T**	0.01	-268.014631487	-268.0146	MINOS
ex2_1_6	10	5					**T**	0.01	-38.999999657	-39	BARON7.2
ex2_1_7	20	10					**T**	0.28	-4,150.410133579	-4,150.4101	BARON7.2
ex2_1_8	24	10					**T**	0.45	15,639.000022211	15,639	BARON7.2
ex2_1_9	10	1					**T**	2.57	-0.375	-0.3750	MINOS
ex2_1_10	20	10					**T**	0.1	49,318.017963635	49,318.018	MINOS
ex3_1_1	8	6	F	60.18	F	60.17	**T**	1.93	7,049.248020538	7,049.2083	BARON7.2
ex3_1_2	5	6					**T**	0	-30,665.538672616	-30,665.54	LINGO8
ex3_1_3	6	6					**T**	0	-309.999998195	-310	BARON7.2
ex3_1_4	3	3	**T**	0	**T**	0	**T**	0	-3.999999985	-4	DONLP2
ex4_1_8	2	1	**T**	0	**T**	0	**T**	0	-16.738893157	-16.7389	MINOS
ex4_1_9	2	2	**T**	0.01	**T**	0.01	**T**	0	-5.508013267	-5.508	BARON7.2
ex5_2_2_case1	9	6					**T**	0.13	-399.999999744	-400	DONLP2
ex5_2_2_case2	9	6					**T**	0.25	-599.999999816	-600	BARON7.2
ex5_2_2_case3	9	6					**T**	0.06	-749.999999952	-750	DONLP2
ex5_2_4	7	6	F	60.24	F	43.97	**T**	0.19	-449.999999882	-450	MINOS
ex5_4_2	8	6	**T**	1.32	**T**	4.94	**T**	2.02	7,512.230144503	7,512.2259	BARON7.2
ex6_1_1	8	6	F	100.95	**T**	53.39	F	62.93	$+\infty$	-0.0202	BARON7.2
ex6_1_2	4	3	**T**	0.41	**T**	0.02	**T**	0	-0.032463785	-0.0325	BARON7.2
ex6_1_3	12	9					F	65.23	$+\infty$	-0.3525	BARON7.2
ex6_1_4	6	4	**T**	4.49	**T**	0.24	**T**	0.05	-0.294541288	-0.2945	MINOS
ex6_2_5	9	3					F	67.58	$+\infty$	-70.7521	MINOS
ex6_2_6	3	1	F	60.24	**T**	5.1	**T**	26.26	-0.000002603	0	DONLP2
ex6_2_7	9	3					F	67.28	$+\infty$	-0.1608	BARON7.2
ex6_2_8	3	1	F	60.01	**T**	3.4	**T**	7.63	-0.027006349	-0.027	BARON7.2
ex6_2_9	4	2	F	60.03	**T**	7.72	**T**	8.71	-0.034066184	-0.0341	MINOS
ex6_2_10	6	3	F	60.1	F	60.09	F	62.92	-3.051949753	-3.052	BARON7.2
ex6_2_11	3	1	F	60.01	**T**	4.55	**T**	2.34	-0.000002672	0	MINOS
ex6_2_12	4	2	F	60.03	**T**	3.27	**T**	1.88	0.289194748	0.2892	BARON7.2
ex6_2_13	6	3	F	60.08	F	60.07	F	60.85	-0.216206601	-0.2162	BARON7.2
ex6_2_14	4	2	**T**	10.41	**T**	0.53	**T**	3.46	-0.695357929	-0.6954	MINOS
ex7_2_1	7	14					**T**	0.41	1,227.226078824	1,227.1896	BARON7.2
ex7_2_2	6	5	**T**	0.46	**T**	0.09	**T**	0.01	-0.388811439	-0.3888	DONLP2
ex7_2_3	8	6					F	61.93	7,049.277305603	7,049.2181	MINOS
ex7_2_5	5	6	**T**	0.27	**T**	0.04	**T**	0.01	10,122.493318794	10,122.4828	BARON7.2
ex7_2_6	3	1	**T**	0.01	**T**	0	**T**	0.02	-83.249728842	-83.2499	BARON7.2
ex7_2_10	11	9					**T**	0.04	0.100000006	0.1	MINOS
ex7_3_1	4	7	**T**	0.2	**T**	0.04	**T**	0.06	0.341739562	0.3417	BARON7.2
ex7_3_2	4	7	**T**	0	**T**	0.01	**T**	0	1.089863971	1.0899	DONLP2
ex7_3_3	5	8	**T**	0.19	**T**	0.06	**T**	0.01	0.817529051	0.8175	BARON7.2
ex7_3_4	12	17					F	66.19	$+\infty$	6.2746	BARON7.2
ex7_3_5	13	15					F	70.99	$+\infty$	1.2036	BARON7.2
ex7_3_6	17	17					**T**	0	no solution	no solution	MINOS
ex8_1_7	5	5	F	60.01	F	60.01	**T**	0.04	0.029310832	0.0293	MINOS
ex8_1_8	6	5	**T**	0.46	**T**	0.09	**T**	0.01	-0.388811439	-0.3888	DONLP2
ex9_2_1	10	9					**T**	0	17	17	DONLP2
ex9_2_2	10	11					F	60.04	$+\infty$	99.9995	DONLP2
ex9_2_3	16	15					**T**	0.01	0	0	MINOS
ex9_2_4	8	7	F	62.8	F	64.04	**T**	0	0.5	0.5	DONLP2
ex9_2_5	8	7	F	61.2	F	39.02	**T**	0.01	5.000000026	5	MINOS
ex9_2_6	16	12					F	62.6	56.3203125	-1	DONLP2
ex9_2_7	10	9					**T**	0.01	17	17	DONLP2
ex14_1_1	3	4	**T**	0.19	**T**	0.2	**T**	0.01	-0.000000009	0	MINOS
ex14_1_2	6	9	**T**	0.21	**T**	0.08	**T**	0.91	0.000000002	0	MINOS
ex14_1_3	3	4	**T**	0.13	**T**	1.79	**T**	0	0.000000008	0	BARON7.2
ex14_1_5	6	6	**T**	0.19	**T**	0.05	**T**	0.05	0.000000007	0	DONLP2
ex14_1_6	9	15					**T**	0.21	0.000000008	0	DONLP2
ex14_1_7	10	17					F	61.72	3,234.994301063	0	BARON7.2
ex14_1_8	3	4					**T**	0	0.000000009	0	BARON7.2
ex14_1_9	2	2	**T**	0.01	**T**	0.01	**T**	0.01	-0.000000004	0	MINOS
ex14_2_1	5	7	**T**	0.55	**T**	0.06	**T**	0.61	0.000000009	0	MINOS
ex14_2_2	4	5	**T**	0.01	**T**	0.01	**T**	0.02	0.000000009	0	MINOS
ex14_2_3	6	9	**T**	1.34	**T**	0.17	**T**	2.89	0.000000004	0	MINOS
ex14_2_4	5	7					**T**	2.13	0.000000009	0	MINOS
ex14_2_5	4	5	**T**	0.32	**T**	0.07	**T**	0.06	0.000000009	0	MINOS
ex14_2_6	5	7					**T**	3.96	0.000000004	0	MINOS
ex14_2_7	6	9					F	64.02	0.218278728	0	MINOS
ex14_2_8	4	5					**T**	0.32	0.000000009	0	MINOS
ex14_2_9	4	5					**T**	0.24	0.000000009	0	MINOS
Average when '**T**' for the sample of 39 problems			26	0.83	32	2.69	36	1.65			
Average when '**T**' for all			26	0.83	26	0.33	26	0.39			

Table 2: Comparison with *GlobSol* approach [16] and our approach.

- The increasing CPU-time to obtain reliable computations is about a factor of 2, see last line of Table 3 and Table 1 where the averages are done for the 61 cases which the reliable code find the global solution in less than one hour. Indeed, the CPU-time average for the reliable method is 69.3 seconds for the 61 results solved in Table 1, compared to about 35 seconds obtained by the two non-reliable dual versions of the code. Similar results are obtained concerning the number of iterations of reliable and non-reliable dual versions of the code which confirms that each iteration of the reliable method is about 2 times more consuming compared to the corresponding non-reliable one.

- All methods presented in Table 3 are efficient: the algorithms are not exactly reliable, but no numerical error results in a wrong optimal solution.

7 Conclusion

In this paper, we present a new reliable affine reformulation technique based on affine forms and their associated arithmetics. These methods construct a linear relaxation of continuous constrained optimization problems in an automatic way. In this study, we integrate these new reliable affine relaxation techniques into our own Interval Branch and Bound algorithm for computing lower bounds and for eliminating boxes which do not contain the global minimizer point. The efficiency of this technique has been validated on 74 problems from the COCONUT database. The main advantage of these techniques is to generate a linear program with the same number of variables as the original problem. Indeed, the linear programs require little time to be solved. Moreover, when the size of the boxes under study becomes small, the errors generated by the relaxation are reduced and the computed bounds are more precise.

Furthermore, inserting this new affine relaxation technique with constraint propagation into an interval Branch and Bound algorithm results in a relatively simple and efficient algorithm.

References

[1] I.P. Androulakis, C.D. Maranas, and C.A. Floudas. Alpha BB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.

[2] C. Audet, P. Hansen, B. Jaumard, and G. Savard. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming Series A*, 87(1):131–152, 2000.

[3] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Waechter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, 24(4-5):597–634, 2009.

[4] J.L.D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proceedings of SIBGRAPI'93 - VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, pages 9–18, 1993.

[5] L. de Figueiredo. Surface intersection using affine arithmetic. In *Proceedings of Graphics Interface'96*, pages 168–175, 1996.

[6] L. de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.

Name	N	M	IBBA+$ART_{\text{AF2\_primal}}$+CP			IBBA+$ART_{\text{AF2\_dual}}$+CP			IBBA+$rART_{\text{AF2}}$+CP		
			ok?	iter	t (s)	ok?	iter	t (s)	ok?	iter	t (s)
hs071	4	2	**T**	1,182	1.28	**T**	1,146	1.28	**T**	809	0.94
ex2_1_1	5	1	**T**	151	0.10	**T**	151	0.33	**T**	151	0.30
ex2_1_2	6	2	**T**	105	0.07	**T**	105	0.30	**T**	105	0.29
ex2_1_3	13	9	**T**	266	0.19	**T**	266	0.58	**T**	266	0.39
ex2_1_4	6	5	**T**	234	0.20	**T**	236	0.38	**T**	245	0.37
ex2_1_5	10	11	**T**	220	0.28	**T**	220	0.50	**T**	268	0.52
ex2_1_6	10	5	**T**	285	0.22	**T**	285	0.40	**T**	285	0.42
ex2_1_7	20	10	**T**	1,209	2.81	**T**	1,207	2.59	**T**	1,574	3.82
ex2_1_8	24	10	**T**	3,359	8.28	**T**	3,359	6.10	**T**	3,359	7.12
ex2_1_9	10	1	**T**	3,061,982	1,968.72	**T**	50,557	45.76	**T**	60,007	53.56
ex2_1_10	20	10	**T**	640	0.93	**T**	668	1.07	**T**	640	1.17
ex3_1_1	8	6	**T**	93,509	68.60	**T**	95,277	63.97	**T**	55,492	53.88
ex3_1_2	5	6	**T**	111	0.11	**T**	183	0.32	**T**	111	0.33
ex3_1_3	6	6	**T**	620	0.40	**T**	620	0.59	**T**	182	0.33
ex3_1_4	3	3	**T**	169	0.15	**T**	169	0.33	**T**	183	0.37
ex4_1_8	2	1	**T**	121	0.07	**T**	121	0.21	**T**	127	0.27
ex4_1_9	2	2	**T**	159	0.13	**T**	159	0.30	**T**	157	0.33
ex5_2_2_case1	9	6	**T**	5,217	6.52	**T**	5,228	5.87	**T**	5,353	6.48
ex5_2_2_case2	9	6	**T**	9,251	11.87	**T**	9,243	10.71	**T**	9,307	11.77
ex5_2_2_case3	9	6	**T**	2,034	2.43	**T**	2,031	2.32	**T**	2,211	2.72
ex5_2_4	7	6	**T**	9,742	8.13	**T**	9,684	8.07	**T**	9,914	8.55
ex5_4_2	8	6	**T**	24,596	12.27	**T**	109,474	46.20	**T**	137,661	59.82
ex6_1_1	8	6	**T**	1,784,279	3,341.29	**T**	1,742,095	2,888.04	**T**	1,724,012	3,098.62
ex6_1_2	4	3	**T**	108	0.10	**T**	108	0.33	**T**	108	3.36
ex6_1_3	12	9	F	1,400,000	3,714.25	F	1,600,000	3,644.57	F	1,500,000	3,612.44
ex6_1_4	6	4	**T**	1,605	6.92	**T**	1,606	1.87	**T**	1,622	2.08
ex6_2_5	9	3	F	1,999,999	1,463.63	F	1,999,999	1,440.09	F	1,999,999	1,469.85
ex6_2_6	3	1	**T**	925,230	771.71	**T**	925,180	767.14	**T**	923,064	842.48
ex6_2_7	9	3	F	1,900,000	3,784.13	F	1,999,999	3,723.61	F	1,900,000	3,737.05
ex6_2_8	3	1	**T**	263,014	246.99	**T**	262,902	226.24	**T**	265,337	247.04
ex6_2_9	4	2	**T**	202,409	204.80	**T**	202,436	212.71	**T**	203,764	223.66
ex6_2_10	6	3	F	4,630,829	3,265.45	F	4,630,829	3,262.37	F	4,630,829	3,294.58
ex6_2_11	3	1	**T**	83,306	77.69	**T**	83,318	70.49	**T**	83,483	74.96
ex6_2_12	4	2	**T**	57,667	48.77	**T**	57,692	50.29	**T**	58,232	53.54
ex6_2_13	6	3	F	4,287,127	2,594.17	F	4,287,127	2,510.00	F	4,287,127	2,557.77
ex6_2_14	4	2	**T**	80,960	77.74	**T**	81,813	79.54	**T**	95,156	103.80
ex7_2_1	7	14	**T**	7,365	12.74	**T**	7,521	11.74	**T**	8,653	14.58
ex7_2_2	6	5	**T**	554	0.81	**T**	554	0.81	**T**	531	0.76
ex7_2_3	8	6	**T**	2,211,218	2,322.57	**T**	2,377,846	2,398.99	**T**	2,387,169	2,579.08
ex7_2_5	5	6	**T**	198	0.42	**T**	201	0.40	**T**	176	1.36
ex7_2_6	3	1	**T**	1,301	1.21	**T**	1,301	1.15	**T**	1,319	1.11
ex7_2_10	11	9	**T**	1,417	1.30	**T**	1,417	1.28	**T**	1,417	1.31
ex7_3_1	4	7	**T**	133,163	85.37	**T**	1,656	1.80	**T**	1,536	1.93
ex7_3_2	4	7	**T**	130	0.33	**T**	130	0.27	**T**	141	0.29
ex7_3_3	5	8	**T**	309	0.62	**T**	280	0.48	**T**	373	0.57
ex7_3_4	12	17	F	3,414,184	3,413.47	F	3,600,000	3,608.75	F	3,200,000	3,619.05
ex7_3_5	13	15	F	3,439,246	3,402.08	F	3,072,081	3,272.27	F	2,228,770	3,398.21
ex7_3_6	17	17	**T**	1	0.15	**T**	1	0.13	**T**	1	0.18
ex8_1_7	5	5	**T**	1,273	1.61	**T**	1,343	1.51	**T**	1,439	1.55
ex8_1_8	6	5	**T**	554	0.82	**T**	554	0.76	**T**	531	0.76
ex9_2_1	10	9	**T**	64	0.31	**T**	64	0.22	**T**	64	0.24
ex9_2_2	10	11	F	5,902,793	3,065.19	F	5,902,793	3,079.29	F	5,902,793	3,417.14
ex9_2_3	16	15	**T**	147	0.51	**T**	154	0.49	**T**	156	0.41
ex9_2_4	8	7	**T**	49	0.29	**T**	49	0.20	**T**	49	0.24
ex9_2_5	8	7	**T**	133	0.41	**T**	133	0.35	**T**	136	0.40
ex9_2_6	16	12	F	2,476,815	2,960.31	F	2,476,815	2,773.41	F	2,476,882	3,409.49
ex9_2_7	10	9	**T**	64	0.83	**T**	64	0.31	**T**	64	0.30
ex14_1_1	3	4	**T**	2	0.23	**T**	247	0.44	**T**	300	0.41
ex14_1_2	6	9	**T**	23,965	20.39	**T**	48,087	38.85	**T**	23,976	20.67
ex14_1_3	3	4	**T**	2	0.24	**T**	91	0.30	**T**	91	0.30
ex14_1_5	6	6	**T**	5,610	3.42	**T**	4,836	2.97	**T**	2,996	3.60
ex14_1_6	9	15	**T**	2	0.27	**T**	2,379	4.61	**T**	2,266	5.32
ex14_1_7	10	17	F	3,900,000	3,700.27	F	3,900,000	3,617.51	F	3,800,000	3,702.24
ex14_1_8	3	4	**T**	72	0.41	**T**	74	0.25	**T**	77	0.29
ex14_1_9	2	2	**T**	235	0.42	**T**	235	0.33	**T**	225	0.32
ex14_2_1	5	7	**T**	406,852	306.00	**T**	262,394	198.59	**T**	16,813	20.75
ex14_2_2	4	5	**T**	1,950	1.68	**T**	1,169	1.14	**T**	1,010	1.02
ex14_2_3	6	9	**T**	47,495	80.99	**T**	47,250	75.38	**T**	48,505	80.69
ex14_2_4	5	7	**T**	235,435	223.18	**T**	128,507	126.53	**T**	29,793	40.65
ex14_2_5	4	5	**T**	2,257	2.12	**T**	2,868	2.55	**T**	2,057	1.99
ex14_2_6	5	7	**T**	73,827	118.14	**T**	73,520	106.65	**T**	74,630	110.38
ex14_2_7	6	9	F	1,800,000	3,727.71	F	1,800,000	3,668.57	F	1,800,000	3,720.75
ex14_2_8	4	5	**T**	10,327	10.92	**T**	10,117	10.75	**T**	10,066	11.01
ex14_2_9	4	5	**T**	30,200	26.47	**T**	6,764	7.92	**T**	6,581	7.78
Average when '**T**'			63	155,712.87	1[illegible].24	63	105,227.7	118.94	63	99,465.49	123.39
Avg when '**T**' for 3rd Alg. Tab. 1			61	95,318.26	72.64	61	41,137.77	36.16	61	35,330.25	34.36

Table 3: Numerical results for non reliable but exact global optimization methods

[7] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, 91(2):201–213, 2002.

[8] E. Fitan, F. Messine, and B. Nogarède. The electromagnetic actuator design problem: A general and rational approach. *IEEE Transactions on Magnetics*, 40(3):1579–1590, 2004.

[9] J. Fontchastagner, F. Messine, and Y. Lefevre. Design of electrical rotating machines by associating deterministic global optimization algorithm with combinatorial analytical and numerical models. *IEEE Transactions on Magnetics*, 43(8):3411–3419, 2007.

[10] L. Granvilliers and F. Benhamou. RealPaver: an interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software*, 32:138156, 2006.

[11] E.R. Hansen. A generalized interval arithmetic. *Lecture Notes in Computer Science*, 29:7–18, 1975.

[12] E.R. Hansen and W.G. Walster. *Global Optimization Using Interval Analysis.* Marcel Dekker Inc., New York, $2^{\text{ème}}$ edition, 2004.

[13] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches.* Springer-Verlag, Berlin, third edition, 1996.

[14] C. Jansson. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization*, 14(3):914–935, 2003.

[15] R.B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academis Publishers, Dordrecht, 1996.

[16] R.B. Kearfott. Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. *Optimization Methods & Software*, 21(5):715–731, 2006.

[17] R.B. Kearfott. GlobSol user guide. *Optimisation Methods & Software*, 24(4-5):687–708, Jan 2009.

[18] R.B. Kearfott and S. Hongthong. Validated linear relaxations and preprocessing: Some experiments. *SIAM Journal on Optimization*, 16(2):418–433, 2005.

[19] C. Keil. LURUPA: Rigorous error bounds in linear programming. In *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, Nov 2006.

[20] J.-B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[21] Y. Lebbah, C. Michel, and M. Rueher. Efficient pruning technique based on linear relaxations. In *Proceedings of Global Optimization and Constraint Satisfaction*, volume 3478, pages 1–14, 2005.

[22] C.D. Maranas and C.A. Floudas. Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21(4):351–369, 1997.

[23] M.C. Markot, J. Fernandez, L.G. Casado, and T. Csendes. New interval methods for constrained global optimization. *Mathematical Programming*, 106(2):287–318, 2006.

[24] F. Messine. Extensions of affine arithmetic: Application to unconstrained global optimization. *Journal of Universal Computer Science*, 8(11):992–1015, 2002.

[25] F. Messine. Deterministic global optimization using interval constraint propagation techniques. *RAIRO-Operations Research*, 38(4):277–293, 2004.

[26] F. Messine. A deterministic global optimization algorithm for design problems. In C. Audet, P. Hansen, and G. Savard, editors, *Essays and Surveys in Global Optimization*, pages 267–294. Springer, New York, 2005.

[27] F. Messine, B. Nogarède, and J.-L. Lagouanelle. Optimal design of electromechanical actuators: A new method based on global optimization. *IEEE Transactions on Magnetics*, 34(1):299–308, 1998.

[28] F. Messine and A. Touhami. A general reliable quadratic form: An extension of affine arithmetic. *Reliable Computing*, 12(3):171–192, 2006.

[29] A. Mitsos, B. Chachuat, and P.I. Barton. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, 20(2):573–601, 2009.

[30] R.E. Moore. *Interval Analysis*. Prentice-Hall Inc., Englewood Cliffs,, 1966.

[31] A. Neumaier. Set of test problems COCONUT. http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html.

[32] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming Series A*, 99(2):283–296, 2004.

[33] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinko. A comparison of complete global optimization solvers. *Mathematical Programming Series B*, 103(2):335–356, 2005.

[34] J. Ninin. *Optimisation Globale basé sur l'Analyse d'Intervalles: Relaxation affine et limitation de la mémoire*. PhD thesis, Institut National Polytechnique de Toulouse, 2010.

[35] S. Perron. *Applications jointes de l'optimisation combinatoire et globale*. PhD thesis, École Polytechnique de Montréal, 2004.

[36] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood Ltd, Chichester, 1988.

[37] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academis Publishers, Dordrecht, 1999.

[38] J. Stolfi and L. de Figueiredo. *Self-Validated Numerical Methods and Applications*. Monograph for 21st Brazilian Mathematics Colloquium. IMPA/CNPq, Rio de Janeiro, Brazil, 1997.

[39] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming Series A*, 99(3):563–591, 2004.

[40] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: a Modelling Language for Global Optimization*. MIT Press, Massachusetts, 1997.

[41] X.-H. Vu, D. Sam-Haroud, and B. Faltings. Enhancing numerical constraint propagation using multiple inclusion representations. *Annals of Mathematics and Artificial Intelligence*, 55(3-4):295–354, 2009.