4OR-Q J Oper Res (2015) 13:247–277
DOI 10.1007/s10288-014-0269-0

RESEARCH PAPER

A reliable affine relaxation method for global optimization

Jordan Ninin · Frédéric Messine · Pierre Hansen

Received: 13 February 2014 / Revised: 7 July 2014 / Published online: 7 October 2014

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 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 74 problems from the COCONUT database with up to 24 variables or 17

The work of Jordan Ninin has been supported by French National Research Agency (ANR) through COSINUS program (project ID4CS nANR-09-COSI-005).

The research of Pierre Hansen was supported by an NSERC Operating Grant as well as by the DIGITEO Fondation.

The work of Frederic Messine has been funded by the Junta de Andaluca (P11-TIC7176), by the Spanish Ministry (TIN2012-37483) and by the grant ANR 12-JS02-009-01 ATOMIC.

J. Ninin
LabSTICC, IHSEV Team, ENSTA-Bretagne, 2 rue François Verny, 29806 Brest, France
e-mail: jordan.ninin@ensta-bretagne.fr

F. Messine (✉)
ENSEEIHT-IRIT, 2 rue Charles Camichel, 31071 Toulouse, France
e-mail: Frederic.Messine@enseeiht.fr

P. Hansen
Studies and Research in Decision Analysis, HEC Montréal, Montreal, Canada
e-mail: Pierre.Hansen@gerad.ca

P. Hansen
LIX, École Polytechnique, Palaiseau, France

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 min.

Keywords Affine arithmetic · Interval analysis · Linear relaxation · Branch and bound algorithm · Global optimization

Mathematics Subject Classification 90C26 · 65H20 · 65G30 · 65G40 · 49M20

1 Introduction

For about thirty years, interval branch and bound algorithms are increasingly used to solve global optimization problems in a deterministic way (Hansen and Walster 2004; Kearfott 1996; Markot et al. 2006; Ratschek and Rokne 1988). 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. Androulakis et al. 1995; Audet et al. 2000; Horst and Tuy 1996; Lasserre 2001; Lebbah et al. 2005; Maranas and Floudas 1997; Mitsos et al. 2009; Sherali and Adams 1999; Tawarmalani and Sahinidis 2004), 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 (Fitan et al. 2004; Fontchastagner et al. 2007; Messine 2005; Messine et al. 1998). This work showed that interval propagation techniques based on constructions of computation trees (Messine 2004; Hentenryck et al. 1997; Vu et al. 2009) and on linearization techniques (Hansen and Walster 2004; Kearfott 1996; Lebbah et al. 2005) improved considerably the speed of convergence of the algorithm.

Another way to solve global optimization problems, initially outside the interval branch and bound framework, is the reformulation-linearization technique developed by Sherali and Adams (1999), see also Audet et al. (2000), Perron (2004) 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. Notice that in Schichl and Neumaier (2005), Schichl and Neumaier proposed another linearization technique based on the slopes of functions. This technique has the advantage of keeping the same number of variables as the original optimization problem.

Kearfott (2006), Kearfott and Hongthong (2005) and Lebbah et al. (2005) embedded the reformulation-linearization 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 (Comba and Stolfi 1993; de Figueiredo 1996; Figueiredo and Stolfi 2004; Stolfi and de Figueiredo 1997) to generate linear relaxations. This arithmetic can be considered as an extension of interval arithmetic (Moore 1966) by converting intervals into 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 with interval arithmetic, affine arithmetic can be implemented in an automated way by using computation trees and operator overloading (Mitsos et al. 2009) which are available in some languages such as C++, Fortran 90/95/2000 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 constraints. This is due to the use of two affine forms introduced by the second author in a previous work (Messine 2002). The linear relaxations have to be solved by specialized codes such as CPLEX. Techniques for obtaining reliable results with such non reliable codes have been proposed by Neumaier and Shcherbina (2004), and are used in some of the algorithms proposed below; (iv) compared with previous, often specialized, works on the interval branch and bound approach (Kearfott 2006; Lebbah et al. 2005), or which could be embedded into such an approach (Audet et al. 2000; Sherali and Adams 1999), the proposed method is fairly general and can deal with many usual functions such as logarithm, exponential, inverse, square root.

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 Sect. 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 extends usual functions of arithmetic to intervals, see Moore (1966). The set of intervals will be denoted by $\mathbb{IR}$, and the set of n-dimensional interval vectors, also called *boxes*, will be denoted by $\mathbb{IR}^n$. The four standard operations of arithmetic are defined by the following equations, where $\boldsymbol{x} = [\underline{x}, \overline{x}]$ and $\boldsymbol{y} = [\underline{y}, \overline{y}]$ are intervals:

$$\begin{aligned}
&\boldsymbol{x} \oplus \boldsymbol{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \qquad \boldsymbol{x} \odot \boldsymbol{y} = [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})],\\
&\boldsymbol{x} \ominus \boldsymbol{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \qquad \boldsymbol{x} \oslash \boldsymbol{y} = [\underline{x}, \overline{x}] \odot [1/\overline{y}, 1/\underline{y}], \qquad \text{if } 0 \notin \boldsymbol{y}.
\end{aligned}$$

These operators are the basis of interval arithmetic, and its principle can be extended to many unary functions, such as cos, sin, exp, log, $\sqrt{\cdot}$. Kearfott (1996), Stolfi and de Figueiredo (1997). We can also write $\inf(\boldsymbol{x}) = \underline{x}$ for the lower bound, $\sup(\boldsymbol{x}) = \overline{x}$ for the upper bound and $\text{mid}(\boldsymbol{x}) = \frac{\underline{x}+\overline{x}}{2}$ for the midpoint of the interval $\boldsymbol{x}$.

Given a function f of one or several variables $x_1, \ldots, x_n$ and the corresponding intervals for the variables $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, 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., $\mathsf{f} : \mathbb{D} \subseteq \mathbb{IR}^n \to \mathbb{IR}$ such that $\forall \boldsymbol{x} \in \mathbb{D}$, $\texttt{range}(f, \boldsymbol{x}) = \{f(x) : x \in \boldsymbol{x}\} \subseteq \mathsf{f}(\boldsymbol{x})$, for details see Ratschek and Rokne (1988, Sect. 2.6).

Example 1 Using the rounded interval arithmetic in Fortran double precision, such as defined in Moore (1966, Chap. 3),

$$\begin{aligned} \boldsymbol{x} &= [1,2] \times [2,6], \quad f(x) = x_1 \cdot x_2^2 - \exp(x_1 + x_2), \\ \mathsf{f}(\boldsymbol{x}) &= [1,2] \odot [2,6]^2 \ominus \exp([1,2] \oplus [2,6]) = [-2976.957987041728, \\ &\qquad 51.914463076812], \end{aligned}$$

We obtain that: $\forall x \in \boldsymbol{x},\ f(x) \in [-2976.957987041728, 51.914463076812]$.

Affine arithmetic was introduced in 1993 by Comba and Stolfi (1993) and developed by De Figueiredo and Stolfi in de Figueiredo (1996), Figueiredo and Stolfi (2004), Stolfi and de Figueiredo (1997). 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 that, the natural interval extension of $f(x) = x - x$, where $x \in \boldsymbol{x} = [\underline{x}, \overline{x}]$, is equal to $[\underline{x} - \overline{x}, \overline{x} - \underline{x}]$ 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 denote this with a slight abuse of notation as $\mathbb{R}$) and ϵ_i are symbolic real variables whose values are unknown but assumed to lie in the interval $\boldsymbol{\epsilon}_i = [-1, 1]$, see Stolfi and de Figueiredo (1997):

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

with $\forall i \in \{0, 1, \ldots, n\}, x_i \in \mathbb{R}$ and $\forall i \in \{1, 2, \ldots, n\}, \epsilon_i \in \boldsymbol{\epsilon}_i = [-1, 1]$.

An affine form can also be defined by a vector of all its components: $(x_0, x_1, \ldots, x_n)$.

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 Stolfi and de Figueiredo (1997). Conversions between affine forms and intervals are done as follows:

Interval $\longrightarrow$ Affine Form

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

Affine Form $\longrightarrow$ Interval

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

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 (Messine 2002; Stolfi and de Figueiredo 1997). These inclusion functions cannot be proved to be equivalent or better than natural interval extensions. However, empirical studies done by Stolfi and de Figueiredo (1997), Messine (2002) and Messine and Touhami (2006) 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 Stolfi and de Figueiredo (1997) 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 (Messine 2002). 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 arithmetics but all the new symbolic terms generated by approximations are accumulated 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 $\forall i \in \{0, 1, \ldots, n\}$, $x_i \in \mathbb{R}$, $x_{n+1} \in \mathbb{R}^+$, $\epsilon_i \in \boldsymbol{\epsilon}_i = [-1, 1]$ and $\epsilon_\pm \in \boldsymbol{\epsilon}_\pm = [-1, 1]$.
- 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 $\forall i \in \{1, 2, \ldots, n\}$, $\epsilon_i \in \boldsymbol{\epsilon}_i = [-1, 1]$, and $(x_{n+1}, x_{n+2}, x_{n+3}) \in \mathbb{R}^3_+$, $\epsilon_\pm \in \boldsymbol{\epsilon}_\pm = [-1, 1]$, $\epsilon_+ \in \boldsymbol{\epsilon}_+ = [0, 1]$, $\epsilon_- \in \boldsymbol{\epsilon}_- = [-1, 0]$.

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 Messine (2002), see Vu et al. (2009).

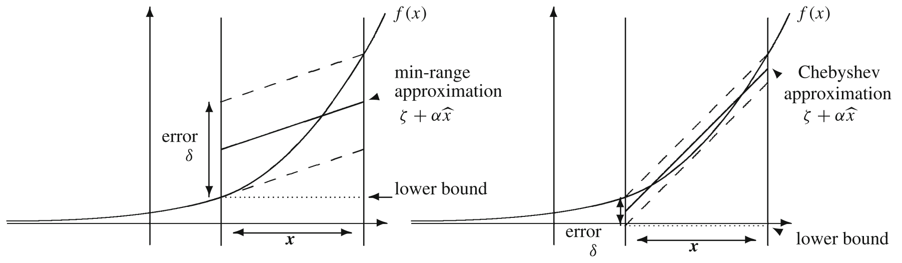

Fig. 1 Affine approximations by min-range methods and Chebyshev

Usual operations and functions are defined by extension of affine arithmetic, see Messine (2002) for details. For example, the multiplication between two affine forms of type AF1 is performed as follows:

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

For computing unary functions in affine arithmetic, Stolfi and de Figueiredo (1997) proposed two linear approximations: the Chebyshev and the min-range approximations, see Fig. 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. These affine approximations are denoted as follows:

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

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 Fig. 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 (1997), we choose to implement the min-range linearization technique. 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).

A representation of the computation of AF1 is shown in Fig. 2 (numbers are truncated with 6 digits after the decimal point). In our implementation, the computation tree is implicitly built by operator overloading (Mitsos et al. 2009). 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_i variables by $[-1, 1]$ and applying interval arithmetic.

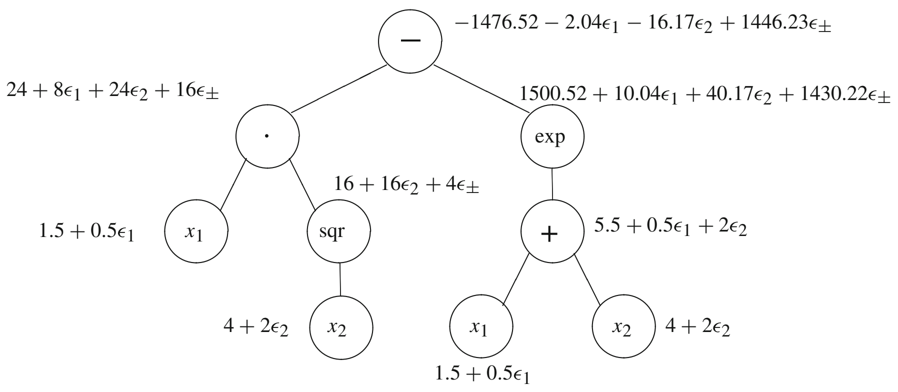

Fig. 2 Visualization of AF1 by computation tree: $f(x) = x_1 \cdot x_2^2 - \exp(x_1 + x_2)$ in $[1, 2] \times [2, 6]$

Example 2 Consider the following function:

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

First, using Eq. (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):

$$\boldsymbol{x}_1 = [1, 2] \to \widehat{x_1} = 1.5 + 0.5\epsilon_1 \text{ and } \boldsymbol{x}_2 = [2, 6] \to \widehat{x_2} = 4 + 2\epsilon_2.$$

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

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

The details of the computation by AF1 are represented in Fig. 2. The variable ϵ_1 corresponds to $\boldsymbol{x}_1$, ϵ_2 to $\boldsymbol{x}_2$ 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 square.

To conclude, using Eq. (3), we convert these affine forms into intervals to have the following bounds:

Using interval arithmetic directly, we obtain

$$\begin{aligned}
&\forall x \in [1, 2] \times [2, 6],\ f(x) \in [-2976.9579870417284, 51.91446307681234], \\
\text{using AF1: } &\forall x \in [1, 2] \times [2, 6],\ f(x) \in [-2940.9579870417297, -12.085536923186737], \\
\text{using AF2: } &\forall x \in [1, 2] \times [2, 6],\ f(x) \in [-2934.9579870417297, -12.085536923186737],
\end{aligned}$$

and the exact range with 12 digits after comma is

$$\texttt{range}(f, [1, 2] \times [2, 6]) = [-2908.957987041728, -16.085536923187668].$$

In this example, the enclosure computed by interval arithmetic contains 0. This introduces an ambiguity on the sign of f over $[1, 2] \times [2, 6]$ while it is clearly negative. This example shows why AF1 and AF2 are interesting.

An empirical comparison among interval arithmetic, AF1 and AF2 affine forms has been done on several randomly generated polynomial functions (Messine 2002), and the proof of the following proposition is given there.

Proposition 1 *Consider a polynomial function f of $\boldsymbol{x} \subset \mathbb{R}^n$ to $\mathbb{R}$ and f_{AF}, f_{AF1} and f_{AF2} the reformulations of f respectively with AF, AF1 and AF2. Then one has:*

$$\texttt{range}(f, \boldsymbol{x}) \subseteq \widehat{\mathsf{f}}_{AF2}(\boldsymbol{x}) \subseteq \widehat{\mathsf{f}}_{AF1}(\boldsymbol{x}) = \widehat{\mathsf{f}}_{AF}(\boldsymbol{x}).$$

3 Affine reformulation technique based on affine arithmetic

Since many years, reformulation techniques have been used for global optimization (Androulakis et al. 1995; Audet et al. 2000; Horst and Tuy 1996; Kearfott and Hongthong 2005; Lebbah et al. 2005; Maranas and Floudas 1997; Perron 2004; Sherali and Adams 1999; Tawarmalani and Sahinidis 2004). In most cases, the main idea is to approximate a mathematical program by a linear relaxation. Thus, solving the 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 constructed.

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 automatically linearize every factorable function (Tawarmalani and Sahinidis 2004). 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 $\boldsymbol{x} \subset \mathbb{R}^n$ and $\boldsymbol{\epsilon} = [-1, 1]^n$ is introduced, see Eq. (2); notice that $\mathcal{T}$ is bijective. Now, we can identify the linear part of AF1 and AF2 as a linear approximation of the original function. Thus, this leads to the following propositions:

Proposition 2 *Consider* $(\mathsf{f}_0, \ldots, \mathsf{f}_{n+1})$, *the components of the affine form AF1 of a function f over $\boldsymbol{x}$.*

$$\text{If } \forall x \in \boldsymbol{x},\ f(x) \leq 0, \text{ then } \forall y \in [-1, 1]^n,\ \sum_{i=1}^{n} \mathsf{f}_i y_i \leq \mathsf{f}_{n+1} - \mathsf{f}_0.$$

$$\text{If } \forall x \in \boldsymbol{x},\ f(x) = 0, \text{ then } \forall y \in [-1, 1]^n,\ \begin{cases} \displaystyle\sum_{i=1}^{n} \mathsf{f}_i y_i \leq \mathsf{f}_{n+1} - \mathsf{f}_0, \\ \displaystyle-\sum_{i=1}^{n} \mathsf{f}_i y_i \leq \mathsf{f}_{n+1} + \mathsf{f}_0. \end{cases}$$

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

$$\widehat{\mathsf{f}}(\boldsymbol{x}) = \mathsf{f}_0 + \sum_{i=1}^{n} \mathsf{f}_i \epsilon_i + \mathsf{f}_{n+1}\epsilon_{\pm},$$
$$\text{with } \forall i \in \{0, 1, \ldots, n\}, \mathsf{f}_i \in \mathbb{R}, \mathsf{f}_{n+1} \in \mathbb{R}^+,$$
$$\forall i \in \{1, 2, \ldots, n\}, \epsilon_i \in \boldsymbol{\epsilon}_i = [-1, 1] \text{ and } \epsilon_{\pm} \in \boldsymbol{\epsilon}_{\pm} = [-1, 1].$$

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

$$\forall x \in \boldsymbol{x},\ f(x) \in \mathsf{f}_0 \oplus \sum_{i=1}^{n} \mathsf{f}_i \boldsymbol{\epsilon}_i \oplus \mathsf{f}_{n+1}\boldsymbol{\epsilon}_{\pm}.$$

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

$$\begin{aligned} &\forall x \in \boldsymbol{x}, f(x) \in \left(\sum_{i=1}^{n} \mathsf{f}_i \mathcal{T}_i(x_i) \oplus \mathsf{f}_0 \oplus \mathsf{f}_{n+1}[-1, 1] \right), \\ &\forall x \in \boldsymbol{x}, f(x) - \sum_{i=1}^{n} \mathsf{f}_i \mathcal{T}_i(x_i) \in [\mathsf{f}_0 - \mathsf{f}_{n+1}, \mathsf{f}_0 + \mathsf{f}_{n+1}], \\ &\qquad \text{where } \mathcal{T}_i \text{ are the components of} \mathcal{T}. \end{aligned} \tag{7}$$

Thus the result follows. □

Proposition 3 *Consider* $(\mathsf{f}_0, \ldots, \mathsf{f}_{n+1}, \mathsf{f}_{n+2}, \mathsf{f}_{n+3})$, *the components of the affine form AF2 of a function* f *over* $\boldsymbol{x}$. .

$$\textit{If } \forall x \in \boldsymbol{x},\ f(x) \le 0 \textit{ then } \forall y \in [-1, 1]^n,\ \sum_{i=1}^{n} \mathsf{f}_i y_i \le \mathsf{f}_{n+1} + \mathsf{f}_{n+3} - \mathsf{f}_0.$$

$$\textit{If } \forall x \in \boldsymbol{x},\ f(x) = 0 \textit{ then } \forall y \in [-1, 1]^n, \begin{cases} \displaystyle\sum_{i=1}^{n} \mathsf{f}_i y_i \le \mathsf{f}_{n+1} + \mathsf{f}_{n+3} - f_0, \\ \displaystyle -\sum_{i=1}^{n} \mathsf{f}_i y_i \le \mathsf{f}_{n+1} + \mathsf{f}_{n+2} + \mathsf{f}_0. \end{cases}$$

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

$$\forall x \in \boldsymbol{x},\ f(x) - \sum_{i=1}^{n} \mathsf{f}_i \mathcal{T}_i(x_i) \in [\mathsf{f}_0 - \mathsf{f}_{n+1} - \mathsf{f}_{n+3}, \mathsf{f}_0 + \mathsf{f}_{n+1} + \mathsf{f}_{n+2}].$$

□

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

$$\begin{cases} \min\limits_{x \in \boldsymbol{x} \subset \mathbb{R}^n} & f(x) \\ \text{s.t.} & g_k(x) \leq 0, \ \forall k \in \{1, \dots, p\}, \\ & h_l(x) = 0, \ \forall l \in \{1, \dots, q\}. \end{cases} \quad (8) \qquad \begin{cases} \min\limits_{y \in [-1,1]^n} & c^T y \\ \text{s.t.} & Ay \leq b. \end{cases} \quad (9)$$

The linear objective function c of the linear program (9) is obtained from the linear part of the affine form of the objective function f of the original problem (8): i.e. $c = (\mathsf{f}_1, \mathsf{f}_2, \dots, \mathsf{f}_n)^T$. The linear constraints ($Ay \leq b$) are composed using Proposition 2 or 3 over each constraints g_k and h_l of the original problem (8).

Remark 1 The size of the linear program (9) still remains small. The number of variables is the same as in the general problem (8) and the number of inequality constraints cannot exceed twice its number of constraints.

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 bijective affine transformation between $\boldsymbol{x}$ and $\boldsymbol{\epsilon} = [-1, 1]^n$, and E_f the lower bound of the error of the affine form of f. Using AF1, $E_f = \inf(\mathsf{f}_0 \oplus \mathsf{f}_{n+1}\boldsymbol{\epsilon}_\pm) = \mathsf{f}_0 - \mathsf{f}_{n+1}$ and using AF2, $E_f = \inf(\mathsf{f}_0 \oplus \mathsf{f}_{n+1}\boldsymbol{\epsilon}_\pm \oplus \mathsf{f}_{n+2}\boldsymbol{\epsilon}_+ \oplus \mathsf{f}_{n+3}\boldsymbol{\epsilon}_-) = \mathsf{f}_0 - \mathsf{f}_{n+1} - \mathsf{f}_{n+3}$.

Proposition 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 2 and 3. □

Corollary 1 *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 4, $S_2 = \emptyset$ implies $S_1 = \emptyset$ and the result follows. □

Proposition 5 *If y_{sol} is a solution which minimizes the linear program (9), then*

$$\forall x \in S_1, \ f(x) \geq (\mathsf{f}_1, \dots, \mathsf{f}_n)^T y_{sol} + E_f,$$

with $E_f = \mathsf{f}_0 - \mathsf{f}_{n+1}$ if AF1 has been used to generate the linear program (9), and $E_f = \mathsf{f}_0 - \mathsf{f}_{n+1} - \mathsf{f}_{n+3}$ if AF2 has been used.

Proof Using Proposition 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 2 and Proposition 3, we have $\forall x \in S_1, \exists y \in [-1, 1], f(x) - c^T y \geq E_f$ and therefore $\forall x \in S_1, f(x) \geq c^T y_{sol} + E_f$.

□

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 $[-1, 1]^n$, instead of $\boldsymbol{x}$.

Proposition 6 *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 another 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 Messine (2002) (with corrections as in Vu et al. (2009)) and mentioned in Proposition 1, 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 4 yields $\forall x \in S_1$, $y = \mathcal{T}(x) \in S_2$ and then,

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

□

Remark 2 Proposition 6 could be generalized to factorable functions depending on the definition of transcendental functions in AF1 and AF2 corresponding arithmetic. In Messine (2002), only affine operations and the multiplication between two affine forms were taken into account.

Proposition 7 *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 $\boldsymbol{x}$ (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). □

Example 3 Let us consider the following problem with $\boldsymbol{x} = [1, 1.5] \times [4.5, 5] \times [3.5, 4] \times [1, 1.5]$:

$$\begin{cases} \min\limits_{x \in \boldsymbol{x}} x_3 + (x_1 + x_2 + x_3)x_1x_4 \\ s.t. \;\; x_1x_2x_3x_4 \geq 25, \\ \qquad x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40, \\ \qquad 5x_1^4 - 2x_2^3 + 11x_3^2 + 6e^{x_4} \leq 50. \end{cases}$$

We denote the first constraint by c_1, the second one by c_2 and the last one by c_3. First we compute an enclosure of each constraint by interval arithmetic:

$$\mathsf{c}_1(\boldsymbol{x}) = [15.75, 45.0],\ \mathsf{c}_2(\boldsymbol{x}) = [34.5, 45.5] \text{ and } \mathsf{c}_3(\boldsymbol{x}) = [-93.940309, 45.952635].$$

So, $\forall x \in \boldsymbol{x}$, $c_3(x) \leq 50$. Thus we do not need to linearize the last constraint.

This example is constructed numerically by using the double precision floating point representation. To simplify the notations, the floating point numbers are rounded to 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:

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

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

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

Hence, using Proposition 5, 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 on the objective function, 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 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 Messine (2002) and in Messine and Touhami (2006), which are based on quadratic formulations.

Remark 4 The expression of the linear program (9) depends on the box $\boldsymbol{x}$. Thus, if $\boldsymbol{x}$ 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 (1993), was not reliable. In Stolfi and de Figueiredo (1997), 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 Messine and Touhami (2006). This affine arithmetic replaces all the floating numbers of the affine form by an interval, see Eq. (10), where the variables in bold indicate the interval variables.

$$\widehat{\mathbf{x}} = \mathbf{x_0} + \sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \tag{10}$$

with $\forall i \in \{0, 1, \dots, n\}$, $\mathbf{x_i} = [\underline{x_i}, \overline{x_i}] \in \mathbb{IR}$ and $\forall i \in \{1, \dots, n\}$, $\epsilon_i \in \boldsymbol{\epsilon}_i = [-1, 1]$.

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

Reliable Affine Form $\longrightarrow$ Interval

$$\begin{aligned} \widehat{\mathbf{x}} &= \mathbf{x_0} + \textstyle\sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \quad \longrightarrow \\ \boldsymbol{x} &= \mathbf{x_0} \oplus \left(\sum_{i=1}^{n} (\mathbf{x_i} \odot [-1, 1])\right). \end{aligned} \tag{11}$$

Affine Form $\longrightarrow$ Reliable Affine Form

$$\begin{aligned} &\widehat{x} = x_0 + \textstyle\sum_{i=1}^{n} x_i\epsilon_i, \qquad \longrightarrow \\ &\forall i \in \{1, 2, \dots, n\},\ \mathbf{x_i} = x_i, \\ &\widehat{\mathbf{x}} = \mathbf{x_0} + \sum_{i=1}^{n} \mathbf{x_i}\epsilon_i. \end{aligned}$$

Reliable Affine Form $\longrightarrow$ Affine Form

$$\begin{aligned} \widehat{\mathbf{x}} &= \mathbf{x_0} + \textstyle\sum_{i=1}^{n} \mathbf{x_i}\epsilon_i, \quad \longrightarrow \\ &\forall i \in \{1, 2, \dots, n\}, x_i = \mathrm{mid}(\mathbf{x_i}), \\ \widehat{x} &= x_0 + \textstyle\sum_{i=1}^{n} x_i\epsilon_i + \\ &\quad \left(\textstyle\sum_{i=1}^{n} \max(\overline{x_i \ominus \mathbf{x_i}}, \overline{\mathbf{x_i} \ominus x_i})\right) \epsilon_{\pm}. \end{aligned} \tag{12}$$

Interval $\longrightarrow$ Reliable Affine Form

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

In this Reliable Affine Arithmetic, all the affine operations are done as for the standard affine arithmetic but using properly rounded interval arithmetic (Moore 1966) to ensure its reliability. In Messine and Touhami (2006), 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 Stolfi and de Figueiredo (1997), for finding that linearization, which minimizes the range of a monotonic 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 Eq. (6) and Fig. 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 ($\boldsymbol{d}, \boldsymbol{\alpha}, \boldsymbol{\zeta}$ and $\boldsymbol{\delta}$),
2: $\mathsf{f}$ := natural interval extension of $f$, $\mathsf{f}'$ := natural interval extension of the first derivative of $f$,
3: $\widehat{\mathbf{x}}$ := the reliable affine form under study, $\boldsymbol{x}$ := the interval corresponding to $\widehat{\mathbf{x}}$ see Eq. (11)] ,
4: $\widehat{\mathbf{f}}(\widehat{\mathbf{x}})$ := the reliable affine form of $f$ over $\widehat{\mathbf{x}}$.
5: $\begin{cases} \text{If } \left(\overline{\mathsf{f}'(\boldsymbol{x})} \leq 0\right), \ \boldsymbol{\alpha} := \overline{\mathsf{f}'(\boldsymbol{x})}, \ \boldsymbol{d} := \left[\inf(\mathsf{f}(\boldsymbol{x}) \ominus \boldsymbol{\alpha} \odot \overline{x}), \sup(\mathsf{f}(\boldsymbol{x}) \ominus \boldsymbol{\alpha} \odot \underline{x})\right]. \\ \text{If } f \text{ is constant}, \ \boldsymbol{\alpha} := 0, \quad \boldsymbol{d} := \boldsymbol{x}. \\ \text{If } \left(\underline{\mathsf{f}'(\boldsymbol{x})} \geq 0\right), \ \boldsymbol{\alpha} := \underline{\mathsf{f}'(\boldsymbol{x})}, \ \boldsymbol{d} := \left[\inf(\mathsf{f}(\boldsymbol{x}) \ominus \boldsymbol{\alpha} \odot \underline{x}), \sup(\mathsf{f}(\boldsymbol{x}) \ominus \boldsymbol{\alpha} \odot \overline{x})\right]. \end{cases}$
6: $\boldsymbol{\zeta} := \text{mid}(\boldsymbol{d})$,
7: $\boldsymbol{\delta} := \max\left(\sup(\overline{\zeta} \ominus \boldsymbol{d}), \sup(\boldsymbol{d} \ominus \underline{\zeta})\right)$,
8: $\widehat{\mathbf{f}}(\widehat{\mathbf{x}}) := \boldsymbol{\zeta} + \boldsymbol{\alpha} \cdot \widehat{\mathbf{x}} + \boldsymbol{\delta}\epsilon_{\pm}$.
```

Remark 5 The arithmetic in Algorithm 1 is a particular case of *the generalized interval arithmetic* introduced by E. Hansen in Hansen (1975). Hansen's generalized arithmetic 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 Hansen (1975). 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 Sect. 3, we apply Proposition 2 and 3 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 Sect. 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 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 (Jansson 2003; Keil 2006) 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 5 and Corollary 1; (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 Eq. (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 4 in a reliable way. Then, we use a result from Neumaier and Shcherbina (2004) to compute a reliable lower bound of our linear program or a reliable certificate of infeasibility. This method applied to our case yields:

$$\begin{cases} \min_{\lambda \in \mathbb{R}^m_+, u, l \in \mathbb{R}^n_+} & b^T\lambda + \sum_{i=1}^{n} (l_i + u_i) \\ s.t. & A^T\lambda - l + u = -c. \end{cases} \tag{13}$$

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 $(\mathbf{\Lambda_S}, \mathbf{L_S}, \mathbf{U_S})$ is the extension of (λ_S, l_S, u_S) into interval arithmetic, i.e. $\mathbf{\Lambda_S} = [\lambda_S]$, $\mathbf{L_S} = [l_S]$ and $\mathbf{U_S} = [u_S]$. This conversion makes it possible to perform all computations using rounded interval arithmetic. Then, we can compute the residual of the dual (13) by interval arithmetic, such as:

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

Hence, using the bounds calculated in Neumaier and Shcherbina (2004), we have:

$$\forall y \in S_2, c^T y \in \left(\mathbf{R}^T\boldsymbol{\epsilon} \ominus \mathbf{\Lambda_S}^T[-\infty, b] \oplus \mathbf{L_S}^T\boldsymbol{\epsilon} \ominus \mathbf{U_S}^T\boldsymbol{\epsilon}\right), \tag{15}$$

where $\boldsymbol{\epsilon} = ([-1, 1], \ldots, [-1, 1])^T$ and $[-\infty, b] = ([-\infty, b_1], \ldots, [-\infty, b_m])^T$.

Proposition 8 *Let (λ_S, l_S, u_S) be an approximate solution which minimizes (13), the dual of the linear program (9). Let $\mathbf{\Lambda_S} = [\lambda_S]$, $\mathbf{L_S} = [l_S]$ and $\mathbf{U_S} = [u_S]$. Then,*

$$\forall x \in S_1, f(x) \geq inf\left(\mathbf{R}^T\boldsymbol{\epsilon} \ominus \mathbf{\Lambda_S}^T[-\infty, b] \oplus \mathbf{L_S}^T\boldsymbol{\epsilon} \ominus \mathbf{U_S}^T\boldsymbol{\epsilon} \oplus E_f\right).$$

Proof The result is obtained by applying Eqs. (14), (15) and Proposition 5. □

When the bounds cannot be computed, the dual program (13) can be unbounded or infeasible. If it is the case, the primal (9) must be infeasible or unbounded. Since the feasible set of the primal (9) is included in the bounded set $[-1, 1]^n$, it must be infeasible. Indeed, to prove that the dual (13) is unbounded, we look for a feasible solution of the constraint satisfaction problem (16) (it is a well-known method directly adapted from Neumaier and Shcherbina 2004), since such a solution provides an unbounded direction:

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

Proposition 9 *Let (λ_c, l_c, u_c) be the approximate solution of the constraint satisfaction problem (16). Let $\mathbf{\Lambda_c} = [\lambda_c]$, $\mathbf{L_c} = [l_c]$ and $\mathbf{U_c} = [u_c]$.*

If $0 \notin \left(\left(A^T\mathbf{\Lambda_c} \ominus \mathbf{L_c} \oplus \mathbf{U_c}\right)^T\boldsymbol{\epsilon} \ominus \mathbf{\Lambda_c}^T[-\infty, b] \oplus \mathbf{L_c}^T\boldsymbol{\epsilon} \ominus \mathbf{U_c}^T\boldsymbol{\epsilon}\right)$ 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 $0 \notin \left(\left(A^T \mathbf{\Lambda_c} \ominus \mathbf{L_c} \oplus \mathbf{U_c}\right)^T \boldsymbol{\epsilon} \ominus \mathbf{\Lambda_c}^T[-\infty, b] \oplus \mathbf{L_c}^T \boldsymbol{\epsilon} \ominus \mathbf{U_c}^T \boldsymbol{\epsilon}\right)$, then the primal program (9) is guaranteed to be infeasible. Thus by applying Corollary 1, Proposition 9 is proven. □

Indeed, using Propositions 8 and 9, 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 (Messine 2005; Ninin 2010). The general principle is described in Algorithm 2. Note that there exist other algorithms based on interval arithmetic such as for example *GlobSol*, developed by Kearfott (1996). The fundamental principle is still the same, except that different acceleration techniques are used.

Algorithm 2 Interval branch and bound algorithm: IBBA

```
1: $\boldsymbol{x}$ := initial hypercube in which the global minimum is searched, $\{\boldsymbol{x} \subseteq \mathbb{R}^n\}$
2: $\tilde{f} := \infty$, denotes the current upper bound on the global minimum value,
3: $\mathcal{L} := \{(\boldsymbol{x}, -\infty)\}$, initialization of the data structure of stored elements,
   {all elements in $\mathcal{L}$ have two components: a box $z$ and $f_z$, a lower bound of $f(z)$}
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 midpoint, to get $z_1$ and $z_2$,
7:   for $j := 1$ to 2 do
8:     Pruning of $z_j$ by a Constraint Propagation Technique, see Messine 2004,
9:     if $z_j$ is not empty then
10:      Compute $f_{z_j}$, a lower bound of $f(z_j)$, and all the lower and upper bounds of all the constraints
         over $z_j$,
11:      if $\left(\tilde{f} - \varepsilon_f \max(|\tilde{f}|, 1) \geq f_{z_j}\right)$ and no constraint is unsatisfied then
12:        Insert $(z_j, f_{z_j})$ into $\mathcal{L}$,
13:        $\tilde{f} := \min(\tilde{f}, f(\mathrm{mid}(z_j)))$, if and only if $\mathrm{mid}(z_j)$ satisfies all the constraints,
14:        if $\tilde{f}$ is modified then
15:          $\tilde{x} := \mathrm{mid}(z_j)$,
16:          Discard from $\mathcal{L}$ all the pairs $(z, f_z)$ which $\left(f_z > \tilde{f} - \varepsilon_f \max(|\tilde{f}|, 1)\right)$ is checked,
17:        end if
18:      end if
19:    end if
20:  end for
21: until $\left(\tilde{f} - \min_{(z, f_z) \in \mathcal{L}} f_z \leq \varepsilon_f \max(|\tilde{f}|, 1)\right)$ or $(\mathcal{L} == \emptyset)$
```

In Algorithm 2, at each iteration, the domain under study is chosen 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 $\left(\tilde{f} - \varepsilon_f \max(|\tilde{f}\,|, 1) \geq f_{z_j}\right)$ has the advantage of reducing both the *cluster problem* (Du and Kearfott 1994; Schichl et al. 2014) and the excess processing that occurs when an infinity of equivalent non-isolated 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 which depends on the magnitude of $\tilde{f}$. 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 (Hansen and Walster 2004, Kearfott 1996; Messine 2005; Ninin 2010, Ratschek and Rokne 1988).

Remark 6 On a computer, each real number is represented by a flaoting point number. This approximation introduces numerous difficulties to certify numerical solutions provided by an algorithm. Denote the set of floating point numbers by $\mathbb{F}$ and the expressions of f, g_k and h_l in floating point arithmetic by $f^{\mathbb{F}}$, $g_k^{\mathbb{F}}$ and $h_l^{\mathbb{F}}$ respectively. Notice that in Problem (8), if we replace $\mathbb{R}$ by $\mathbb{F}$, in many cases, there will be no floating point number satisfying the equality constraints. That is why the constraints must be relaxed. Hence, optimization codes have to deal with the following problem:

$$\left\{\begin{array}{lll} \min\limits_{x \in \boldsymbol{x}^{\mathbb{F}} \subset \mathbb{F}^n} & f^{\mathbb{F}}(x) & \\ \text{s.t.} & g_k^{\mathbb{F}}(x) \leq \varepsilon_g, & \forall k \in \{1, \ldots, p\}, \\ & h_l^{\mathbb{F}}(x) \in [-\varepsilon_h, \varepsilon_h], & \forall l \in \{1, \ldots, q\}. \end{array}\right. \tag{17}$$

where ε_g and ε_h are small positive floating point numbers, and the box $\boldsymbol{x}^{\mathbb{F}}$ is the smaller box such that $\boldsymbol{x}$ is included in its convex hull. Thus, by considering Problem (17) over $\mathbb{R}^n$ in place of $\mathbb{F}^n$, we obtain a relaxation of Problem (8). Therefore, at the end of Algorithm 2, it is proven that there is no real vector x satisfying the relaxed constraints such that: $f(x) < \tilde{f} - \varepsilon_f \max(|\tilde{f}\,|, 1)$. Hence, the returning floating point vector $\tilde{x}$ is certified to be a ε_f−global optimum of Problem (17). Notice that Algorithm 2 could not find such a point $\tilde{x}$ if the set defined by the constraints is too small or does not contain any floating point vector. Moreover, using our upper bounding technique, we can find a solution of Problem (17) better and also different from the real one of Problem (8). Nevertheless, notice that the solutions of Problem (17) depend directly on ε_g and ε_h given by the user.

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 z_j; for more details refer to Messine (2004). Another implementation of this method is included in the code *RealPaver* (Granvilliers and Benhamou 2006), the code *Couenne* of the project *COIN-OR* (Belotti et al. 2009), the code IBEX (2014) and the code *GlobSol* (Kearfott Jan 2009). 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 z_1 and z_2, the associated linear program (9) is automatically generated and a linear solver (such as CPLEX) is applied. If the linear program is proved to be 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 z_1 and z_2.

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

```
1: Let $(z, f_z)$ be the current element and $f_z$ a lower bound of $f$ over $z$,
2: Initialize a linear program with the same number of variables as the general problem (8),
3: Generate the affine form of $f$ using AF1 or AF2,
4: Define $c$ the objective function of (9), $E_f$ the lower bound of the error term of the affine form of $f$;
   $c := (\mathsf{f}_1, \dots, \mathsf{f}_n)$, $E_f := \mathsf{f}_0 - \mathsf{f}_{n+1}$ with AF1, or resp. $E_f := \mathsf{f}_0 - \mathsf{f}_{n+1} - \mathsf{f}_{n+3}$ with AF2,
5: for all constraints $g_k$ or resp. $h_l$ of the general problem (8) do
6:   Calculate $\mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z)$, (e.g., using the natural interval extension inclusion function),
7:   if the constraint $\mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z)$ is proved to be infeasible then
8:     Eliminate the element $(z, f_z)$
9:     Exit Algorithm 3
10:  end if
11:  if $\varepsilon_g \in \mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z) \nsubseteq [-\varepsilon_h, \varepsilon_h]$ then
12:    Generate the affine form of $g_k$ or resp. $h_l$ using AF1 or AF2,
13:    Add the associated linear constraint(s) into the linear program (9), such as described in Section 3,
14:  end if
15: end for
16: Solve the linear program (9) with a linear solver such as CPLEX,
17: if the linear program has a solution $y_{sol}$ then
18:   $f_z := \max(f_z, c^T y_{sol} + E_f)$,
19: else if the linear program is infeasible then
20:   Eliminate the element $(z, f_z)$
21: end if
```

Remark 7 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 solution by reducing the number of iterations and the computation time of Algorithm 2. At Line 11 of Algorithm 3, Proposition 7 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 18 provides a lower bound for the general problem over a particular box by using Proposition 5. Corollary 1 involves the elimination part which corresponds to Line 20. 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 8 Affine arithmetic cannot be used when in an intermediate node of the computation tree, the resulting interval is unbounded. For example $\{\min_{x\in[-1,1]} \frac{1}{x}$ s.t. $x^2 \geq 1/4\}$, it is impossible to construct a linearization of the objective 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 Sect. 4, we have explained how the affine reformulation technique can be reliable and rigorous. Algorithm 4 summarizes this method named $rART_{\mathrm{rAF}}$ 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 Eq. (12), Propositions 2 and 3. Then, Proposition 7 is used in Line 11 of Algorithm 4 to reduce the number of constraints. Thus, in most cases, the number of added constraints is small, and the dual solution 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 8 guarantees a lower bound of the objective function, Line 20 of Algorithm 4. Otherwise, if the solver returns that the dual is unbounded or infeasible, Proposition 9 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. $rART_{\mathrm{rAF}}$ (Algorithm 4) included in *IBBA* (Algorithm 2) allows us to take rounding errors into account 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 (Neumaier Neumaier; Neumaier et al. 2005). 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 (Messine 2002; Ninin 2010); however square root, inverse, logarithm and exponential functions are included, using Algorithm 1. For all 74 test problems, no symbolic reformulation has been done. The expressions of the equations are exactly the same as those provided 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 and compiled using the `f90` ORACLE compiler which includes a library for interval arithmetic. In order to solve the linear programming relaxation, CPLEX 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: $\tilde{f} - \min_{(z, f_z)\in\mathcal{L}} f_z \leq \varepsilon_f \max(|\tilde{f}|, 1)$.

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

```
1: Let $(z, f_z)$ be the current element and $f_z$ a lower bound of $f(z)$,
2: Initialize a linear program with the same number of variables as the general problem (8),
3: Generate the reliable affine form of $f$ using rAF1 or rAF2,
4: Using the conversion rAF1/AF1 or rAF2/AF2, generate $E_f$ and $c$ of the linear program (9),
5: for all constraints $g_k$ or resp. $h_l$ of the general problem (8) do
6:   Calculate $\mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z)$, (e.g, using the natural interval extension inclusion function),
7:   if the constraint $\mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z)$ is proved to be infeasible then
8:     Eliminate the element $(z, f_z)$
9:     Exit of Algorithm 4
10:  end if
11:  if $\varepsilon_g \in \mathsf{g}_k(z)$ or resp. $\mathsf{h}_l(z) \nsubseteq [-\varepsilon_h, \varepsilon_h]$ then
12:    Generate the affine form of $g_k$ or resp. $h_l$ using rAF1 or rAF2 forms and conversions rAF1/AF1 or
       rAF2/AF2,
13:    Add the associated linear constraint(s) into the linear program (9), such as described in Section 3,
14:  end if
15: end for
16: Generate the dual program (13) of the corresponding linear program (9),
17: Solve the dual (13) with a primal linear solver,
18: if the dual program has a solution $(\lambda_S, l_S, u_S)$ then
19:   $\mathbf{\Lambda_S} := [\lambda_S]$, $\mathbf{L_S} := [l_S]$ and $\mathbf{U_S} := [u_S]$,
20:   $f_z := \max\left(f_z, \inf\left(\mathbf{R}^T\boldsymbol{\epsilon} \ominus \mathbf{\Lambda_S}^T[-\infty, b] \oplus \mathbf{L_S}^T\boldsymbol{\epsilon} \ominus \mathbf{U_S}^T\boldsymbol{\epsilon} \oplus E_f\right)\right)$,
21: else
22:   Solve the program (16) associated with the dual (13),
23:   if program (16) has a solution $(\lambda_c, l_c, u_c)$ then
24:     $\mathbf{\Lambda_c} := [\lambda_c]$, $\mathbf{L_c} := [l_c]$ and $\mathbf{U_c} := [u_c]$,
25:     if $0 \notin \left(\left(A^T\mathbf{\Lambda_c} \ominus \mathbf{L_c} \oplus \mathbf{U_c}\right)^T\boldsymbol{\epsilon} \ominus \mathbf{\Lambda_c}^T[-\infty, b] \oplus \mathbf{L_c}^T\boldsymbol{\epsilon} \ominus \mathbf{U_c}^T\boldsymbol{\epsilon}\right)$ then
26:       Eliminate the element $(z, f_z)$
27:     end if
28:   end if
29: end if
```

This relative accuracy of the objective function is fixed to $\varepsilon_f = 10^{-8}$ for all the problems and the accuracies of the constraints are $\varepsilon_g = 10^{-8}$ and $\varepsilon_h = 10^{-8}$. The accuracy to solve the linear program by CPLEX is fixed to 10^{-8} and we limit the number of iterations of a run of CPLEX 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 millions (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 names 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 Library 2 of COCONUT corresponds to Example 3 when the box is $\boldsymbol{x} = [1, 5]^4$ and the constraints are only c_1 and c_2.

Table 1 Numerical results for reliable IBBA based methods

Name	N	M	IBBA + CP			IBBA + $rART_{\text{rAF2}}$			IBBA + $rART_{\text{rAF2}}$ + CP		
			Ok?	Iter	t(s)	Ok?	Iter	t(s)	Ok?	Iter	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

Table 1 continued

Name	N	M	IBBA + CP			IBBA + $rART_{\mathrm{rAF2}}$			IBBA + $rART_{\mathrm{rAF2}}$ + CP		
			Ok?	Iter	t(s)	Ok?	Iter	t(s)	Ok?	Iter	t(s)
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 2 Comparison with *GlobSol* approach Kearfott 2006 and our approach

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 3 Numerical results for non reliable but exact global optimization methods

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

Table 3 continued

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)
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	160.24	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

For all tables and performance profiles, *IBBA+CP* indicates results obtained with Algorithm 2 (IBBA) and the constraint propagation technique (CP) described in Messine (2004). $IBBA + rART_{\mathrm{rAF2}}$ 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 (Messine 2002;Ninin 2010; Messine and Touhami 2006). $IBBA + rART_{\mathrm{rAF2}}$ *+CP* represents results obtained with Algorithm 2 and both acceleration techniques. *GlobSol+LR* and *GlobSol* represent the results extracted from Kearfott (2006) and obtained using (or not) the linear relaxation based on RLT (Kearfott and Hongthong 2005).

The performance profiles, defined by Dolan and Moré (2002), are visual tools to benchmark algorithms. Thus, Tables 1, 2 and 3 are summarized in Fig. 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 functions for the ratio, are computed.

Remark 9 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.

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, $IBBA + rART_{\mathrm{rAF2}}$ 52 test problems and $IBBA + rART_{\mathrm{rAF2}}$ *+CP* 61 test problems.
- The solved cases are not the same using the two distinct techniques (*CP* or $rART_{\mathrm{rAF2}}$). 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 $IBBA + rART_{\mathrm{rAF2}}$ 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 9 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 $IBBA + rART_{\mathrm{rAF2}}$, but is divided by a factor of about 10 when those two techniques are combined. Considering the number of iterations, the gain of $IBBA + rART_{\mathrm{rAF2}}$ *+CP* is a factor of about 200 compared to *IBBA+CP*, and about 3.5 compared to $IBBA + rART_{\mathrm{rAF2}}$.

The performance profiles of Fig. 3 confirm that $IBBA+rART_{\mathrm{rAF2}}$ *+CP* is the most efficient and effective of the three first studied algorithms. Considering the curve of the algorithms $IBBA + rART_{\mathrm{rAF2}}$ and *IBBA+CP* shows that *IBBA+CP* is in general

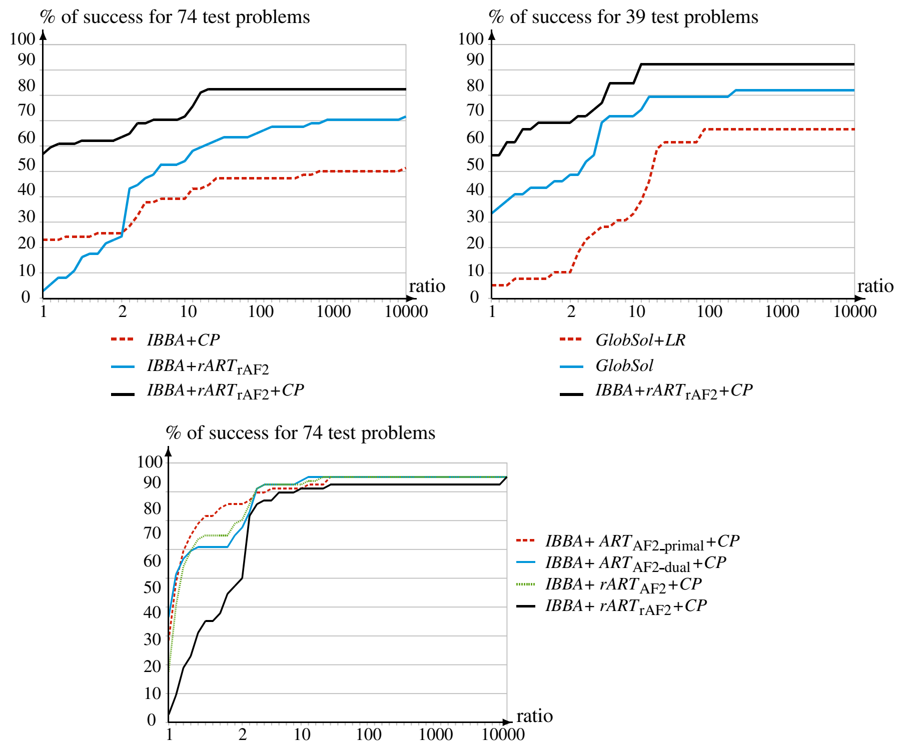

Fig. 3 Performance profile comparing the results of various versions of algorithms

faster than the other but $IBBA + rART_{\mathrm{rAF2}}$ solves more problems, which implies a crossing of the two curves.

By observing how the acceleration techniques work on a box, the reformulation $rART_{\mathrm{rAF2}}$ 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 $rART_{\mathrm{rAF2}}$ is so efficient: *CP* reduces quickly the size of boxes and then $rART_{\mathrm{rAF2}}$ considerably improves 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 Neumaier and found by the algorithm of the column 'Algorithm'. We remark that all 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 (2005) 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 (Tawarmalani and Sahinidis 2004). Another paper by Kearfott (2006) studies its integration into an interval branch and bound algorithm named *GlobSol*. In Kearfott (2006), the termination criteria of the branch and bound code are not exactly the same for *GlobSol* and $IBBA+rART_{\text{rAF2}}+CP$. The stopping criterion of *GlobSol* ensures enclosures of all exactly optimizing points. This difference leads to favor *IBBA*. Thus, this empirical comparison between *GlobSol* and $IBBA+rART_{\text{rAF2}}+CP$ should be considered as a first approximation. Moreover, (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 Kearfott (2006) is not the last one, and (iii) it is the first version of $IBBA+rART_{\text{rAF2}}+CP$ 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 $IBBA+rART_{\text{rAF2}}+CP$ solves 36.
- 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 $IBBA+rART_{\text{rAF2}}+CP$ have at most 24 variables and 17 constraints.
- *GlobSol* without *LR* solved 1 problem in 53 minutes that $IBBA+rART_{\text{rAF2}}+CP$ does not solve in 60 minutes (ex6_1_1). $IBBA+rART_{\text{rAF2}}+CP$ 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 Fig. 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 $IBBA+rART_{\text{rAF2}}+CP$ 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_{AF2} techniques. Thus, we test for all three cases an IBBA algorithm associated with the *CP* and ART_{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 $IBBA+ART_{\text{AF2\_primal}}+CP$ and the dual formulation for the column $IBBA+ART_{\text{AF2\_dual}}+CP$. In the third columns $IBBA+rART_{\text{AF2}}+CP$, 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 $IBBA + rART_{\text{rAF2}}$ $+CP$, 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 Fig. 3, the primal formulation seems to be more efficient. Indeed up to 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 $IBBA + ART_{\text{AF2\_primal}}$ $+CP$. 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 $IBBA + ART_{\text{AF2\_primal}}$ $+CP$ even if it is generally the most efficient (see Fig. 3). In fact, it appears that $IBBA + ART_{\text{AF2\_primal}}$ $+CP$ 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).
- The increasing CPU-time to obtain reliable computations is about a factor of 2, see last line of Tables 3 and 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 this method is that the number of variables in the linear programs generated is the same as in the original optimization problem. Indeed, the linear programs require short times to be solved. Moreover, when the width 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

Androulakis IP, Maranas CD, Floudas CA, Alpha BB (1995) A global optimization method for general constrained nonconvex problems. J Global Optim 7(4):337–363

Audet C, Hansen P, Jaumard B, Savard G (2000) A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. Mathe Program Ser A 87(1):131–152

Belotti P, Lee J, Liberti L, Margot F, Waechter A (2009) Branching and bounds tightening techniques for non-convex MINLP. Optim Methods Softw 24(4–5):597–634

Comba JLD, Stolfi J (1993) Affine arithmetic and its applications to computer graphics. In: Proceedings of SIBGRAPI'93 - VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pp 9–18

de Figueiredo L (1996) Surface intersection using affine arithmetic. In: Proceedings of graphics interface'96, pp 168–175

de Figueiredo L, Stolfi J (2004) Affine arithmetic: concepts and applications. Numer Algorithms 37(1–4):147–158

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program Ser A 91(2):201–213

Du K, Kearfott RB (1994) The cluster problem in multivariate global optimization. J Global Optim 5(3):253265

Fitan E, Messine F, Nogarède B (2004) The electromagnetic actuator design problem: a general and rational approach. IEEE Trans Magn 40(3):1579–1590

Fontchastagner J, Messine F, Lefevre Y (2007) Design of electrical rotating machines by associating deterministic global optimization algorithm with combinatorial analytical and numerical models. IEEE Trans Magn 43(8):3411–3419

Granvilliers L, Benhamou F (2006) RealPaver: an interval solver using constraint satisfaction techniques. ACM Trans Mathe Softw 32:138156

Hansen ER (1975) A generalized interval arithmetic. Lect Notes Comput Sci 29:7–18

Hansen ER, Walster WG (2004) Global optimization using interval analysis, 2nd edn. Marcel Dekker Inc., New York

Horst R, Tuy H (1996) Global optimization: deterministic approaches, 3rd edn. Springer, Berlin

Jansson C (2003) Rigorous lower and upper bounds in linear programming. SIAM J Optim 14(3):914–935

IBEX (2014) a C++ numerical library based on interval arithmetic and constraint programming. http://www.ibex-lib.org

Kearfott RB (1996) Rigorous global search: continuous problems. Kluwer Academis Publishers, Dordrecht

Kearfott RB (2006) Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. Optim Methods Softw 21(5):715–731

Kearfott RB (Jan 2009) GlobSol user guide. Optim Methods Softw 24(4–5):687–708

Kearfott RB, Hongthong S (2005) Validated linear relaxations and preprocessing: some experiments. SIAM J Optim 16(2):418–433

Keil C (2006) Lurupa - Rigorous Error Bounds in Linear Programming. In: Buchberger B, Oishi S, Plum M, Rump SM (eds) Algebraic andNumerical Algorithms and Computer-assisted Proofs. Dagstuhl Seminar Proceedings 05391. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany

Lasserre J-B (2001) Global optimization with polynomials and the problem of moments. SIAM J Optim 11(3):796–817

Lebbah Y, Michel C, Rueher M (2005) Efficient pruning technique based on linear relaxations. In: Proceedings of global optimization and constraint satisfaction, vol 3478, pp 1–14

Maranas CD, Floudas CA (1997) Global optimization in generalized geometric programming. Comput Chem Eng 21(4):351–369

Markot MC, Fernandez J, Casado LG, Csendes T (2006) New interval methods for constrained global optimization. Math Program 106(2):287–318

Messine F (2002) Extensions of affine arithmetic: application to unconstrained global optimization. J Univ Comput Sci 8(11):992–1015

Messine F (2004) Deterministic global optimization using interval constraint propagation techniques. RAIRO Oper Res 38(4):277–293

Messine F (2005) A deterministic global optimization algorithm for design problems. In: Audet C, Hansen P, Savard G (eds) Essays and Surveys in Global Optimization. Springer, New York, pp 267–294

Messine F, Nogarède B, Lagouanelle J-L (1998) Optimal design of electromechanical actuators: a new method based on global optimization. IEEE Trans Magn 34(1):299–308

Messine F, Touhami A (2006) A general reliable quadratic form: an extension of affine arithmetic. Reliable Comput 12(3):171–192

Mitsos A, Chachuat B, Barton PI (2009) McCormick-based relaxations of algorithms. SIAM J Optim 20(2):573–601

Moore RE (1966) Interval analysis. Prentice-Hall Inc., Englewood Cliffs

Neumaier A, Shcherbina O (2004) Safe bounds in linear and mixed-integer linear programming. Math Program Ser A 99(2):283–296

Neumaier A, Shcherbina O, Huyer W, Vinko T (2005) A comparison of complete global optimization solvers. Math Program Ser B 103(2):335–356

Ninin J (2010) Optimisation Globale basé sur l'Analyse d'Intervalles: relaxation affine et limitation de la mémoire. PhD thesis, Institut National Polytechnique de Toulouse

Perron S (2004) Applications jointes de l'optimisation combinatoire et globale. PhD thesis, École Polytechnique de Montréal

Ratschek H, Rokne J (1988) New computer methods for global optimization. Ellis Horwood Ltd, Chichester

Schichl H, Markót MC, Neumaier A (2014) Exclusion regions for optimization problems. J Global Optim 59(2–3):569–595. doi:10.1007/s10898-013-0137-z

Schichl H, Neumaier A (2005) Interval analysis on directed acyclic graphs for global optimization. J Global Optim 33(4):541–562

Shcherbina O, Neumaier A, Sam-Haroud D, Vu X-H, Nguyen T-V (2003) Set of test problems COCONUT, Benchmarking global optimization and constraint satisfaction codes. Springer.http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html

Sherali HD, Adams WP (1999) A reformulation-linearization technique for solving discrete and continuous nonconvex problems. Kluwer Academis Publishers, Dordrecht

Stolfi J, de Figueiredo L (1997) Self-validated numerical methods and applications. Monograph for 21st Brazilian Mathematics Colloquium. IMPA/CNPq, Rio de Janeiro, Brazil

Tawarmalani M, Sahinidis NV (2004) Global optimization of mixed-integer nonlinear programs: a theoretical and computational study. Math Program Ser A 99(3):563–591

Van Hentenryck P, Michel L, Deville Y (1997) Numerica: a modelling language for global optimization. MIT Press, Massachusetts

Vu X-H, Sam-Haroud D, Faltings B (2009) Enhancing numerical constraint propagation using multiple inclusion representations. Ann Math Artif Intell 55(3–4):295–354