Hybridizing two linear relaxation techniques in interval-based solvers

Ignacio Araya, Frédéric Messine, Jordan Ninin, Gilles Trombettoni

Journal of Global Optimization
https://doi.org/10.1007/s10898-024-01449-2

Hybridizing two linear relaxation techniques in interval-based solvers

Ignacio Araya[1] · **Frédéric Messine**[2] · **Jordan Ninin**[3] · **Gilles Trombettoni**[4]

Received: 28 August 2023 / Accepted: 20 October 2024

Abstract
In deterministic global optimization, techniques for linear relaxation of a non-convex program are used in the lower bound calculation phase. To achieve this phase, most deterministic global optimization codes use reformulation-linearization techniques. However, there exist also two interval-based polyhedral relaxation techniques which produce reliable bounds without adding new auxiliary variables, and which can take into account mathematical operations and most transcendental functions: (i) the affine relaxation technique, used in the IBBA code, based on affine forms and affine arithmetic, and (ii) the extremal Taylor technique, used in the Ibex-Opt code, which is based on a specific interval-based Taylor form. In this paper, we describe how these two interval-based linear relaxation techniques can be hybridized. These two approaches appear to be complementary, and such a hybrid method performs well on a representative sample of constrained global optimization instances.

Keywords Interval analysis · Global optimization · Linear relaxation · Reliable computing · Branch-and-bound

1 Introduction

Lower bounding appears to be a crucial phase for solving constrained global optimization problems in a deterministic way. This ensures that no feasible point exists with an objective function value inferior to a provided lower bound. Linear relaxation techniques are one of the most efficient acceleration routines in the lower-bounding phase.

✉ Frédéric Messine
frederic.messine@laplace.univ-tlse.fr

Ignacio Araya
ignacio.araya@ucv.cl

Jordan Ninin
jordan.ninin@ensta-bretagne.fr

Gilles Trombettoni
gilles.trombettoni@lirmm.fr

1 Pontificia Universidad Católica de Valparaíso, Valparaiso, Chile

2 University of Toulouse, ENSEEIHT-LAPLACE, CNRS, Toulouse, France

3 ENSTA-Bretagne, Lab-STICC, team MATRIX, Brest, France

4 University of Montpellier, CNRS, Montpellier, France

Published online: 27 November 2024

Polyhedral (linear) relaxations are the most commonly used convex relaxations techniques and *reformulation* is the most common way to achieve polyhedral relaxations [39]. Roughly, the reformulation-linearization techniques replace recursively each nonlinear term in mathematical expressions with an auxiliary variable and a nonlinear equality constraint. The feasible domain created by each new equation is then linearly relaxed by one or several linear constraints. Thus, the nonlinear system is relaxed to a polytope including the linearized constraints and the constraints corresponding to the nonlinear terms. Finding the minimum value of the linearized objective function subject to this polytope can improve the lower bound. Reformulation methods compute generally a sharp approximation of the feasible domain leading to a smaller number of branching nodes (iterations) in the tree expanded by the branch-and-bound algorithm. Unfortunately, these relaxation techniques can rarely guarantee that the computed polytope contains the whole feasible domain due to rounding errors performed by the computations with floating-point numbers.

An increasing number of global optimization algorithms based on interval arithmetic is now available [16, 19, 23, 26, 36, 42, 44]. These interval-based methods take into account numerical errors due to calculations with floating-point numbers and thus, can compute the global solution with a guaranteed error. However, because developing reliable reformulation-linearization techniques is challenging, only a few interval-based strategies use such linear relaxations. The `Icos` solver proposes the `Quad` operator [22] where only quadratic and bilinear operators are rigorously relaxed by linear constraints. A version of the `GlobSol` solver [19] also uses a rigorous polyhedral relaxation described in [18].

Two interval-based polyhedral relaxation techniques also exist for producing reliable bounds: (i) Affine Relaxation Technique (ART), used in the `IBBA` code [24, 26], which is based on affine forms and affine arithmetic to provide linear under and over-estimations of a function over a box [25, 28, 34, 35];
(ii) eXtremal Taylor (X-Taylor), used in the `IbexOpt` code [42], which is based on a first-order interval Taylor expansion in some box *vertices* [3].

Note that, unlike classical reformulation techniques, interval-based linearization techniques do not introduce new auxiliary variables and constraints. Moreover, these linearization techniques can take into account the four operations and most transcendental functions such as for example exp, log, sin and cos.

The performances of these solvers were shown on numerous examples [34, 35, 42] and on several applications such as the design of electrical motors [26, 27]. In both solvers, the performance mainly comes from two acceleration procedures: the linear relaxation techniques mentioned above and other *contraction* techniques used to reduce the bounds of the domain under study without loss of feasible points. *Constraint propagation* is the most famous contraction technique. Constraint propagation, and especially the HC4 algorithm [7, 24], appears in almost all deterministic global optimization algorithms including `BARON` [41], `COUENNE` [6], `IbexOpt` [42] and `IBBA`, [24, 26, 34, 35]. It has been empirically shown in [35, 42] that associating constraint propagation and interval-based linear relaxation techniques provides very efficient interval global optimization codes.

The main ideas of this paper are: (i) a comparison of the ART and X-Taylor linearization procedures inside the same branch-and-bound `IbexOpt` code, and (ii) a new hybrid linearization operator implemented in `Ibex`.

An important observation is that, for a given non-convex function, X-Taylor achieves a Taylor expansion at a vertex of the box under study, whereas ART performs a better approximation of the function at the center of the box. This may explain why both techniques are complementary in practice and why it is relevant to hybridize them.

Section 2 introduces several notations and definitions related to the handled problem and interval arithmetic tools. In Section 3, ART and X-Taylor are presented. In Section 4, we propose a new contractor hybridizing ART and X-Taylor. In Section 5, we present numerous tests coming from the COCONUT global optimization library, [38]. These numerical results obtained by using the operator implemented in `Ibex` highlight the efficiency of the hybrid method on several hard instances.

2 Background

2.1 Global optimization, interval analysis

In this paper, we address the following *constrained global optimization* problem:

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

where $g : \mathbf{x} \subseteq \mathbb{R}^n \to \mathbb{R}^p$ and $h : \mathbf{x} \subseteq \mathbb{R}^n \to \mathbb{R}^q$. Note that f, g_k and h_l are not all linear and they do not have convex properties. Nevertheless, they have to be continuous, differentiable (at least one time) and defined in a formal way in order that interval and affine arithmetics can be used.

Interval arithmetic enables handling this mathematical problem defined over the real numbers *rigorously*; i.e., they take into account rounding errors implied by floating-point operations. An *interval* $\mathbf{x}_i = [\underline{x_i}, \overline{x_i}]$ defines the set of real numbers x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. A *box* $\mathbf{x}$ is a Cartesian product of intervals $\mathbf{x}_1 \times \cdots \times \mathbf{x}_n$.

Several interval tools, including `GlobSol` [19] and `Icos` [22], solve problem $(\mathcal{P})$ by returning a small box which guarantees to contain an optimal point.

The codes `IBBA` and `IbexOpt`, used in this paper, handle problem $(\mathcal{P})$ where the equations $h(x) = 0$ are relaxed by inequalities $-\epsilon_{eq} \leq h(x) \leq +\epsilon_{eq}$.[1] In the following, all the equality constraints will be reformulated as above and included in the set of inequality constraints. `IBBA` and `IbexOpt` interval solvers return a floating-point vector that is an ϵ_f-global minimum; i.e., for which the objective function value is optimal modulo an upper bounded error ϵ_f.

Note that most of global optimizers (nonlinear programming solvers) from the mathematical programming community also handle an ϵ_f-optimization problem. This includes the deterministic global optimization codes `Baron` [41], `Lindo` [37], `Antigone` [29], `SCIP` [1], `COUENNE` [6]. However, due to a lack of rigorous computations over the floating-point numbers, they cannot guarantee that the answer has an error bounded by ϵ_f and therefore sometimes these codes could miss the global minimum.

The operator introduced in this paper is implemented in the rigorous C++ interval-based `Ibex` library and is rigorous; i.e., no feasible point can be lost. However, this operator can also be used by any rigorous or non-rigorous NLP solver.

[1] This type of relaxation often occurs in practical problems, e.g. in physics or robotics, where the coefficients of equations are generally known with a bounded uncertainty (e.g., a measured distance), thus providing an ϵ_{eq} specific to a given equation. The relaxation of equations in `IBBA` and `IbexOpt` is motivated by the upper-bounding phase (find a feasible point with an accuracy ϵ_{eq}). For the lower-bounding phase studied in this paper, it is not necessary to relax equations so that the results presented still hold if equality constraints are viewed as two inequalities $0 \leq h(x) \leq 0$.

Finally, note that the operator proposed in this paper does not apply only to optimization problems, but it could also be used to solve feasibility problems. It can be viewed as a *contracting* operator that can reduce the domain of each variable without loss of feasible points.

2.2 Interval branch-and-bound algorithm

We summarize in this section how the interval branch-and-bound (B&B) algorithm behind `IBBA` and `IbexOpt` works. We show an example where the interval-based polyhedral operator proposed in this paper can be used for constrained global optimization problem. The B&B scheme is described by Algorithm 1. Algorithm 2 is dedicated to the contraction of variable intervals.

Algorithm `Interval Branch-and-Bound` $(f, g, \mathbf{x}, \epsilon_f)$
- $f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$; $x_{\widetilde{f}} \leftarrow \emptyset$; $lb_{\mathbf{x}} \leftarrow -\infty$;
- $\mathcal{L} \leftarrow \{(\mathbf{x}, lb_{\mathbf{x}})\}$
- **while** $\mathcal{L} \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_f$ **and** $\frac{\widetilde{f} - f_{min}}{|f|} > \epsilon_f$ **do**
  - $\mathbf{x} \leftarrow$ `BestBox` $(\mathcal{L})$; $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\mathbf{x}\}$
  - $(\mathbf{x}_1, \mathbf{x}_2) \leftarrow$ `Bisect` $(\mathbf{x})$
  - $(\mathbf{x}_1, lb_{\mathbf{x}_1}, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` $(\mathbf{x}_1, f, g, \widetilde{f}, \epsilon_f)$
  - $(\mathbf{x}_2, lb_{\mathbf{x}_2}, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` $(\mathbf{x}_2, f, g, \widetilde{f}, \epsilon_f)$
  - $(f_{min}, \mathcal{L}) \leftarrow$ `UpdateBoxesLB` $((\mathbf{x}_1, lb_{\mathbf{x}_1}), (\mathbf{x}_2, lb_{\mathbf{x}_2}), \mathcal{L}, \epsilon_f)$

Algorithm 1: Interval Branch-and-Bound Algorithm.

The B&B algorithm maintains the following main information during the iterations: the current value of the best solution for f denoted by $\widetilde{f}$ and the corresponding best feasible point $x_{\widetilde{f}}$ found so far, and the lowest lower bound of all the boxes $\mathbf{x}$ stored in $\mathcal{L}$ which is denoted by f_{min}. For every box $\mathbf{x}$, there is a guarantee that no feasible point with a value lower than f_{min} exists.

The algorithm is launched with the set of constraints (g), the objective function f and the initial domain $\mathbf{x}$. ϵ_f is the required precision on the global minimum value.

The procedure `BestBox` selects the next box to handle. The box $\mathbf{x}$ with the lowest lower bound on the objective function is selected, hence the B&B algorithm achieves a best-first search.

The selected box $\mathbf{x}$ is then split into two sub-boxes $\mathbf{x}_1$ and $\mathbf{x}_2$ along one dimension. Several branching heuristics can be used, and our default strategy uses a variant of the Kearfott Smear function described in [2, 19, 42].

Both sub-boxes are then handled by the `Contract&Bound` procedure (see Algorithm 2). A constraint $f(x) \leq \widetilde{f} - \epsilon_f$ is added to the system for decreasing the upper bound of the objective function over the box. The ϵ_f value implies to find a solution significantly better than the current best feasible point. The `Contraction` procedure contracts the handled box without loss of feasible part. In other words, some infeasible parts of the domain are discarded by `HC4` and the convexification algorithms presented in this paper. `Contraction` returns the contracted box and a lower bound of its cost obtained by interval computation, i.e. $lb_{\mathbf{x}} = \underline{\mathbf{f}(\mathbf{x})}$.

Algorithm `Contract&Bound` $(\boldsymbol{x}, f, g, \widetilde{f}, \epsilon_f)$
  $g' \leftarrow g \cup \{f(x) \leq \widetilde{f} - \epsilon_f\}$
  $(\mathbf{x}, lb_{\mathbf{x}}) \leftarrow$ `Contraction` $(\mathbf{x}, f, g')$
  if $\boldsymbol{x} \neq \emptyset$ **then**
    $(x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `FeasibleSearch` $(\mathbf{x}, f, g', \epsilon_f)$ /* Upperbounding */
  return $(\mathbf{x}, lb_{\mathbf{x}}, x_{\widetilde{f}}, \widetilde{f})$

Algorithm 2: The Contract&Bound procedure of the B&B algorithm.

The last part of the procedure carries out upper-bounding. `FeasibleSearch` calls several heuristics searching for a feasible point $x_{\widetilde{f}}$ that improves the best minimum value found $\widetilde{f}$. More precisely, these algorithms extract inner regions inside the feasible domain; i.e., domain in which all points satisfy the inequality and relaxed equality constraints, details can be found in [4]. The last call to `UpdateBoxesLB` in Algorithm 1 pushes the two sub-boxes in the list $\mathcal{L}$ of open nodes if they are sufficiently large and updates f_{min} with $lb_{\mathbf{x}_1}$ and $lb_{\mathbf{x}_2}$.

2.3 Contraction algorithms

In this section, we detail the `Contraction` procedure which runs the polyhedral convexification operators described in this paper.

The `Contraction` procedure is implemented in the Interval-Based EXplorer (`Ibex`) C++ library[2] [8] and used into the global optimization solver called `IbexOpt` [42]. It is very close to the contraction procedure implemented in `IBBA` [35]. In `IbexOpt`, the by default contraction is achieved by `HC4` [7, 24], then by `ACID` [33] and finally by `PolytopeHull`, see Section 2.4.

`HC4` is the classical constraint propagation algorithm mentioned in the introduction. `ACID` is an adaptive constraint programming algorithm enforcing a strong consistency called `CID` [32, 43].[3] `CID` is *variable oriented* and runs a `VarCID` procedure on several variables: `VarCID` splits the domain of a variable into several sub-intervals; another contraction operator, e.g. `HC4`, is launched on each corresponding sub-problem and the hull of the resulting contracted boxes is returned. `ACID` auto-adapts (i.e., limits) the number of variables handled by `VarCID` during the `B&B` algorithm [32].

After the call to `HC4` and `ACID`, the `PolytopeHull` contractor is applied. The last two methods are embedded in a fixed-point process; i.e., they iteratively run until a quasi fixed-point is reached in terms of contraction.

[2] https://github.com/ibex-team/ibex-lib

[3] CID is a consistency slightly stronger than Singleton Arc Consistency (SAC [11]) in finite domains, but limited to the bounds of the domains.

2.4 The polytope hull contraction

The linear relaxation methods studied in this paper are included in a contraction algorithm [5, 21]. Algorithm 3 describes this procedure. The inputs of the procedure `PolytopeHull` are: (i) the objective function f, the vector function g corresponding to the inequalities; (ii) the box $\mathbf{x}$ under study; (iii) a parameter RT, indicating the relaxation technique employed for linearly relaxing the system. This parameter can take the values X-Taylor, ART, or Hyb, corresponding to the techniques detailed in Sections 3.1, 3.2, and 4, respectively. The outputs of this procedure are: (i) the box $\mathbf{x}$ after the contraction, which is necessarily included in the input domain; (ii) $lb_{\mathbf{x}}$, the new lower bound of the objective function over $\mathbf{x}$.

The `PolytopeHull` algorithm constructs first a linear relaxation $g_l(x) \leq 0$ of the constraint system by calling the method `LinearRelaxation`. This method applies the selected linearization technique, as specified by RT, to the vector function g over $\mathbf{x}$. Then, the algorithm improves the bounds of the variables domains without loss of feasible solutions using a linear programming (LP) solver. Then, a linearization of the objective function f_l is generated by using the method `LinearRelaxation`. Finally, the lower bound is improved by minimizing the relaxed objective function.

Algorithm `PolytopeHull` *(**x**, f, g, RT)*
  $g_l \leftarrow$ `LinearRelaxation`(**x**, g, RT)
  for *i from 1 to n* **do**
    /* Two calls to a linear programming algorithm contracts $\mathbf{x}_i$: */
    $\underline{x_i} \leftarrow \min x_i$ s.t. $g_l(x) \leq 0$
    $\overline{x_i} \leftarrow \max x_i$ s.t. $g_l(x) \leq 0$
  $f_l \leftarrow$ `LinearRelaxation`(**x**, f, RT)
  $lb_{\mathbf{x}} \leftarrow \min f_l(x)$ s.t. $g_l(x) \leq 0$ /* Lower-bounding */
  return (**x**, $lb_{\mathbf{x}}$)

Algorithm 3: The `PolytopeHull` algorithm for contracting the domains and improving the lower bound.

The construction of the polytope follows the principles described in Section 3. An LP solver is called twice per variable to improve the bounds. We have implemented the heuristics mentioned in [5] to determine in which order the variables have to be handled. This improvement avoids in practice to solve $2n$ linear programs. Even though, the polytope computation is safe, the floating-point round-off errors made by the LP solver could provide a numerical-feasible solution that is outside the polytope. A cheap post-processing proposed by Neumaier and Shcherbina in [31], using interval arithmetic, has been added to certify that the solutions so-obtained by the LP solver, provide certified lower bounds; note that for a broader approach to solution certification, especially relevant in handling degenerate and/or ill-posed problems, the work of Jansson [17] proposes a more comprehensive method.

3 Linear Relaxation Methods

In deterministic global optimization and more recently in interval-based methods, linear and nonlinear relaxation techniques were proposed to compute lower bounds of problem ($\mathcal{P}$). Two of them were implemented in the solvers `Ibex` [42] and `IBBA` [24, 35] respectively.

These two linear relaxation methods are different and complementary. They are recalled and summarized in the following two subsections.

3.1 Taylor-based linear relaxation technique

In the `Ibex` solver, a linear relaxation technique was derived from an interval Taylor expansion of each constraint, where the expansion point is a vertex (extremal point) of the box $\mathbf{x}$ [3]; this technique is named *X-Taylor*. Thanks to the use of box vertices, this technique generates a convex polyhedral relaxation.

Let v denote a vertex of a box $\mathbf{x}$, where $\mathbf{x}_i = [\underline{x_i}, \overline{x_i}]$ and let $\hat{v}$ denote the opposite vertex of v, i.e. if $v_i = \underline{x_i}$ then $\hat{v}_i = \overline{x}_i$ and vice versa. $V_{\mathbf{x}}$ denotes the set of vertices v of the box $\mathbf{x}$. Considering a general differentiable function $f : \mathbf{x} \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, one has, $\forall x \in \mathbf{x}$ and $\forall v \in V_{\mathbf{x}}$:

$$f(v) + (x - v)^{\top}.\nabla\mathbf{f}^{v}(\mathbf{x}) \leq f(x) \leq f(v) + (x - v)^{\top}.\nabla\mathbf{f}^{\hat{v}}(\mathbf{x}), \tag{1}$$

where the interval vector $\nabla\mathbf{f}(\mathbf{x})$ denotes the enclosure of the gradient of f over $\mathbf{x}$, which can be computed by interval automatic differentiation, and the scalar vector $\nabla\mathbf{f}^{v}(\mathbf{x})$ denotes the extremal value of $\nabla\mathbf{f}(\mathbf{x})$ corresponding to vertex v, i.e. if $v_i = \overline{x_i}$ then $\nabla\mathbf{f}_i^{v}(\mathbf{x}) = \overline{\nabla\mathbf{f}_i(\mathbf{x})}$, and if $v_i = \underline{x_i}$ then $\nabla\mathbf{f}_i^{v}(\mathbf{x}) = \underline{\nabla\mathbf{f}_i(\mathbf{x})}$.

Example 1 Let us introduce the following univariate example:

$$f(x) = 3x^3 - 2\left(x + \frac{1}{2}\right)^2 + 2x + 1, \text{ with } x \in \mathbf{x} = [0, 1].$$

By computing an enclosure of the first derivative estimated on the interval $[0, 1]$, one obtains: $\nabla\mathbf{f}(\mathbf{x}) = [-4, 9]$. Indeed, $f'(x) = 9x^2 - 4x$ and by evaluating the expression f' on the interval $[0, 1]$ the result directly follows.

Hence, by using the vertex $v = 0$, one has:

$$\forall x \in [0, 1],\ \frac{1}{2} - 4x \leq f(x) \leq \frac{1}{2} + 9x,$$

and with the vertex $v = 1$, one obtains:

$$\forall x \in [0, 1],\ \frac{3}{2} + 9(x - 1) = -\frac{15}{2} + 9x \leq f(x) \leq \frac{3}{2} - 4(x - 1) = \frac{11}{2} - 4x.$$

In Figure 1, the linear hull is drawn by taking into account these 4 hyperplanes (here blue lines).

Therefore, by relaxing the functions of problem $(\mathcal{P})$ by using X-Taylor using the set of vertices $|V_{\mathbf{x}}|$ of the box $\mathbf{x}$ as expansion points, we obtain the following linear relaxation:

$$(\mathcal{P}_{XT}) \begin{cases} \min\limits_{\substack{x \in \mathbf{x} \subseteq \mathbb{R}^n \\ z \in \mathbb{R}}} z \\ \text{s.t.} \quad f(v) + (x - v)^{\top}\nabla\mathbf{f}^{v}(\mathbf{x}) \leq z, \text{ with } v \in V_{\mathbf{x}}, \\ \qquad g_k(v) + (x - v)^{\top}\nabla\mathbf{g}_k^{v}(\mathbf{x}) \leq 0, \forall k \in \{1, \ldots, p\}, \text{ with } v \in V_{\mathbf{x}}. \end{cases}$$

where $\nabla\mathbf{f}(\mathbf{x})$ is a vector of enclosures of the partial derivative $\frac{\partial f}{\partial x_i}(x)$ over the box $\mathbf{x}$. A scalar vector $\mathbf{a}^{v}$ denotes the extremal value of $\mathbf{a}$ related to vertex v, for instance, if $v = \{\underline{x_1}, \overline{x_2}, \overline{x_3}\}$, then $\mathbf{a}^{v} = \{\underline{a_1}, \overline{a_2}, \overline{a_3}\}$.

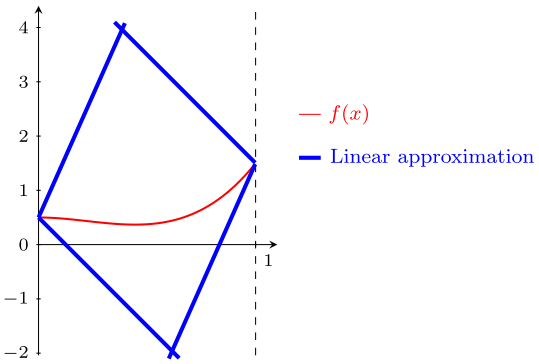

Fig. 1 X-Taylor linear relaxation technique

Choice of vertices

Since the box studied contains 2^n vertices (i.e., $|V_{\mathbf{x}}| = 2^n$), we can select one or several linear underestimations of g_k among the 2^n ones. Providing a linear relaxation using the 2^n hyperplanes for each function is computationally expensive. Several heuristics for selecting a subset of the 2^n vertices were proposed in [3]. It appears that selecting a vertex that tries to minimize the volume lost by the corresponding linear relaxation is not the most significant criterion.

Instead, a simple policy has been kept in the distributed code: for each inequality, we generate two hyperplanes by selecting two opposite vertices: a vertex v is randomly selected and its opposite vertex $\hat{v}$ is deduced. The reason why this simple approach is efficient in practice is because two opposite vertices touch every face of the studied box/domain. We know that an (interval or not) Taylor form is exact at the selected expansion point. In X-Taylor, the possible expansion points are the vertices of the box studied. Therefore, a constraint relaxed twice at two opposite vertices using X-taylor is likely to be well approximated in the $2n$ faces of the box. However, all the contraction methods used by interval solvers including IbexOpt and IBBA (see Sections 2.3 and 2.4) remove infeasible points from the bounds of the box studied; i.e., focus on the $2n$ faces of the box.

Despite the non-deterministic nature of this approach, significant variations in CPU time between runs have not been observed. This consistency is attributed to the comparable effectiveness of contraction methods across different sets of randomly selected vertices. Consequently, for the purposes of experimental analysis, it is considered sufficient to conduct only one run per instance to achieve a thorough evaluation.

Management of rounding errors

Finally, considering numerical/floating-point values issues, we have to take care about the real coefficients in the linear expressions. Using *rounded interval arithmetic* [30], the interval enclosure of the partial derivatives is carefully and reliably computed, as well as the constants $f(v)$, and $g_k(v)$. This produces a reliable relaxed linear program ($\mathcal{P}_{XT}$).

3.2 Affine arithmetic based relaxation methods

Affine arithmetic was introduced in 1993 by Comba and Stolfi [9] and developed by De Figueiredo and Stolfi in [13, 15, 40]. In this section, the notations introduced by Comba

and Stolfi in [9] are used[4]. Affine arithmetic can be understood as an extension of interval arithmetic by replacing *intervals* with *affine forms* in order to keep some linear information (about the dependencies between the variables) during the computations of bounds of a function f over a box.

3.2.1 New affine forms, devoted arithmetics and reliable affine arithmetics

Affine arithmetic was first developed for drawing functions on computers and it was extended to be used in global optimization by De Figueiredo and Stolfi [13], Messine [25] and Messine and Touhami [28]. When a non-affine operation is performed, this yields a result which is non-affine and which has to be converted into an affine form. The first idea introduced in [13] was to convert all the non-affine terms into a new affine one which represents the approximation error. In [25, 28], new affine and quadratic forms were proposed. One of the main idea in [25] was to gather all the error terms into only one new extra term representing all the approximation errors. Indeed, an affine form is denoted by:

$$\widehat{x} = x_0 + \left(\sum_{i=1}^{N} x_i \varepsilon_i\right) + x_{\pm}\varepsilon_{\pm}, \tag{2}$$

with $\forall i \in \{0, \ldots, N\}, x_i \in \mathbb{R}$, $x_{\pm} \in \mathbb{R}^+$, $\forall i \in \{1, \ldots, N\}, \varepsilon_i \in \boldsymbol{\epsilon}_i$ and $\varepsilon_{\pm} \in [-1, 1]$.

From two distinct affine forms $\widehat{x} = x_0 + \sum_{i=1}^{N} x_i\varepsilon_i + x_{\pm}\varepsilon_{\pm}$, and $\widehat{y} = y_0 + \sum_{i=1}^{N} y_i\varepsilon_i + y_{\pm}\varepsilon_{\pm}$, of the same size N (by adding some null components if it is necessary), the affine operations between the affine forms are defined as follows:

$$\begin{aligned}
&\widehat{x} \pm \widehat{y} = (x_0 \pm y_0) + \textstyle\sum_{i=1}^{N}(x_i \pm y_i)\varepsilon_i + (x_{\pm} + y_{\pm})\varepsilon_{\pm},\\
&a \pm \widehat{x} = (a \pm x_0) \pm \textstyle\sum_{i=1}^{N} x_i\varepsilon_i + x_{\pm}\varepsilon_{\pm},\\
&a \times \widehat{x} = ax_0 + \textstyle\sum_{i=1}^{N} ax_i\varepsilon_i + ax_{\pm}\varepsilon_{\pm},
\end{aligned}$$

where a is a real number.

The approximation of the multiplication has been improved in [20, 45]. For example, when a multiplication is performed between two affine forms, an efficient approximation which keeps the inclusion properties is the following:

$$\widehat{\widehat{x} \times \widehat{y}} = z_0 + \sum_{i=1}^{N} z_i\varepsilon_i + z_{\pm}\varepsilon_{\pm},$$

with

$$\begin{aligned}
z_0 &= x_0y_0 + \tfrac{1}{2}\sum_{i=1}^{N} x_iy_i,\\
z_i &= x_0y_i + y_0x_i, \ \forall i \in \{1, \ldots, N\},\\
z_{\pm} &= x_{\pm}y_{\pm} + |x_0|y_{\pm} + |y_0|x_{\pm} + y_{\pm}\sum_{i=1}^{N}|x_i| + x_{\pm}\sum_{i=1}^{N}|y_i| +\\
&\quad \frac{1}{2}\sum_{i=1}^{N}|x_iy_i| + \sum_{1\le i,j\le N; i\ne j}|x_iy_j|.
\end{aligned}$$

[4] Note that ε is the chosen notation for the variables in the affine forms and should not be confused with the accuracy parameters ϵ_f and ϵ_{eq} in Section 3.2.

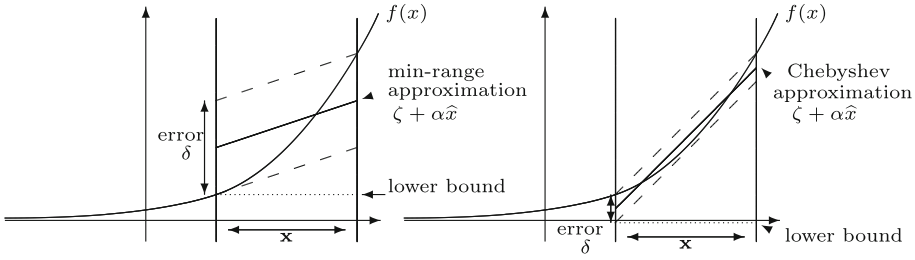

Fig. 2 Affine approximations by min-range methods and Chebyshev

For computing unary functions in affine arithmetic, two linear approximations: the Chebyshev and the min-range approximations are mainly used, see Figure 2 and [14]. These affine approximations have the following form:

$$\widehat{\mathrm{f}}(\widehat{x}) = \zeta + \alpha\widehat{x} + \delta\varepsilon_{\pm},$$
$$\text{with } \widehat{x} \text{ given by equation (2) and } \zeta \in \mathbb{R}, \alpha \in \mathbb{R}, \delta \in \mathbb{R}^{+}.$$

Thus, on the one hand, the Chebyshev linearization produces the affine approximation which minimizes the error δ, but the lower bound is worse than the actual minimum of the range, see Figure 2. On the other hand, the min-range linearization is less efficient for estimating the linear dependency between the symbolic variables ε_i, while the lower bound is equal to the actual minimum of the range. Of course, for a unary function f, the interval where f is considered and the choice of the Chebyshev or the min-range approximation provides the respective values for ζ, α and δ, see [40]. All the usual functions (log, exp, cos, sin, acos, etc.) have been included in the affine arithmetic of the `Ibex` library.

3.2.2 Management of rounding error

Using affine forms and affine arithmetics directly, such as they are defined above, provides non-reliable computations. Indeed, all the coefficients of affine forms are real numbers and have to be converted to floating-point ones. Thus, these approximations and the floating-point operations which will then be used could introduce some numerical difficulties and errors. In order to deal with these numerical errors, three methods were proposed. The first one is due to Stolfi and De Figueiredo in [40]. The idea is to keep floating-point coefficients in the affine forms but particular rounding rules are used in the affine computations in order to render the bounds rigorous. The second reliable method is due to Messine and Touhami [28]. The idea was to consider interval coefficients inside the affine forms and to compute the resulting affine forms using interval arithmetic. The third method is due to Ninin [34], the coefficients are floating-point numbers but an upper bound of all the rounding errors is computed and added to the error terms of the affine forms; this upper bound is computed by using the maximal truncation error number which is 2^{-52} in double precision. Note that these three methods were implemented and compared within `IBBA`, and the third one appears to be clearly the most efficient, see [34]. Thus, the last method is included in the `Ibex` library.

3.2.3 Linear relaxations using affine arithmetics

In [34, 35], affine forms were directly used to generate a linear relaxation of problem $(\mathcal{P})$. Indeed, by converting a box $\mathbf{x}$ into a vector of affine forms $\widehat{x}_i$ depending on ε_i, and by replacing

all the operations in an expression of a function f by affine arithmetic operations, a resulting affine form can be provided; we denote it $\hat{f}(\widehat{x})$. Its corresponding real coefficients are denoted f_i for all $i \in \{1, \ldots, n\}$ and by $\mathsf{f}_\pm$ for the error term corresponding to $\varepsilon_\pm$. By denoting the transformation $\mathcal{T}_{\mathbf{x}}$ which maps $x \in \mathbf{x}$ into $\varepsilon \in [-1, 1]^n$, we have that $x = \mathcal{T}_{\mathbf{x}}(\varepsilon)$ with $\mathbf{x}_i = \text{mid}(\mathbf{x}_i) + \text{rad}(\mathbf{x}_i)\varepsilon_i$; note that $\mathcal{T}_{\mathbf{x}}$ is a bijective mapping if and only if $\text{rad}(\mathbf{x}_i) \neq 0$. Therefore, this yields $\forall x \in \mathbf{x}$, $f(x) = f(\mathcal{T}_{\mathbf{x}}(\varepsilon)) \in [\widehat{f}(\mathcal{T}_{\mathbf{x}}(\varepsilon))]$, where $\widehat{f}(\mathcal{T}_{\mathbf{x}}(\varepsilon)) = \mathsf{f}_0 + \sum_{i=1}^n \mathsf{f}_i\varepsilon_i + \mathsf{f}_\pm\varepsilon_\pm$ denotes the resulting affine form using affine arithmetic operations in the computation of an expression of the function f. Thus, the following inequalities arise:

$$\begin{array}{l} \forall \varepsilon \in \boldsymbol{\epsilon},\ f(\mathcal{T}_{\mathbf{x}}(\varepsilon)) \geq \sum_{i=1}^n \mathsf{f}_i\varepsilon_i + \mathsf{f}_0 - \mathsf{f}_\pm, \\ \forall \varepsilon \in \boldsymbol{\epsilon},\ f(\mathcal{T}_{\mathbf{x}}(\varepsilon)) \leq \sum_{i=1}^n \mathsf{f}_i\varepsilon_i + \mathsf{f}_0 + \mathsf{f}_\pm. \end{array}$$

This result is detailed in Propositions 3.1 and 3.2 of [35].

Therefore, we can derive and obtain the following linear relaxation of problem $(\mathcal{P})$:

$$(\mathcal{P}_{ART}) \left\{ \begin{array}{ll} \min\limits_{\substack{\varepsilon \in [-1,1]^n \\ z \in \mathbb{R}}} & z, \\ \text{s.t.} & \sum_{i=1}^n \mathsf{f}_i\varepsilon_i - z \leq \mathsf{f}_\pm - \mathsf{f}_0, \\ & \sum_{i=1}^n \mathsf{g}_i^k\varepsilon_i \leq \mathsf{g}_\pm^k - \mathsf{g}_0^k, \forall k \in \{1, \ldots, p\}. \end{array} \right.$$

Here g_i^k denotes the i-th component of the corresponding affine form performed using the constraint function g_k over a box $\mathbf{x}$.

Example 2 Let us consider the same function f as for Example 1, over the same interval $\mathbf{x} = [0, 1]$. Now linear relaxations based on affine forms and affine arithmetics are used. By converting $\mathbf{x} = [0, 1]$ into an affine form, one has: $\widehat{x} = \frac{1}{2} + \frac{1}{2}\varepsilon$, with $\varepsilon \in [-1, 1]$. In the computations defined in subsection 3.2.1 with the min-range approximation, we obtain the following affine forms:

$$\begin{array}{rl} \widehat{3\widehat{x}^3} & = \frac{9-2\sqrt{3}}{6} + \frac{3}{2}\varepsilon + \frac{\sqrt{3}}{3}\varepsilon_\pm, \\ \widehat{-2\left(\widehat{x} + \frac{1}{2}\right)^2} & = -\frac{9}{4} - 2\varepsilon + \frac{1}{4}\varepsilon_\pm, \\ 2\widehat{x} + 1 & = 2 + \varepsilon. \end{array}$$

Thus, we obtain the following affine result:

$$\hat{f}(\widehat{x}) = 3\widehat{x}^3 - 2\left(\widehat{x} + \frac{1}{2}\right)^2 + 2\widehat{x} + 1 = \frac{15 - 4\sqrt{3}}{12} + \frac{1}{2}\varepsilon + \frac{3 + 4\sqrt{3}}{12}\varepsilon_\pm$$

Hence, the following affine enclosures are:

$$\forall \varepsilon \in [-1, 1],\ \frac{3 - 2\sqrt{3}}{3} + \frac{1}{2}\varepsilon \leq f(\mathcal{T}_{\mathbf{x}}(\varepsilon)) \leq \frac{3}{2} + \frac{1}{2}\varepsilon.$$

By converting ε into x, with $\varepsilon = \mathcal{T}_{\mathbf{x}}^{-1}(x) = 2x - 1$, we obtain:

$$\forall x \in [0, 1],\ \frac{3 - 4\sqrt{3}}{6} + x \leq f(x) \leq \frac{1}{2} + x.$$

In Figure 3, the linear enclosures are drawn by taking into account the two linear relaxations obtained directly using affine forms and computations.

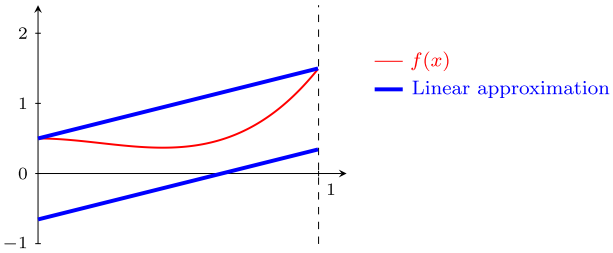

Fig. 3 Affine Arithmetic Relaxation Technique

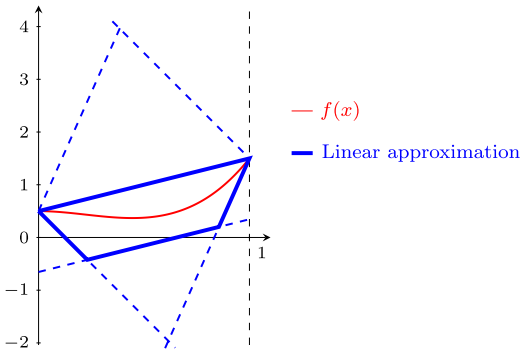

Fig. 4 Hybridation of X-Taylor and ART linear relaxation methods

4 Hybridization of two Linear Relaxations

The X-Taylor linear relaxation method provides bounds by constructing linear underestimations on some *vertices* of a box. Considering linear relaxation technique ART based on affine arithmetic, a linear approximation is provided by taking about the *center* of the box. Thus, the combination of the two distinct approaches appears to be promising. Figure 4 illustrates the complementarity between both approaches on our example (see Section 3). The polyhedral hull of the linear constraints coming from both relaxations is plotted in solid blue line. Computing a lower bound of problem $(\mathcal{P})$ amounts to solving a linear program where the constraints come from the combination of X-Taylor and ART methods.

In order to consider those two types of constraints together, we have first to compute conversions between the X-Taylor and ART linear programs. Indeed, the use of affine arithmetic provides automatically a conversion of the initial domain described by the box $\mathbf{x}$ into a box including the symbolic errors ε where all the variables are in $[-1, 1]$.

Let v denote a vertex of a box $\mathbf{x}$, as detailed in Subsection 3.1 using X-Taylor, considering a general function denoted by $f : \mathbf{x} \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, one has:

$$f(x) \geq a_T(x) = f(v) + (x - v)^\top . \nabla \mathbf{f}^v(\mathbf{x}),$$

where $\nabla \mathbf{f}(\mathbf{x})$ denotes the enclosure of the gradient of f computed for instance via interval automatic differentiation.

Hence, we have:

$$a_T(x) = \underbrace{f(v) - \sum_{i=1}^{n} v_i \nabla \mathbf{f}_i^v(\mathbf{x})}_{\text{constant term}} + \underbrace{\sum_{i=1}^{n} \nabla \mathbf{f}_i^v(\mathbf{x}) x_i}_{\text{linear terms}}.$$

In the same way, as detailed in subsection 3.2 using ART, one obtains:

$$f(\mathcal{T}_{\mathbf{x}}(\varepsilon)) \in \hat{a}_{\mathrm{AF}}(\varepsilon) = \mathsf{f}_0 + \sum_{i=1}^{n} \mathsf{f}_i \varepsilon_i + \mathsf{f}_{\pm}\varepsilon_{\pm},$$

where $\varepsilon_i \in [-1, 1], \forall i \in \{1, \ldots n\}$ and $\varepsilon_{\pm} \in [-1, 1]$ (which is the error term).

In order to combine X-Taylor and ART, we use the variable changes defined above by $x = \mathcal{T}_{\mathbf{x}}(\varepsilon)$. Thus, one has:

$$\hat{a}_T(\varepsilon) = a_T(\mathcal{T}_{\mathbf{x}}(\varepsilon)) = \underbrace{f(v) - \sum_{i=1}^{n} v_i \nabla \mathbf{f}_i^v(\mathbf{x}) + \sum_{i=1}^{n} \nabla \mathbf{f}_i^v(\mathbf{x}) \frac{x_i + \overline{x_i}}{2}}_{\text{constant term}} + \underbrace{\sum_{i=1}^{n} \nabla \mathbf{f}_i^v(\mathbf{x}) \frac{\overline{x_i} - x_i}{2} \varepsilon_i}_{\text{linear terms}}.$$

Conversely, from an affine formulation denoted by $\hat{a}_{\mathrm{AF}}(\varepsilon)$ and by using $\varepsilon_i = \left(x_i - \frac{\overline{x_i} + x_i}{2}\right) \frac{2}{\overline{x_i} - x_i}$, we have the following affine function depending on x and $\varepsilon_{\pm}$:

$$a_{\mathrm{AF}}(x) = \hat{a}_{\mathrm{AF}}(\mathcal{T}_{\mathbf{x}}^{-1}(x)) = \underbrace{\mathsf{f}_0 - \sum_{i=1}^{n} \mathsf{f}_i \frac{x_i + \overline{x_i}}{\overline{x_i} - x_i}}_{\text{constant term}} + \underbrace{\sum_{i=1}^{n} \frac{2\mathsf{f}_i}{\overline{x_i} - x_i} x_i}_{\text{linear terms}} + \underbrace{\mathsf{f}_{\pm}\varepsilon_{\pm}}_{\text{error term}},$$

where $x \in \mathbf{x}$ and $\varepsilon_{\pm} \in [-1, 1]$.

The error term is changed by their corresponding intervals and this provides the two following affine under and over-estimations:

$$\underline{\hat{a}_{\mathrm{AF}}}(\varepsilon) \geq \underline{a_{\mathrm{AF}}}(x) = \underbrace{\mathsf{f}_0 - \mathsf{f}_{\pm} - \sum_{i=1}^{n} \mathsf{f}_i \frac{x_i + \overline{x_i}}{\overline{x_i} - x_i}}_{\text{constant term}} + \underbrace{\sum_{i=1}^{n} \frac{2\mathsf{f}_i}{\overline{x_i} - x_i} x_i}_{\text{linear terms}}.$$

$$\overline{\hat{a}_{\mathrm{AF}}}(\varepsilon) \leq \overline{a_{\mathrm{AF}}}(x) = \underbrace{\mathsf{f}_0 + \mathsf{f}_{\pm} - \sum_{i=1}^{n} \mathsf{f}_i \frac{x_i + \overline{x_i}}{\overline{x_i} - x_i}}_{\text{constant term}} + \underbrace{\sum_{i=1}^{n} \frac{2\mathsf{f}_i}{\overline{x_i} - x_i} x_i}_{\text{linear terms}}.$$

Thus, it is easy to convert an affine form coming from affine arithmetic computations into an affine form with x_i as variables and conversely.

The idea is then to combine both methods in order to provide a relaxed linear program where the linear constraints come from both techniques (see $(\mathcal{P}_{XT})$ and $(\mathcal{P}_{ART})$); these general conversions between affine forms depending on variables x_i and on ε_i make possible

to provide the two following equivalent linear relaxations of problem $(\mathcal{P})$:

$$(\mathcal{P}_{H_x})\begin{cases}\min\limits_{\substack{x\in\mathbf{x}\subseteq\mathbb{R}^n\\ z\in\mathbb{R}}} z,\\ \text{s.t.} \quad f(v)+(x-v)^\top\nabla\mathbf{f}^v(\mathbf{x})\le z, \text{ for any } v\in V_{\mathbf{x}},\\ \quad\ g_k(v)+(x-v)^\top\nabla\mathbf{g}_k^v(\mathbf{x})\le 0, \text{ for any } v\in V_{\mathbf{x}}, \forall k\in\{1,\dots,p\},\\ \quad\ \sum_{i=1}^n \frac{2\mathsf{f}_i}{\overline{x_i}-\underline{x_i}}x_i - z \le \mathsf{f}_\pm - \mathsf{f}_0 + \sum_{i=1}^n \mathsf{f}_i\frac{\overline{x_i}+x_i}{\overline{x_i}-\underline{x_i}},\\ \quad\ \sum_{i=1}^n \frac{2\mathsf{g}_i^k}{\overline{x_i}-\underline{x_i}}x_i \le \mathsf{g}_\pm^k - \mathsf{g}_0^k + \sum_{i=1}^n \mathsf{g}_i^k\frac{\overline{x_i}+x_i}{\overline{x_i}-\underline{x_i}}, \forall k\in\{1,\dots,p\}.\end{cases}$$

and,

$$(\mathcal{P}_{H_\varepsilon})\begin{cases}\min\limits_{\substack{\varepsilon\in[-1,1]^n\\ z\in\mathbb{R}}} z,\\ \text{s.t.} \quad \sum_{i=1}^n \mathsf{f}_i\varepsilon_i - z \le \mathsf{f}_\pm - \mathsf{f}_0,\\ \quad\ \sum_{i=1}^n \mathsf{g}_i^k\varepsilon_i \le \mathsf{g}_\pm^k - \mathsf{g}_0^k, \forall k\in\{1,\dots,p\},\\ \quad\ f(v)+(\mathcal{T}_{\mathbf{x}}(\varepsilon)-v)^\top\nabla\mathbf{f}^v(\mathbf{x})\le z, \text{ for any } v\in V_{\mathbf{x}},\\ \quad\ g_k(v)+(\mathcal{T}_{\mathbf{x}}(\varepsilon)-v)^\top\nabla\mathbf{g}_k^v(\mathbf{x})\le 0,\\ \qquad\qquad\qquad\qquad \text{for any } v\in V_{\mathbf{x}}, \forall k\in\{1,\dots,p\}.\end{cases}$$

Remark 1 In order to solve in a reliable way the relaxed linear programs $(\mathcal{P}_{H_x})$ or $(\mathcal{P}_{H_\varepsilon})$, as explained in the two previous subsections, we have to use the Neumaier and Shcherbina result [31] to provide safe and reliable bounds of those so-obtained linear programs.

Remark 2 Note that a difficulty occurs when a degenerate interval is considered: $\underline{\mathbf{x}}_i = \overline{\mathbf{x}}_i$. In this case, particular attention must be paid in solving the relaxed programs $(\mathcal{P}_{H_x})$ and $(\mathcal{P}_{H_\varepsilon})$. Indeed, in $(\mathcal{P}_{H_x})$ a division by 0 will occur and then, $(\mathcal{P}_{H_x})$ will be not well defined. In order to eliminate this difficulty, we have to remove the variable x_i from $(\mathcal{P}_{H_x})$ by moving it in the second member of the linear inequality constraints. Moreover, a linked numerical difficulty will occur if the width of the interval is too small (because a division by a small floating-point number arises). In this case, we can remove the variable x_i from the linear program and add it to the second member by taking care of its small variation in the small interval. Considering now program $(\mathcal{P}_{H_\varepsilon})$, this difficulty is easiest to discard because the coefficient of ε_i will become 0 and the variable ε_i will directly be removed from the relaxed linear program $(\mathcal{P}_{H_\varepsilon})$.

Obviously, the linear relaxed programs $(\mathcal{P}_{H_x})$ and $(\mathcal{P}_{H_\varepsilon})$ contain all the constraints of $(\mathcal{P}_{XT})$ and $(\mathcal{P}_{H_{ART}})$, and we have the following proposition:

Proposition 1

$$(P)\ge(\mathcal{P}_{H_x})=(\mathcal{P}_{H_\varepsilon})\ge(\mathcal{P}_{XT}) \textit{ and } (P)\ge(\mathcal{P}_{H_x})=(\mathcal{P}_{H_\varepsilon})\ge(\mathcal{P}_{ART}).$$

Remark 3 Note that $(\mathcal{P}_{XT})$ and $(\mathcal{P}_{ART})$ are not comparable.
Moreover, $(\mathcal{P}_{H_x})=(\mathcal{P}_{H_\varepsilon})$ means that the two problems provide the same optimal value for the objective function.

5 Numerical Results

In order to validate our approach, we implemented the `PolytopeHull` contractor (see Section 2) using the different linear relaxation techniques described above. The contractor

was implemented in the Interval-Based EXplorer (`Ibex`) `C++` library[5] [8] and embedded into the default global optimization software called `IbexOpt` [42]. The branching strategy chosen is the `SmearSumRel` variant of the Kearfott Smear function [10] described in [2, 42].

We selected all 112 global optimization problems from the series 1 and 2 of the COCONUT constrained global optimization benchmark[6] solved in less than 1 hour and more than 0.5 seconds by at least one compared strategy, [38].

5.1 Chebyshev versus min-range linearizations

In a preliminary experiment, we have compared the ART Chebyshev and the ART min-range approximations for unary nonlinear functions. This will allow us to select the best linearization technique for the affine arithmetic based relaxation method. Table 1 summarizes the reported results[7].

Those numerical results show the superiority of the Chebyshev linearization over the min-range one. Indeed, 39% of the instances are most quickly solved using the Chebyshev linearization, whereas the min-range is the best strategy for 11% of the instances. (The results are similar in 50% of the instances.) Hence, we decided to use the Chebyshev linearization in the ART affine forms that we develop and use in our code.

This superiority in the numerical tests can be explained because Chebyshev minimizes the maximum absolute error in the linear approximation while min-range minimizes only the range of the unary function. When we linearize a nonlinear region for minimization, the objective is to find hyperplanes which better approximate this region.

5.2 Comparison between different linear relaxation methods

We compared the following implementations of the `PolytopeHull` contractor:

- ART: Using the ART-based relaxation, see Section 3.2.
- XT: Using the X-Taylor relaxation, see Section 3.1.
- Hyb: Using the hybrid approach, which combines the linear constraints obtained with the two above methods, see Section 4.
- ART-XT: Calling twice the `PolytopeHull` contractor, the first one using the ART relaxation and the second one using the X-Taylor relaxation.

Table 1 Comparison of Min-range and Chebyshev linearizations. The Gain value x indicates the number of instances in which the Min-range (resp. Chebyshev) strategy is x times faster than the Chebyshev (resp. Min-range) strategy. The column "$\approx$" indicates the number of instances in which both strategies are equivalent.

Winner	Min-range			$\approx$	Chebyshev		
Gain	> 10	$[2, 10]$	$[\frac{6}{5}, 2]$	≈ 1	$[\frac{6}{5}, 2]$	$[2, 10]$	> 10
#instances	2	1	2	23	7	5	6

[5] https://github.com/ibex-team/ibex-lib.

[6] www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html.

[7] We selected all the 46 instances from series 1 of the COCONUT benchmark requiring less than 1 hour to be solved and with a search-tree size of at least 50 nodes.

Table 2 Comparison of pairs of strategies. The Gain value x indicates the number of instances in which the strategy A (resp. B) was at least x times faster than a strategy B (resp. A). The column "$\approx$" indicates the number of instances in which the strategies A and B are equivalent.

Strategies		Winner A				$\approx$	Winner B			
A	B	> 10	$[5, 10]$	$[2, 5]$	$[\frac{4}{3}, 2]$	≈ 1	$[\frac{4}{3}, 2]$	$[2, 5]$	$[5, 10]$	> 10
ART	XT	9	5	16	22	39	12	5	2	2
Hyb	XT	10	3	12	18	55	11	3	0	0
Hyb	ART	5	2	3	8	69	20	3	2	0
Hyb	ART-XT	1	1	4	27	76	1	1	1	0
Hyb	Best	2	1	0	5	70	27	5	2	0

Table 2 summarizes the results by comparing pairs of strategies A/B named in the first two columns. The line Hyb/Best, corresponds to the comparison between the hybrid approach and the Best among ART and XT.

In the left part (Winner A), an entry in a line A/B and a gain column x (given in second line) indicates the number of instances in which the strategy A runs x times faster than the strategy B. In the right part (Winner B), an entry indicates the number of instances in which the strategy B runs x times faster than the strategy A. The column "$\approx$" indicates the number of instances in which the strategies A and B are equivalent or both spend less than 0.5 seconds.

Note that, in the ART/XT column, ART is generally better than the X-Taylor technique. However, in several particular cases, X-Taylor can be much more efficient; e.g., in 4 instances, it is at least 5 times faster than ART, and in 2 of these 5 instances, X-Taylor is at least 10 times faster.

The hybridization Hyb reduces drastically the number of instances in which X-Taylor performs better while maintaining most of the good results obtained by ART (compare columns ART/XT and Hyb/XT). Compared to ART (see column Hyb/ART), the hybridization Hyb is, in 7 instances, at least 5 times faster than ART and, in 5 of them, at least 10 times faster. On the other hand, ART is at least 5 times faster than Hyb in only 2 instances.

It is interesting to note that, except for 3 instances, the hybridization approach Hyb is strictly better than the ART-XT strategy, which calls twice the PolytopeHull contractor (once with each relaxation method). This is not surprising: assuming that ART-XT and Hyb have a similar time complexity, the Hyb approach filters the domains by using a high-restricted polytope. Indeed, ART-XT uses two polytopes independently, both are relaxations, thus generally offering a poorer contraction.

Summarizing, results show that Hyb outperforms both X-Taylor and ART in many cases, achieving speeds up to ten times faster than either alternative in certain scenarios. Aside from a limited number of exceptions, Hyb surpasses the combined ART-XT strategy, demonstrating superior efficiency. This is largely attributed to its strategic use of a highly restricted polytope, which enables more effective contraction compared to the independent relaxations employed by ART-XT.

Figure 5 shows performance profiles [12]. Each curve represents the percentage of instances solved by a given strategy in less than:
`factor` $\cdot$ (*the CPU time spent by the best strategy*). For instance, when `factor` $= 1$, each curve shows the percentage of instances in which the corresponding strategy provides the best results. Note that ART spent less CPU time in most of the instances (54% of instances of ART versus 29% for Hyb). However, the hybrid approach Hyb outperforms ART when `factor`

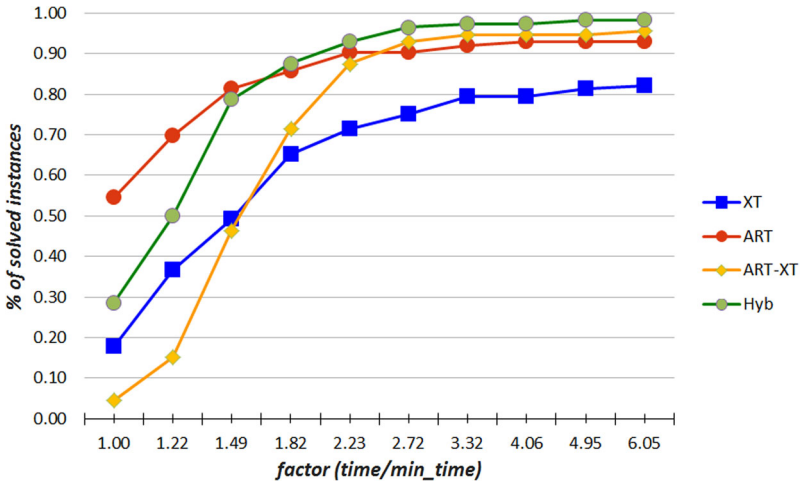

Fig. 5 Performance profile

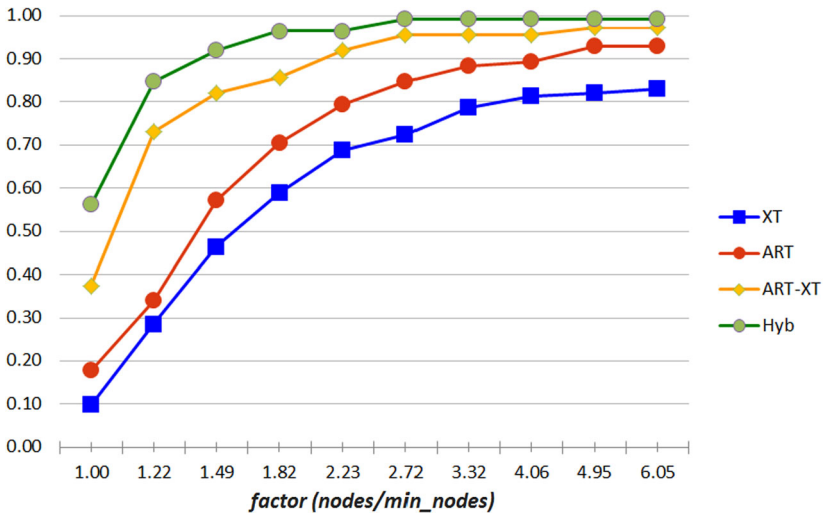

Fig. 6 Tree search size comparison

> 1.8. This behavior can be explained because the ART-based relaxation generates fewer constraints than the hybrid one. Hence, calling `PolytopeHull` using ART is cheaper than calling `PolytopeHull` using Hyb. When the hybrid approach does not improve enough the contraction, the ART-based approach is slightly better because it is cheaper. On the other hand, when the hybrid approach Hyb offers a better contraction than ART, it can greatly improve the performance of the optimization algorithm, because it reduces the tree search size. For the same reason, ART-XT outperforms ART when `factor` > 2.7.

Figure 6 shows a comparison in the size of the tree search, i.e. the number of branching nodes in Algorithm 1. Each curve represents the percentage of instances solved by a given strategy generating less than: `factor` $\cdot$ (*number of nodes of the best strategy*). We can note that the hybrid strategy visits generally fewer nodes than its competitors followed by ART-

XT, ART and X-Taylor. In 56% of the instances, the hybrid approach Hyb is the strategy generating the lowest number of nodes.

In order to evaluate the overhead of the approaches, we compute the following ratio: CPU time divided by the number of treated nodes. The ART-based strategy spent on average 17 milliseconds (ms) per node while the X-Taylor strategy spent 20ms per node. The hybrid approach Hyb is, of course, more expensive and spent 27ms per node on average, while the ART-XT strategy spent 30ms per node.

In summary, the performance profiles show that for over 90% of the cases, the hybrid approach delivers solutions within a CPU time less than double and produces search trees no more than 1.5 times larger than those generated by alternative strategies. This efficiency indicates that, even in the most challenging cases, the performance reduction is capped at a factor of two. Performance comparisons consistently show that ART-XT and X-Taylor lag behind the hybrid approach in performance.

6 Conclusion

In this paper, a new reliable linear relaxation technique is presented by combining two distinct interval-based methods. One of them performs linear relaxations from some vertices of a box using Taylor expansion, and the other one from about the center of the box using the affine arithmetic. Thus, the hybridization of these two reliable linear techniques is possible. The efficiency is validated on 279 test problems coming from the COCONUT library. This shows that the hybrid method solved more problems than the two other techniques, and generally in less CPU time.

Indeed, by hybridization, the generated linear relaxations are more accurate. The advantages of both techniques are merged without drawbacks. The number of nodes visiting by the branch-and-bound algorithm is reduced, and this improvement is not made up by the CPU time spent on each node to solve a larger linear relaxation. In conclusion, this hybrid linear relaxation is now included by default in the last version of `IbexOpt` in the `Ibex` library.

Acknowledgements Ignacio Araya is supported by Fondecyt Project 1200035.

Funding Open access funding provided by Institut National Polytechnique de Toulouse.

References

1. Achterberg, T.: SCIP: Solving Constraint Integer Programs. Math. Program. Comput. **1**(1), 1–41 (2009)
2. Araya, I., Reyes, V., C., O.: More Smear-Based Variable Selection Heuristics for NCSPs. In: Proc. ICTAI, pp. 1004–1011 (2013)
3. Araya, I., Trombettoni, G., Neveu, B.: A contractor based on convex interval taylor. In: Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, pp. 1–16. Springer (2012)

4. Araya, I., Trombettoni, G., Neveu, B., Chabert, G.: Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. J. Glob. Optim. **60**(2), 145–164 (2014)
5. Baharev, A., Achterberg, T., Rév, E.: Computation of an Extractive Distillition Column with Affine Arithmetic. AIChE Journal **55**(7), 1695–1704 (2009)
6. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branch and Bounds Tightening Techniques for Non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)
7. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising Hull and Box Consistency. In: Proceedings of ICLP, pp. 230–244 (1999)
8. Chabert, G., Jaulin, L.: Contractor programming. Artif. Intell. **173**(11), 1079–1100 (2009)
9. Comba, J., Stolfi, J.: 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 (1993)
10. Csendes, T., Ratz, D.: Subdivision direction selection in interval methods for global optimization. Siam J. Numer. Anal. **34**(3), 922–938 (1997)
11. Debruyne, R., Bessière, C.: Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In: Proc. IJCAI, pp. 412–417 (1997)
12. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
13. de Figueiredo, L.: Surface Intersection using Affine Arithmetic. In: Proceedings of Graphics Interface'96, pp. 168–175 (1996)
14. de Figueiredo, L., Stolfi, J.: Adaptive Enumeration of Implicit Surfaces with Affine Arithmetic. Computer Gr. Forum **15**(5), 287–296 (1996)
15. de Figueiredo, L., Stolfi, J.: Affine Arithmetic: Concepts and Applications. Numerical Algorithms **37**(1–4), 147–158 (2004)
16. G., C.L., Martinez, Garcia, E.M.T, H.: Branch-and-Bound interval global optimization on shared memory multiprocessors. Optimization Methods and Software **23**(5), 689–701 (2008)
17. Jansson, C.: Rigorous lower and upper bounds in linear programming. SIAM J. Optim. **14**(3), 914–935 (2004)
18. Kearfott, R., Hongthong, S.: Validated Linear Relaxations and Preprocessing: Some Experiments. Siam J. Optim. **16**(2), 418–433 (2005)
19. Kearfott, R.B.: Rigourous Global Search: Continuous Problems. Kluwer Academic Publishers (1996)
20. Kolev, L.V.: An Improved Interval Linearization for Solving Nonlinear Problems. Numerical Algorithms **37**(1–4), 213–224 (2004)
21. Lebbah, Y., Michel, C., Rueher, M.: A Rigorous Global Filtering Algorithm for Quadratic Constraints. Constraints **10**, 47–65 (2005)
22. Lebbah, Y., Michel, C., Rueher, M.: An Efficient and Safe Framework for Solving Optimization Problems. J. Comput. Appl. Math. **199**, 372–377 (2007)
23. Lebbah, Y., Michel, C., Rueher, M., Daney, D., Merlet, J.: Efficient and Safe Global Constraints for Handling Numerical Constraint Systems. SIAM J. Numerical Anal. **42**(5), 2076–2097 (2005)
24. Messine, F.: Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes. Ph.D. thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse (1997)
25. Messine, F.: Extensions of Affine Arithmetic: Application to Unconstrained Global Optimization. J. Univer. Computer Sci. **8**(11), 992–1015 (2002)
26. Messine, F.: A Deterministic Global Optimization Algorithm for Design Problems. In: C. Audet, P. Hansen, G. Savard (eds.) Essays and Surveys in Global Optimization, pp. 267–294. Springer (2005)
27. Messine, F., Nogarède, B., Lagouanelle, J.L.: Optimal design of electromechanical actuators: A new method based on global optimization. IEEE Trans. Magn. **34**(1), 299–308 (1998)
28. Messine, F., Touhami, A.: A General Reliable Quadratic Form: An Extension of Affine Arithmetic. Reliable Comput. **12**(3), 171–192 (2006)
29. Misener, R., Floudas, C.: ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. J. Global Optim. **59**(2–3), 503–526 (2014)
30. Moore, R.: Interval Anal. Prentice-Hall Inc., Englewood Cliffs (1966)
31. Neumaier, A., Shcherbina, O.: Safe Bounds in Linear and Mixed-Integer Programming. Math. Program. **99**, 283–296 (2004)
32. Neveu, B., Trombettoni, G., Araya, I.: Adaptive Constructive Interval Disjunction: Algorithms and Experiments. Constraints J. **20**(4), 452–467 (2015)
33. Neveu, B., Trombettoni, G., et al.: Adaptive constructive interval disjunction. In: 25th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 900–906 (2013)
34. Ninin, J.: Optimisation Globale basé sur l'Analyse d'Intervalles: Relaxation affine et limitation de la mémoire. Ph.D. thesis, Institut National Polytechnique de Toulouse (2010)

35. Ninin, J., Messine, F., Hansen, P.: A reliable affine relaxation method for global optimization. 4OR **13**(3), 247–277 (2014)
36. Pál, L., Csendes, T.: INTLAB implementation of an interval global optimization algorithm. Optim. Methods Softw. **24**(4–5), 749–759 (2009)
37. Schrage, L.: Optimization Modeling With Lindo. Brooks/Cole (1997)
38. Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.H., Nguyen, T.V.: Benchmarking global optimization and constraint satisfaction codes. In: Bliek, C., Jermann, C., Neumaier, A. (eds.) Global Optim. Constr. Satisfaction, pp. 211–222. Springer, Berlin Heidelberg, Berlin, Heidelberg (2003)
39. Sherali, H., Liberti, L.: Reformulation-Linearization Technique for Global Optimization. In: Encyclopedia of Optimization, vol. 18, pp. 3263–3268. Springer-Verlag (2009)
40. Stolfi, J., de Figueiredo, L.: Self-Validated Numerical Methods and Applications. Monograph for 21st Brazilian Mathematics Colloquium. IMPA/CNPq (1997)
41. Tawarmalani, M., Sahinidis, N.V.: A Polyhedral Branch-and-Cut Approach to Global Optimization. Math. Program. **103**(2), 225–249 (2005)
42. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner Regions and Interval Linearizations for Global Optimization. In: AAAI, pp. 99–104 (2011)
43. Trombettoni, G., Chabert, G.: Constructive interval disjunction. In: Principles and Practice of Constraint Programming (CP), pp. 635–650. Springer (2007)
44. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica : A Modeling Language for Global Optimization. MIT Press (1997)
45. Vu, X.H., Sam-Haroud, D., Faltings, B.: Enhancing numerical constraint propagation using multiple inclusion representations. Annals Math. Artificial Intell. **55**(3), 295–354 (2009)