A Contractor Based on Convex Interval Taylor

Ignacio Araya, Gilles Trombettoni, Bertrand Neveu

UTFSM (Chile), IRIT, INRIA, I3S, Université Nice–Sophia (France), Imagine LIGM
Université Paris–Est (France)
`iaraya@inf.utfsm.cl,Gilles.Trombettoni@inria.fr,Bertrand.Neveu@enpc.fr`

Abstract. Interval Taylor has been proposed in the sixties by the interval analysis community for relaxing continuous non-convex constraint systems. However, it generally produces a non-convex relaxation of the solution set. A simple way to build a convex polyhedral relaxation is to select a *corner* of the studied domain/box as expansion point of the interval Taylor form, instead of the usual midpoint. The idea has been proposed by Neumaier to produce a sharp range of a single function and by Lin and Stadtherr to handle $n \times n$ (square) systems of equations.
This paper presents an interval Newton-like operator, called `X-Newton`, that iteratively calls this interval convexification based on an endpoint interval Taylor. This general-purpose contractor uses no preconditioning and can handle any system of equality and inequality constraints. It uses Hansen's variant to compute the interval Taylor form and uses two opposite corners of the domain for every constraint.
The `X-Newton` operator can be rapidly encoded, and produces good speedups in constrained global optimization and constraint satisfaction. First experiments compare `X-Newton` with affine arithmetic.

1 Motivation

Interval B&B algorithms are used to solve continous constraint systems and to handle constrained global optimization problems in a *reliable* way, i.e., they provide an optimal solution and its cost with a bounded error or a proof of infeasibility. The functions taken into account may be non-convex and can include many (piecewise) differentiable operators like arithmetic operators ($+$, $-$, $.$, $/$), power, log, exp, sinus, etc.

Interval Newton is an operator often used by interval methods to contract/filter the search space [12]. The interval Newton operator uses an *interval Taylor* form to iteratively produce a linear system with interval coefficients. The main issue is that this system is *not* convex. Restricted to a single constraint, it forms a non-convex cone (a "butterfly"), as illustrated in Fig. 1-left. An n-dimensional constraint system is relaxed by an intersection of butterflies that is not convex either. (Examples can be found in [24, 15, 23].) Contracting optimally a box containing this non-convex relaxation has been proven to be NP-hard [16]. This explains why the interval analysis community has worked a lot on this problem for decades [12].

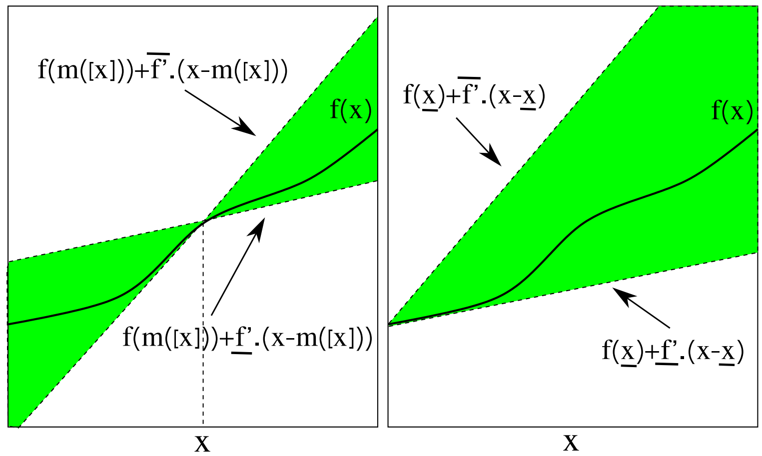

Fig. 1. Relaxation of a function f over the real numbers by a function $g : \mathbb{R} \rightarrow \mathbb{IR}$ using an interval Taylor form (graph in gray). **Left:** Midpoint Taylor form, using a midpoint evaluation $f(m([x]))$, the maximum derivative $\overline{f'}$ of f inside the interval $[x]$ and the minimum derivative $\underline{f'}$. **Right:** Extremal Taylor form, using an endpoint evaluation $f(\underline{x})$, $\overline{f'}$ and $\underline{f'}$.

Only a few polynomial time solvable subclasses have been studied. The most interesting one has been first described by Oettli and Prager in the sixties [27] and occurs when the variables are all non-negative or non-positive. Unfortunately, when the Taylor expansion point is chosen strictly inside the domain (the midpoint typically), the studied box must be previously split into 2^n subproblems/quadrants before falling in this interesting subclass [1, 5, 8]. Hansen and Bliek independently proposed a sophisticated and beautiful algorithm for avoiding explicitly handling the 2^n quadrants [14, 7]. However, the method is restricted to $n \times n$ (square) systems of equations (no inequalities). Also, the method requires the system be first preconditioned (i.e., the interval Jacobian matrix must be multiplied by the inverse matrix of the domain midpoint). The preconditioning has a cubic time complexity, implies an overestimate of the relaxation and requires non-singularity conditions often met only on small domains, at the bottom of the search tree.

In 2004, Lin & Stadtherr [19] proposed to select a *corner* of the studied box, instead of the usual midpoint. Graphically, it produces a convex cone, as shown in Fig. 1-right. The main drawback of this *extremal* interval Taylor form is that it leads to a larger system relaxation surface. The main virtue is that the solution set belongs to a unique quadrant and is convex. It is a polytope that can be (box) hulled in polynomial-time by a linear programming (LP) solver: two calls to an LP solver compute the minimum and maximum values in this polytope for each of the n variables (see Section 4). Upon this extremal interval Taylor, they have built an interval Newton restricted to square $n \times n$ systems of *equations* for which they had proposed in a previous work a specific preconditioning. They have presented a corner selection heuristic optimizing their preconditioning. The selected corner is common to all the constraints.

The idea of selecting a corner as Taylor expansion point is mentioned, in dimension 1, by A. Neumaier (see page 60 and Fig. 2.1 in [24]) for computing a range enclosure (see Def. 1) of a univariate function. Neumaier calls this the *linear boundary value form*. The idea has been exploited by Messine and Laganouelle for lower bounding the objective function in a Branch & Bound algorithm for unconstrained global optimization [21].

McAllester et al. also mention this idea in [20] (end of page 2) for finding cuts of the box in constraint systems. At page 211 of Neumaier's book [24], the step (4) of the presented pseudo-code also uses an endpoint interval Taylor form for contracting a system of equations.[1]

Contributions

We present in this paper a new contractor, called `X-Newton` (for eXtremal interval Newton), that iteratively achieves an interval Taylor form on a corner of the studied domain. `X-Newton` does not require the system be preconditioned and can thus reduce the domains higher in the search tree. It can treat well-constrained systems as well as under-constrained ones (with fewer equations than variables and with inequalities), as encountered in constrained global optimization. The only limit is that the domain must be bounded, although the considered intervals, i.e., the initial search space, can be very large.

This paper experimentally shows that such a contractor is crucial in constrained global optimization and is also useful in continuous constraint satisfaction where it makes the whole solving strategy more robust.

After the background introduced in the next section, we show in Section 3 that the choice of the best expansion corner for any constraint is an NP-hard problem and propose a simple selection policy choosing two opposite corners of the box. Tighter interval partial derivatives are also produced by Hansen's recursive variant of interval Taylor. Section 4 details the extremal interval Newton operator that iteratively computes a convex interval Taylor form. Section 5 highlights the benefits of `X-Newton` in satisfaction and constrained global optimization problems.

This work provides an alternative to the two existing *reliable* (interval) convexification methods used in global optimization. The `Quad` [18, 17] method is an interval *reformulation-linearization technique* that produces a convex polyhedral approximation of the quadratic terms in the constraints. *Affine arithmetic* produces a polytope by replacing in the constraint expressions every basic operator by specific affine forms [10, 32, 4]. It has been recently implemented in an efficient interval B&B [26]. Experiments provide a first comparison between this affine arithmetic and the corner-based interval Taylor.

[1] The aim is not to produce a convex polyhedral relaxation (which is not mentioned), but to use as expansion point the farthest point in the domain from a current point followed by the algorithm. The contraction is not obtained by calls to an LP solver but by the general purpose Gauss-Seidel without taking advantage of the convexity.

2 Background

Intervals allow reliable computations on computers by managing floating-point bounds and outward rounding.

Intervals

An **interval** $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i s.t. $\underline{x_i} \leq x_i \leq \overline{x_i}$, where $\underline{x_i}$ and $\overline{x_i}$ are floating-point numbers. $\mathbb{IR}$ denotes the set of all intervals. The size or **width** of $[x_i]$ is $w([x_i]) = \overline{x_i} - \underline{x_i}$. A **box** $[x]$ is the Cartesian product of intervals $[x_1] \times ... \times [x_i] \times ... \times [x_n]$. Its width is defined by $\max_i w([x_i])$. $m([x])$ denotes the middle of $[x]$. The **hull** of a subset S of $\mathbb{R}^n$ is the smallest n-dimensional box enclosing S.

Interval arithmetic [22] has been defined to extend to $\mathbb{IR}$ elementary functions over $\mathbb{R}$. For instance, the interval sum is defined by $[x_1]+[x_2] = [\underline{x_1}+\underline{x_2}, \overline{x_1}+\overline{x_2}]$. When a function f is a composition of elementary functions, an *extension* of f to intervals must be defined to ensure a conservative image computation.

Definition 1 (Extension of a function to $\mathbb{IR}$; inclusion function; range enclosure)
Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
$[f] : \mathbb{IR}^n \rightarrow \mathbb{IR}$ *is said to be an* **extension** *of* f *to intervals iff:*

$$\begin{array}{ll} \forall [x] \in \mathbb{IR}^n & [f]([x]) \supseteq \{f(x),\ x \in [x]\} \\ \forall x \quad \in \mathbb{R}^n & \quad f(x) = [f](x) \end{array}$$

The **natural extension** $[f]_N$ of a real function f corresponds to the mapping of f to intervals using interval arithmetic. The outer and inner interval linearizations proposed in this paper are related to the first-order **interval Taylor extension** [22], defined as follows:

$$[f]_T([x]) = f(\dot{x}) + \sum_i [a_i] \,.\, ([x_i] - \dot{x}_i)$$

where $\dot{x}$ denotes any point in $[x]$, e.g., $m([x])$, and $[a_i]$ denotes $\left[\frac{\partial f}{\partial x_i}\right]_N([x])$.
Equivalently, we have: $\forall x \in [x],\ \underline{[f]_T([x])} \leq f(x) \leq \overline{[f]_T([x])}$.

Example. Consider $f(x_1, x_2) = 3x_1^2 + x_2^2 + x_1 x_2$ in the box $[x] = [-1, 3] \times [-1, 5]$. The natural evaluation provides: $[f]_N([x_1], [x_2]) = 3[-1, 3]^2 + [-1, 5]^2 + [-1, 3][-1, 5] = [0, 27] + [0, 25] + [-5, 15] = [-5, 67]$. The partial derivatives are: $\frac{\partial f}{\partial x_1}(x_1, x_2) = 6x_1 + x_2$, $[\frac{\partial f}{\partial x_1}]_N([-1, 3], [-1, 5]) = [-7, 23]$, $\frac{\partial f}{\partial x_2}(x_1, x_2) = x_1 + 2x_2$, $[\frac{\partial f}{\partial x_2}]_N([x_1], [x_2]) = [-3, 13]$. The interval Taylor evaluation with $\dot{x} = m([x]) = (1, 2)$ yields: $[f]_T([x_1], [x_2]) = 9 + [-7, 23][-2, 2] + [-3, 13][-3, 3] = [-76, 94]$.

A simple convexification based on interval Taylor

Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ defined on a domain $[x]$, and the inequality constraint $f(x) \leq 0$. For any variable $x_i \in x$, let us denote $[a_i]$ the interval partial derivative $\left[\frac{\partial f}{\partial x_i}\right]_N([x])$. The first idea is to lower tighten $f(x)$ with one of the following interval linear forms that hold for all x in $[x]$.

$$f(\underline{x}) + \underline{a_1} y_1^l + ... + \underline{a_n} y_n^l \ \leq f(x) \tag{1}$$

$$f(\overline{x}) + \overline{a_1} y_1^r + ... + \overline{a_n} y_n^r \ \leq f(x) \tag{2}$$

where: $y_i^l = x_i - \underline{x_i}$ and $y_i^r = x_i - \overline{x_i}$.

A *corner* of the box is chosen: $\underline{x}$ in form (1) or $\overline{x}$ in form (2). When applied to a set of inequality and equality[2] constraints, we obtain a polytope enclosing the solution set.

The correctness of relation (1) – see for instance [30, 19] – lies on the simple fact that any variable y_i^l is non-negative since its domain is $[0, d_i]$, with $d_i = w([y_i^l]) = w([x_i]) = \overline{x_i} - \underline{x_i}$. Therefore, minimizing each term $[a_i]\, y_i^l$ for any point $y_i^l \in [0, d_i]$ is obtained with $\underline{a_i}$. Symmetrically, relation (2) is correct since $y_i^r \in [-d_i, 0] \leq 0$, and the minimal value of a term is obtained with $\overline{a_i}$.

Note that, even though the polytope computation is safe, the floating-point round-off errors made by the LP solver could render the hull of the polytope unsafe. A cheap post-processing proposed in [25], using interval arithmetic, is added to guarantee that no solution is lost by the Simplex algorithm.

3 Extremal interval Taylor form

3.1 Corner selection for a tight convexification

Relations (1) and (2) consider two specific corners of the box $[x]$. We can remark that every other corner of $[x]$ is also suitable. In other terms, for every variable x_i, we can indifferently select one of both bounds of $[x_i]$ and combine them in a combinatorial way: either $\underline{x_i}$ in a term $\underline{a_i}\,(x_i - \underline{x_i})$, like in relation (1), or $\overline{x_i}$ in a term $\overline{a_i}\,(x_i - \overline{x_i})$, like in relation (2).

A natural question then arises: Which corner x^c of $[x]$ among the 2^n-set X^c ones produces the tightest convexification? If we consider an inequality $f(x) \leq 0$, we want to compute a hyperplane $f^l(x)$ that approximates the function, i.e., for all x in $[x]$ we want: $f^l(x) \leq f(x) \leq 0$.

Following the standard policy of linearization methods, for every inequality constraint, we want to select a corner x^c whose corresponding hyperplane is the closest to the non-convex solution set, i.e., adds the smallest volume. This is exactly what represents Expression (3) that maximizes the Taylor form for

[2] An equation $f(x) = 0$ can be viewed as two inequality constraints: $0 \leq f(x) \leq 0$.

all the points $x = \{x_1, ..., x_n\} \in [x]$ and adds their different contributions: one wants to select a corner x^c from the set of corners X^c such that:

$$max_{x^c \in X^c} \int_{x_1=\underline{x_1}}^{\overline{x_1}} ... \int_{x_n=\underline{x_n}}^{\overline{x_n}} (f(x^c) + \sum_i z_i)\, dx_n \dots dx_1 \tag{3}$$

where: $z_i = \overline{a_i}(x_i - \overline{x_i})$ iff $x_i^c = \overline{x_i}$, and $z_i = \underline{a_i}(x_i - \underline{x_i})$ iff $x_i^c = \underline{x_i}$.
Since:

- $f(x^c)$ is independent from the x_i values,
- any point z_i depends on x_i but does not depend on x_j (with $j \neq i$),
- $\int_{x_i=\underline{x_i}}^{\overline{x_i}} \underline{a_i}(x_i - \underline{x_i})dx_i = \underline{a_i} \int_{y_i=0}^{d_i} y_i\, dy_i = \underline{a_i}\, 0.5\, d_i^2$,
- $\int_{x_i=\underline{x_i}}^{\overline{x_i}} \overline{a_i}(x_i - \overline{x_i})dx_i = \overline{a_i} \int_{-d_i}^{0} y_i\, dy_i = -0.5\, \overline{a_i}\, d_i^2$,

Expression (3) is equal to:

$$max_{x^c \in X^c} \prod_i d_i\, f(x^c) + \prod_i d_i \sum_i 0.5\, a_i^c\, d_i$$

where $d_i = w([x_i])$ and $a_i^c = \underline{a_i}$ or $a_i^c = -\overline{a_i}$.
We simplify by the positive factor $\prod_i d_i$ and obtain:

$$max_{x^c \in X^c} \quad f(x^c) + 0.5 \sum_i a_i^c\, d_i \tag{4}$$

Unfortunately, we have proven that this maximization problem (4) is NP-hard.

Proposition 1 *(Corner selection is NP-hard)*
Consider a polynomial[3] $f : \mathbb{R}^n \to \mathbb{R}$, *with rational coefficients, and defined on a domain* $[x] = [0, 1]^n$. *Let* X^c *be the* 2^n*-set of corners, i.e., in which every component is a bound 0 or 1. Then,*

$$max_{x^c \in X^c} \; -(f(x^c) + 0.5 \textstyle\sum_i a_i^c\, d_i)$$
$$\text{(or } min_{x^c \in X^c} \; f(x^c) + 0.5 \textstyle\sum_i a_i^c\, d_i\text{)}$$

is an NP-hard problem.

The extended paper [3] shows straightforward proofs that maximizing the first term of Expression 4 ($f(x^c)$) is NP-hard and maximizing the second term $0.5 \sum_i a_i^c\, d_i$ is easy, by selecting the maximum value among $\underline{a_i}$ and $-\overline{a_i}$ in every term. However, proving Proposition 1 is not trivial (see [3]) and has been achieved with a polynomial reduction from a subclass of `3SAT`, called `BALANCED-3SAT`.[4]

[3] We cannot prove anything on more complicated, e.g., transcendental, functions that make the problem undecidable.

[4] In an instance of `BALANCED-3SAT`, each Boolean variable x_i occurs n_i times in a negative literal and n_i times in a positive literal. We know that `BALANCED-3SAT` is NP-complete thanks to the dichotomy theorem by Thomas J. Schaefer [28].

Even more annoying is that experiments presented in Section 5 suggest that the criterion (4) is not relevant in practice. Indeed, even if the best corner was chosen (by an oracle), the gain in box contraction brought by this strategy w.r.t. a random choice of corner would be not significant. This renders pointless the search for an efficient and fast corner selection heuristic.

This study suggests that this criterion is not relevant and leads to explore another criterion. We should notice that when a hyperplane built by endpoint interval Taylor removes some inconsistent parts from the box, the inconsistent subspace more often includes the selected corner x_c because the approximation at this point is exact. However, the corresponding criterion includes terms mixing variables coming from all the dimensions simultaneously, and makes difficult the design of an efficient corner selection heuristic.

This qualitative analysis nevertheless provides us rationale to adopt the following policy.

Using two opposite corners

To obtain a better contraction, it is also possible to produce *several*, i.e., c, linear expressions lower tightening a given constraint $f(x) \leq 0$. Applied to the whole system with m inequalities, the obtained polytope corresponds to the intersection of these $c\,m$ half-spaces. Experiments (see Section 5.2) suggest that generating two hyperplanes (using two corners) yields a good ratio between contraction (gain) and number of hyperplanes (cost). Also, choosing opposite corners tends to minimize the redundancy between hyperplanes since the hyperplanes remove from the box preferably the search subspaces around the selected corners.

Note that, for managing several corners simultaneously, an expanded form must be adopted to put the whole linear system in the form $Ax-b$ before running the Simplex algorithm. For instance, if we want to lower tighten a function $f(x)$ by expressions (1) and (2) simultaneously, we must rewrite:

1. $f(\underline{x}) + \sum_i \underline{a_i}(x_i - \underline{x_i}) = f(\underline{x}) + \sum_i \underline{a_i} x_i - \underline{a_i}\,\underline{x_i} = \sum_i \underline{a_i} x_i + f(\underline{x}) - \sum_i \underline{a_i}\,\underline{x_i}$
2. $f(\overline{x}) + \sum_i \overline{a_i}(x_i - \overline{x_i}) = f(\overline{x}) + \sum_i \overline{a_i} x_i - \overline{a_i}\overline{x_i} = \sum_i \overline{a_i} x_i + f(\overline{x}) - \sum_i \overline{a_i}\,\overline{x_i}$

Also note that, to remain safe, the computation of constant terms $\underline{a_i}\,\underline{x_i}$ (resp. $\overline{a_i}\,\overline{x_i}$) must be achieved with degenerate intervals: $[\underline{a_i},\underline{a_i}]\,[\underline{x_i},\underline{x_i}]$ (resp. $[\overline{a_i},\overline{a_i}]$ $[\overline{x_i},\overline{x_i}]$).

3.2 Preliminary interval linearization

Recall that the linear forms (1) and (2) proposed by Neumaier and Lin & Stadtherr use the bounds of the interval *gradient*, given by $\forall i \in \{1,...,n\}, [a_i] = \left[\frac{\partial f}{\partial x_i}\right]_N([x])$.

Eldon Hansen proposed in 1968 a variant in which the Taylor form is achieved recursively, one variable after the other [13, 12]. The variant amounts in producing the following tighter interval coefficients:

$$\forall i \in \{1,...,n\}, [a_i] = \left[\frac{\partial f}{\partial x_i}\right]_N ([x_1] \times ... \times [x_i] \times \dot{x_{i+1}} \times ... \times \dot{x_n})$$

where $\dot{x_j} \in [x_j]$, e.g., $\dot{x_j} = m([x_j])$.

By following Hansen's recursive principle, we can produce Hansen's variant of the form (1), for instance, in which the scalar coefficients $\underline{a_i}$ are:

$$\forall i \in \{1, ..., n\},\ \underline{a_i} = \underline{\left[\frac{\partial f}{\partial x_i}\right]_N ([x_1] \times ... \times [x_i] \times \underline{x_{i+1}} \times ... \times \underline{x_n})}.$$

We end up with an `X-Taylor` algorithm (`X-Taylor` stands for *eXtremal interval Taylor*) producing 2 linear expressions lower tightening a given function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ on a given domain $[x]$. The first corner is randomly selected, the second one is opposite to the first one.

4 eXtremal interval Newton

We first describe in Section 4.1 an algorithm for computing the (box) hull of the polytope produced by `X-Taylor`. We then detail in Section 4.2 how this `X-NewIter` procedure is iteratively called in the `X-Newton` algorithm until a quasi-fixpoint is reached in terms of contraction.

4.1 X-Newton iteration

Algorithm 1 describes a well-known algorithm used in several solvers (see for instance [18, 4]). A specificity here is the use of a corner-based interval Taylor form (`X-Taylor`) for computing the polytope.

Algorithm 1 `X-NewIter` (f, x, $[x]$): $[x]$

for j from 1 to m **do**
  polytope ← polytope ∪ {`X-Taylor`(f_j,x,$[x]$)}
end for
for i from 1 to n **do**
  /* Two calls to a Simplex algorithm: */
  $\underline{x_i} \leftarrow \min x_i$ subject to polytope
  $\overline{x_i} \leftarrow \max x_i$ subject to polytope
end for
return $[x]$

All the constraints appear as inequality constraints $f_j(x) \leq 0$ in the vector/set $f = (f_1, ..., f_j, ..., f_m)$. $x = (x_1, ..., x_i, ..., x_n)$ denotes the set of variables with domains $[x]$.

The first loop on the constraints builds the polytope while the second loop on the variables contracts the domains, without loss of solution, by calling a Simplex algorithm twice per variable. When embedded in an interval B&B for constrained global optimization, `X-NewIter` is modified to also compute a lower bound of the objective in the current box: an additional call to the Simplex algorithm minimizes an `X-Taylor` relaxation of the objective on the same polytope.

Heuristics mentioned in [4] indicate in which order the variables can be handled, thus avoiding in practice to call $2n$ times the Simplex algorithm.

4.2 X-Newton

The procedure `X-NewIter` allows one to build the `X-Newton` operator (see Algorithm 2). Consider first the basic variant in which `CP-contractor` $= \perp$.

Algorithm 2 `X-Newton` (f, x, $[x]$, `ratio_fp`, `CP-contractor`): $[x]$

```
repeat
  [x]_save ← [x]
  [x] ← X-NewIter (f, x, [x])
  if CP-contractor ≠ ⊥ and gain([x],[x]_save) > 0 then
    [x] ← CP-contractor(f,x,[x])
  end if
until empty([x]) or gain([x],[x]_save) < ratio_fp)
return [x]
```

`X-NewIter` is iteratively run until a quasi fixed-point is reached in terms of contraction. More precisely, `ratio_fp` is a user-defined percentage of the interval size and:

$$\texttt{gain}([x'], [x]) := \max_i \frac{w([x_i]) - w([x_i'])}{w([x_i])}.$$

We also permit the use of a contraction algorithm, typically issued from constraint programming, inside the main loop. For instance, if the user specifies `CP-contractor=Mohc` and if `X-NewIter` reduces the domain, then the `Mohc` constraint propagation algorithm [2] can further contract the box, before waiting for the next choice point. The guard $\texttt{gain}([x], [x]_{save}) > 0$ guarantees that `CP-contractor` will not be called twice if `X-NewIter` does not contract the box.

Remark

Compared to a standard interval Newton, a drawback of *X-Newton* is the loss of quadratic convergence when the current box belongs to a convergence basin. It is however possible to switch from an endpoint Taylor form to a midpoint one and thus be able to obtain quadratic convergence, as detailed in [3].

Also note that *X-Newton* does not require the system be preconditioned so that this contractor can cut branches early during the tree search (see Section 5.2). In this sense, it is closer to a reliable convexification method like `Quad` [18, 17] or affine arithmetic [26].

5 Experiments

We have applied X-Newton to constrained global optimization and to constraint satisfaction problems.

5.1 Experiments in constrained global optimization

We have selected a sample of global optimization systems among those tested by Ninin et al. [26]. They have proposed an interval Branch and Bound, called here IBBA+, that uses constraint propagation and a sophisticated variant of affine arithmetic. From their benchmark of 74 polynomial and non polynomial systems (without trigonometric operators), we have extracted the 27 ones that required more than 1 second to be solved by the simplest version of IbexOpt (column 4). In the extended paper [3], a table shows the 11 systems solved by this first version in a time comprised between 1 and 11 seconds. Table 1 includes the 13 systems solved in more than 11 seconds.[5] Three systems (ex6_2_5, ex6_2_7 and ex6_2_13) are removed from the benchmark because they are not solved by any solver. The reported results have been obtained on a same computer (Intel X86, 3Ghz).

We have implemented the different algorithms in the Interval-Based EXplorer Ibex [9]. Reference [30] details how our interval B&B, called IbexOpt, handles constrained optimization problems by using recent and new algorithms. Contraction steps are achieved by the Mohc interval constraint propagation algorithm [2] (that also lower bounds the range of the objective function). The upper bounding phase uses original algorithms for extracting *inner regions* inside the feasible search space, i.e., zones in which all points satisfy the inequality and relaxed equality constraints.[6] The cost of any point inside an inner region may improve the upper bound. Also, at each node of the B&B, the X-Taylor algorithm is used to produce hyperplanes for each inequality constraints and the objective function. On the obtained convex polyhedron, two types of tasks can be achieved: either the lower bounding of the cost with one call to a Simplex algorithm (results reported in columns 4 to 13), or the lower bounding and the contraction of the box, with X-NewIter (i.e., $2n+1$ calls to a Simplex algorithm; results reported in column 10) or X-Newton (columns 11, 13). The bisection heuristic is a variant of Kearfott's Smear function described in [30].

The first two columns contain the name of the handled system and its number of variables. Each entry contains generally the CPU time in second (first line of a multi-line) and the number of branching nodes (second line). The same precision on the cost ($1.e{-}8$) and the same timeout (TO = 1 hour) have been used by IbexOpt and IBBA+.[7] Cases of memory overflow (MO) sometimes occur. For each

[5] Note that most of these systems are also difficult for the *non* reliable state-of-the-art global optimizer Baron [29], i.e., they are solved in a time comprised between 1 second and more than 1000 seconds (time out).
[6] An equation $h_j(x) = 0$ is relaxed by two inequality constraints: $-\epsilon \leq h_j(x) \leq +\epsilon$.
[7] The results obtained by IBBA+ on a similar computer are taken from [26].

method m, the last line includes an average gain on the different systems. For a given system, the gain w.r.t. the basic method (column 4) is $\frac{CPU\ time(Rand)}{CPU\ time(m)}$. The last 10 columns of Table 1 compare different variants of `X-Taylor` and `X-Newton`. The differences between variants are clearer on the most difficult instances. All use Hansen's variant to compute the interval gradient (see Section 3.2). The gain is generally small but Hansen's variant is more robust: for instance `ex_7_2_3` cannot be solved with the basic interval gradient calculation.

In the column 3, the convexification operator is removed from our interval B&B, which underlines its significant benefits in practice.

Table 1. Experimental results on difficult constrained global optimization systems

1	2	3	4	5	6	7	8	9	10	11	12	13	14
System	n	No	Rand	R+R	R+op	RRRR	Best	B+op	XIter	XNewt	Ibex'	Ibex"	IBBA+
ex2_1_7	20	TO	42.96 20439	43.17 16492	40.73 15477	49.48 13200	TO	TO	**7.74** 1344	10.58 514	TO	TO	16.75 1574
ex2_1_9	10	MO	40.09 49146	29.27 30323	22.29 23232	24.54 19347	57560	26841	**9.07** 5760	9.53 1910	46.58 119831	103 100987	154.02 60007
ex6_1_1	8	MO	20.44 21804	19.08 17104	**17.23** 14933	22.66 14977	24204	15078	31.24 14852	38.59 13751	TO	633 427468	TO
ex6_1_3	12	TO	1100 522036	711 2.7e+5	529 205940	794 211362	TO	TO	262.5 55280	**219** 33368	TO	TO	TO
ex6_2_6	3	TO	162 172413	175 1.7e+5	169 163076	207 163967	1.7e+5	1.6e+5	172 140130	**136** 61969	1033 1.7e+6	583 770332	1575 9.2e+5
ex6_2_8	3	97.10 1.2e+5	121 117036	119 1.1e+5	110 97626	134.7 98897	1.2e+5	97580	78.1 61047	**59.3** 25168	284 523848	274 403668	458 2.7e+5
ex6_2_9	4	**25.20** 27892	33.0 27892	36.7 27826	35.82 27453	44.68 27457	27881	27457	42.34 27152	43.74 21490	455 840878	513 684302	523 2.0e+5
ex6_2_10	6	TO	3221 1.6e+6	2849 1.2e+6	**1924** 820902	2905 894893	1.1e+6	8.2e+5	2218 818833	2697 656360	TO	TO	TO
ex6_2_11	3	10.57 17852	19.31 24397	**7.51** 8498	7.96 8851	10.82 10049	5606	27016	13.26 12253	11.08 6797	41.21 93427	11.80 21754	140.51 83487
ex6_2_12	4	2120 2e+6	232 198156	160 1.1e+5	118.6 86725	155 90414	1.9e+5	86729	51.31 31646	**22.20** 7954	122 321468	187 316675	112.58 58231
ex7_3_5	13	TO	44.7 45784	54.9 44443	60.3 50544	75.63 43181	45352	42453	29.88 6071	**28.91** 5519	TO	TO	TO
ex14_1_7	10	TO	433 223673	445 1.7e+5	**406** 156834	489 125121	1.7+5	1.1+5	786 179060	938 139111	TO	TO	TO
ex14_2_7	6	93.10 35517	94.16 25802	102.2 21060	83.6 16657	113.7 15412	20273	18126	**66.39** 12555	97.36 9723	TO	TO	TO
Sum			5564 3.1e+6	4752 2.2e+6	**3525** 1.7e+6	5026 1.7e+6			3767 1.4e+6	4311 983634	1982 3.6e+6	1672 2.3e+6	2963 1.6e+6
Gain			1	1.21	1.39	1.07			**2.23**	1.78			
ex7_2_3	8	MO	MO	MO	MO	MO			**544** 611438	691 588791	TO	719 681992	TO

The column 4 corresponds to an `X-Taylor` performed with one corner randomly picked for every constraint. The next column (R+R) corresponds to a tighter polytope computed with two randomly chosen corners per inequality constraint. The gain is small w.r.t. *Rand*. The column 6 (R+op) highlights the best `X-Taylor` variant where a random corner is chosen along with its opposite corner. Working with more than 2 corners appeared to be counter-productive, as shown by the column 7 (*RRRR*) that corresponds to 4 corners randomly picked.

We have performed a very informative experiment whose results are shown in columns 8 (*Best*) and 9 (*B+op*): an exponential algorithm selects the best corner, maximizing the expression (4), among the 2^n ones.[8] The reported number of branching nodes shows that the best corner (resp. *B+op*) sometimes brings no additional contraction and often brings a very small one w.r.t. a random corner (resp. *R+op*). Therefore, the combination *R+op* has been kept in all the remaining variants (columns 10 to 14).

The column 10 (*XIter*) reports the results obtained by `X-NewIter`. It shows the best performance on average while being robust. In particular, it avoids the memory overflow on `ex7_2_3`. `X-Newton`, using `ratio_fp=20%`, is generally slightly worse, although a good result is obtained on `ex6_2_12` (see column 11).

The last three columns report a first comparison between AA (affine arithmetic; Ninin et al.'s AF2 variant) and our convexification methods. Since we did not encode AA in our solver due to the significant development time required, we have transformed `IbexOpt` into two variants `Ibex'` and `Ibex''` very close to `IBBA+`: `Ibex'` and `Ibex''` use a non incremental version of `HC4` [6] that loops only once on the constraints, and a *largest-first* branching strategy. The upper bounding is also the same as `IBBA+` one. Therefore we guess that only the convexification method differs from `IBBA+`: `Ibex'` improves the lower bound using a polytope based on a random corner and its opposite corner; `Ibex''` builds the same polytope but uses `X-Newton` to better contract on all the dimensions.[9]

First, `Ibex'` reaches the timeout once more than `IBBA+`; and `IBBA+` reaches the timeout once more than `Ibex''`. Second, the comparison in the number of branching points (the line *Sum* accounts only the systems that the three strategies solve within the timeout) underlines that AA contracts generally more than `Ibex'`, but the difference is smaller with the more contracting `Ibex''` (that can also solve `ex7_2_3`). This suggests that the job on all the variables compensates the relative lack of contraction of `X-Taylor`. Finally, the performances of `Ibex'` and `Ibex''` are better than `IBBA+` one, but it is probably due to the different implementations.

5.2 Experiments in constraint satisfaction

We have also tested the `X-Newton` contractor in constraint satisfaction, i.e., for solving well constrained systems having a finite number of solutions. These systems are generally square systems (n equations and n variables). The constraints correspond to non linear differentiable functions (some systems are polynomial, others are not). We have selected from the COPRIN benchmark[10] all the systems that can be solved by one of the tested algorithms in a time between 10 s and 1000 s: we discarded easy problems solved in less than 10 seconds, and too difficult problems that no method can solve in less than 1000 seconds. The timeout

[8] We could not thus compute the number of branching nodes of systems with more than 12 variables because they reached the timeout.

[9] We have removed the call to `Mohc` inside the `X-Newton` loop (i.e., `CP-contractor=`$\perp$) because this constraint propagation algorithm is not a convexification method.

[10] `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`

was fixed to one hour. The required precision on the solution is 10^{-8}. Some of these problems are scalable. In this case, we selected the problem with the greatest size (number of variables) that can be solved by one of the tested algorithms in less than 1000 seconds.

We compared our method with the state of art algorithm for solving such problems in their original form (we did not use rewriting of constraints and did not exploit common subexpressions). We used as reference contractor our best contractor `ACID(Mohc)`, an adaptive version of `CID` [31] with `Mohc` [2] as basic contractor, that exploits the monotonicity of constraints. We used the same bisection heuristic as in optimization experiments. Between two choice points in the search tree, we called one of the following contractors (see Table 2).

- `ACID(Mohc)`: see column 3 (Ref),
- `X-NewIter`: `ACID(Mohc)` followed by one call to Algorithm 1 (column 4, Xiter),
- `X-Newton`: the most powerful contractor with `ratio_fp=20%`, and `ACID(Mohc)` as internal CP contractor (see Algorithm 2).

For `X-Newton`, we have tested 5 ways for selecting the corners (see columns 5–9):

- `Rand`: one random corner,
- `R+R`: two random corners,
- `R+op`: one random corner and its opposite,
- `RRRR`: four random corners,
- `2R+op`: four corners, i.e., two random corners and their two respective opposite ones.

We can observe that, as for the optimization problems, the corner selection `R+op` yields the lowest sum of solving times and often good results. The last line of Table 2 highlights that all the 24 systems can be solved in 1000 s by `X-Newton R+op`, while only 18 systems are solved in 1000 s by the reference algorithm with no convexification method. Each entry in Table 2 contains the CPU time in second (first line of a multi-line) and the number of branching nodes (second line). We have reported in the last column (Gain) the gains obtained by the best corner selection strategy `R+op` as the ratio w.r.t. the reference method (column 3 Ref), i.e., $\frac{CPU\,time(R+op)}{CPU\,time(Ref)}$. Note that we used the inverse gain definition compared to the one used in optimization (see 5.1) in order to manage the problems reaching the timeout. We can also observe that our new algorithm `X-Newton R+op` is efficient and robust: we can obtain significant gains (small values in bold) and lose never more than 39% in CPU time.

We have finally tried, for the scalable systems, to solve problems of bigger size. We could solve `Katsura-30` in 4145 s, and `Yamamura1-16` in 2423 s (instead of 33521 s with the reference algorithm). We can remark that, for these problems, the gain grows with the size.

Table 2. Experimental results on difficult constraint satisfaction problems: the best results and the gains (< 1) appear in bold

1	2	3	4	5	6	7	8	9	10
System	n	Ref	Xiter	Rand	R+R	R+op	RRRR	2R+op	Gain
Bellido	9	10.04	3.88	4.55	3.71	**3.33**	**3.35**	**3.28**	**0.33**
		3385	1273	715	491	443	327	299	
Bratu-60	60	494	**146**	306	218	190	172	357	**0.38**
		9579	3725	4263	3705	3385	3131	5247	
Brent-10	10	**25.31**	28	31.84	33.16	34.88	37.72	37.11	1.38
		4797	4077	3807	3699	3507	3543	3381	
Brown-10	10	TO	**0.13**	0.17	0.17	0.17	0.17	0.18	**0**
			67	49	49	49	49	49	
Butcher8-a	8	**233**	246	246	248	242	266	266	1.06
		40945	39259	36515	35829	35487	33867	33525	
Butcher8-b	8	**97.9**	123	113.6	121.8	122	142.4	142.2	1.26
		26693	23533	26203	24947	24447	24059	24745	
Design	9	**21.7**	23.61	22	22.96	22.38	25.33	25.45	1.03
		3301	3121	2793	2549	2485	2357	2365	
Direct Kinematics	11	85.28	**81.25**	84.96	83.52	84.28	86.15	85.62	**0.99**
		1285	1211	1019	929	915	815	823	
Dietmaier	12	3055	1036	**880**	979	960	1233	1205	**0.31**
		493957	152455	113015	96599	93891	85751	83107	
Discrete integral-16	32	TO	480	**469**	**471**	**472**	478	476	**0**
2nd form.			57901	57591	57591	57591	57591	57591	
Eco9	8	**12.85**	14.19	14.35	14.88	15.05	17.48	17.3	1.17
		4573	3595	3491	2747	2643	2265	2159	
Ex14-2-3	6	45.01	3.83	4.39	3.88	**3.58**	3.87	3.68	**0.08**
		3511	291	219	177	181	145	139	
Fredtest	6	74.61	47.73	54.46	47.43	44.26	42.67	**40.76**	**0.59**
		18255	12849	11207	8641	7699	6471	6205	
Fourbar	4	**258**	317	295	319	320	366	367	1.24
		89257	83565	79048	73957	75371	65609	67671	
Geneig	6	57.32	46.1	46.25	41.33	40.38	**38.4**	**38.43**	**0.7**
		3567	3161	2659	2847	2813	2679	2673	
I5	10	**17.21**	20.59	19.7	20.53	20.86	23.23	23.43	1.21
		5087	4931	5135	4885	4931	4843	4861	
Katsura-25	26	TO	711	1900	1258	**700**	1238	1007	**0**
			9661	17113	7857	4931	5013	4393	
Pramanik	3	**14.69**	20.08	19.16	20.31	20.38	24.58	25.15	1.39
		18901	14181	14285	11919	11865	11513	12027	
Synthesis	33	**212**	235	264	316	259	631	329	1.22
		9097	7423	7135	6051	4991	7523	3831	
Trigexp2-17	17	**492**	568	533	570	574	630	637	1.17
		27403	27049	26215	25805	25831	25515	25055	
Trigo1-14	14	2097	1062	1314	1003	910	865	**823**	**0.43**
		8855	5229	4173	2773	2575	1991	1903	
Trigonometric	5	33.75	30.99	**30.13**	**30.11**	30.65	31.13	31.75	**0.91**
		4143	3117	2813	2265	2165	1897	1845	
Virasoro	8	760	715	729	**704**	**709**	713	715	**0.93**
		32787	35443	33119	32065	32441	30717	27783	
Yamamura1-14	14	1542	**407**	628	557	**472**	520	**475**	**0.26**
		118021	33927	24533	23855	14759	13291	11239	
Sum		>42353	6431	8000	7087	**6185**	7588	7131	
		>1.8e6	531044	477115	432232	415396	382862	382916	
Gain		1	0.75	0.77	0.78	0.76	0.9	0.85	
Solved in 1000 s		18	22	22	22	**24**	22	22	

6 Conclusion

Endowing a solver with a reliable convexification algorithm is useful in constraint satisfaction and crucial in constrained global optimization. This paper has presented the probably simplest way to produce a reliable convexification of the solution space and the objective function. `X-Taylor` can be encoded in 100 lines of codes and calls a standard Simplex algorithm. It rapidly computes a polyhedral convex relaxation following Hansen's recursive principle to produce the gradient and using two corners as expansion point of Taylor: a corner randomly selected and the opposite corner.

This convex interval Taylor form can be used to build an eXtremal interval Newton. The `X-NewIter` variant contracting all the variable intervals once provides on average the best performance on constrained global optimization systems. For constraint satisfaction, both algorithms yield comparable results.

Compared to affine arithmetic, preliminary experiments suggest that our convex interval Taylor produces a looser relaxation in less CPU time. However, the additional job achieved by `X-Newton` can compensate this lack of filtering at a low cost, so that one can solve one additional tested system in the end. Therefore, we think that this reliable convexification method has the potential to complement affine arithmetic and `Quad`.

Acknowledgment

We would like to particularly thank G. Chabert for useful discussions about existing interval analysis results.

References

1. O. Aberth. The Solution of Linear Interval Equations by a Linear Programming Method. *Linear Algebra and its Applications*, 259:271–279, 1997.
2. I. Araya, G. Trombettoni, and B. Neveu. Exploiting Monotonicity in Interval Constraint Propagation. In *Proc. AAAI*, pages 9–14, 2010.
3. I. Araya, G. Trombettoni, and B. Neveu. A Contractor Based on Convex Interval Taylor. Technical Report 7887, INRIA, february 2012.
4. A. Baharev, T. Achterberg, and E. Rév. Computation of an Extractive Distillition Column with Affine Arithmetic. *AIChE Journal*, 55(7):1695–1704, 2009.
5. O. Beaumont. *Algorithmique pour les intervalles*. PhD thesis, Université de Rennes, 1997.
6. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.
7. C. Bliek. *Computer Methods for Design Automation*. PhD thesis, MIT, 1992.
8. G. Chabert. *Techniques d'intervalles pour la résolution de systèmes d'intervalles*. PhD thesis, Université de Nice–Sophia, 2007.
9. G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
10. L. de Figueiredo and J. Stolfi. Affine Arithmetic: Concepts and Applications. *Numerical Algorithms*, 37(1–4):147–158, 2004.

11. A. Goldsztejn and L. Granvilliers. A New Framework for Sharp and Efficient Resolution of NCSP with Manifolds of Solutions. *Constraints (Springer)*, 15(2):190–212, 2010.
12. E. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker inc., 1992.
13. E.R. Hansen. On Solving Systems of Equations Using Interval Arithmetic. *Mathematical Comput.*, 22:374–384, 1968.
14. E.R. Hansen. Bounding the Solution of Interval Linear Equations. *SIAM J. Numerical Analysis*, 29(5):1493–1503, 1992.
15. R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, 1996.
16. V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations.* Kluwer, 1997.
17. Y. Lebbah, C. Michel, and M. Rueher. An Efficient and Safe Framework for Solving Optimization Problems. *J. Computing and Applied Mathematics*, 199:372–377, 2007.
18. Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J.P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.
19. Y. Lin and M. Stadtherr. LP Strategy for the Interval-Newton Method in Deterministic Global Optimization. *Industrial & engineering chemistry research*, 43:3741–3749, 2004.
20. D. McAllester, P. Van Hentenryck, and D. Kapur. Three Cuts for Accelerated Interval Propagation. Technical Report AI Memo 1542, Massachusetts Institute of Technology, 1995.
21. F. Messine, , and J.-L. Laganouelle. Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. *Journal of Universal Computer Science*, 4(6):589–603, 1998.
22. R. E. Moore. *Interval Analysis.* Prentice-Hall, 1966.
23. R.E. Moore, R. B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis.* SIAM, 2009.
24. A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge Univ. Press, 1990.
25. A. Neumaier and O. Shcherbina. Safe Bounds in Linear and Mixed-Integer Programming. *Mathematical Programming*, 99:283–296, 2004.
26. J. Ninin, F. Messine, and P. Hansen. A Reliable Affine Relaxation Method for Global Optimization. *Submitted (research report RT-APO-10-05, IRIT, march 2010)*, 2010.
27. W. Oettli. On the Solution Set of a Linear System with Inaccurate Coefficients. *SIAM J. Numerical Analysis*, 2(1):115–118, 1965.
28. T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proc. STOC, ACM symposium on theory of computing*, pages 216–226, 1978.
29. M. Tawarmalani and N. V. Sahinidis. A Polyhedral Branch-and-Cut Approach to Global Optimization. *Mathematical Programming*, 103(2):225–249, 2005.
30. G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *AAAI*, pages 99–104, 2011.
31. G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.
32. X.-H. Vu, D. Sam-Haroud, and B. Faltings. Enhancing Numerical Constraint Propagation using Multiple Inclusion Representations. *Annals of Mathematics and Artificial Intelligence*, 55(3–4):295–354, 2009.