Journal of Global Optimization (2021) 79:413–429
https://doi.org/10.1007/s10898-020-00878-z

AbsTaylor: upper bounding with inner regions in nonlinear continuous global optimization problems

Victor Reyes[1] · **Ignacio Araya**[1]

Received: 15 January 2019 / Accepted: 14 January 2020 / Published online: 25 January 2020

Abstract
In this paper we propose AbsTaylor, a simple and quick method for extracting *inner polytopes*, i.e., entirely feasible convex regions in which all points satisfy the constraints. The method performs an inner linearization of the nonlinear constraints by using a Taylor form. Unlike a previous proposal, the expansion point of the Taylor form is not limited to the bounds of the domains, thus producing, in general, a tighter approximation. For testing the approach, AbsTaylor was introduced as an upper bounding method in a state-of-the-art global branch & bound optimizer. Furthermore, we implemented a local search method which extracts feasible inner polytopes for iteratively finding better solutions inside them. In the studied instances, the new method finds in average four times more inner regions and significantly improves the optimizer performance.

Keywords Upper-bounding · Taylor-based linearization · Nonlinear continuous global optimization

1 Introduction

This paper deals with Nonlinear Continuous global Optimization Problems (NCOPs), which are defined as follows:

$$\min_{x \in \boldsymbol{x}} f(x) \quad s.t. \quad g(x) \leq 0, \tag{1}$$

with $x \in \mathbb{R}^n$ the set of variables varying in the box[1] $\boldsymbol{x}$. $f : \mathbb{R}^n \to \mathbb{R}$ is the real-valued objective function and $g : \mathbb{R}^n \to \mathbb{R}^m$ is a set of inequality constraints. f and g may be nonlinear (convex or non-convex) functions. A **feasible solution** is a vector x satisfying all the constraints of the problem. An **optimal solution** x^* is a feasible solution such that

[1] An interval $\boldsymbol{x}_i = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i, such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. A box $\boldsymbol{x}$ is a Cartesian product of intervals $\boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_i \times \cdots \times \boldsymbol{x}_n$.

✉ Victor Reyes
vareyesrodriguez@lirmm.fr

Ignacio Araya
ignacio.araya@pucv.cl

[1] Pontificia Universidad Católica de Valparaíso, Valparaiso, Chile

$f(x^*) \leq f(x')$ for any feasible solution $x' \in \mathbb{R}$. In this work we focus on global optimization solvers (or global optimizers) able to find *just one* ϵ**-optimal solution**, i.e., a feasible solution x^ϵ such that $f(x^\epsilon) \leq f(x^*) + \epsilon$.

Global optimizers generally follow a branch & bound strategy for solving NCOPs (e.g., IbexOpt [1], Baron [2], Antigone [3], LINDO [4], Couenne [5], Icos [6], GlobSol [7]). Starting with a root node that considers the initial variable domains, optimizers build a search tree by alternating *filtering*, *upper bounding*, and *bisection* steps until an ϵ-optimal solution is found (i.e., they guarantee ϵ-optimality). Filtering consists in removing values from the domains which are not consistent with the constraints (or with some optimality conditions). Bisection consists in expanding the current node by bisecting the domain of one variable and generating two children. Upper bounding consists in finding a feasible solution in the current box that improves the cost of the best solution found so far. Global optimizers, in general, differ mainly in the procedures used for filtering, bisection, upper bounding, and the selection of the next node to process. In particular, interval-based optimizers (e.g., IbexOpt, GlobSol and Icos) are able to solve NCOPs in a *rigorous* way, i.e., they account for the rounding errors that are implied by floating-point operations in their implementations. Interval-based optimizers distinguish from other optimizers in that (1) they account for uncertainties in the parameter values and computation errors over the floating-point numbers and (2) they are able to treat the complete search space, thus offering proofs of infeasibility and/or certification of solutions. Because of these features, these methods can be used in several research areas, including robotics, localization, model qualification, and the design of control systems [8]. From the existent interval-based optimizers, IbexOpt seems to be the most robust [9].

Upper bounding the objective function is a crucial task of global optimizers in general. Finding a feasible solution x^{ub} improving the best solution found so far, helps the optimizer to reduce the search space through the auxiliary constraint: $f(x) < f(x^{ub}) - \epsilon$ [8]. Quick local search methods are generally used to find feasible solutions in each box of the search tree (e.g., selecting the midpoint of the box [10], applying the Newton method [6,11] or extracting convex inner regions [9]), plus some feasibility tests (e.g., the natural evaluation of the candidate point [9,10] or the Borsuk proof [6]). As the upper bounding is performed in all the nodes of the search tree, the methods may sometimes fail to find feasible solutions without compromising the global convergence of the optimizer.

Some optimizers (e.g., Antigone and Baron) use local solvers like CONOPT [12] or IPOpt [13] for upper bounding. Due to local solvers are relatively costly in time, they are generally used only at the beginning of the search or, dynamically, just in some nodes of the search tree [14] (in the rest of nodes quick methods are applied instead). CONOPT implements the Generalized Reduced Gradient (GRG) method [15] which is a generalization of the gradient descent method for problems with nonlinear constraints. GRG first transforms the problem into an unconstrained one, an then the search direction is obtained by combining the gradient of the objective function and a *pseudo-gradient* derived from the equality constraints. The search direction provides that any active constraint remains active for some small step. IPOpt implements a primal-dual interior point method. Interior point methods [16] are iterative methods which approximate the nonlinear problem by using the Lagrangian and a logarithmic barrier function for the slack variables. Then, they apply the Newton's method to the KKT conditions[2] of the approximation for obtaining the search directions of the method (functions should be twice continuously differentiable). Finally, a line search method is used for computing the step size of the iteration. An important decision is related to the procedure for choosing the parameter for the barrier function [18].

[2] Karush–Kuhn–Tucker (KKT) conditions are necessary conditions for a solution to be optimal [17].

Trust region methods for constrained optimization [19–21] are also local search algorithms that could be used for upper bounding. These algorithms generally start from an initial point and try to obtain a new iterate point by just searching in the neighbourhood of the current one. In each iteration k, an approximation of the problem (1) is solved for a sub-region of the search space around the current best solution x_k (trust region). If a new best solution x_{k+1} is found, i.e., $f(x_{k+1}) < f(x_k)$, then the current solution is updated. The approximated problem consists in minimizing an approximation of the objective function (e.g., a linearization or a quadratic approximation) subject to a linearization of the constraints in the trust region. We have to highlight that this kind of algorithms does not warrant the feasibility of the found solutions.

In [9] the authors propose XTaylor, an upper bounding procedure which builds an *inner convex region*, i.e., a subset of the search space in which all solutions are feasible. The procedure uses a first order Taylor form related to a box $\boldsymbol{x}$ and generates, for each inequality $g_j(x) \leq 0$, a linear constraint $h_j(x) \leq 0$, such that $g_j(x) \leq h_j(x)$ for any x in $\boldsymbol{x}$. If it succeeds in building the inner region, a linearization of the objective function is minimized using a simplex algorithm and a feasible solution, approximating the optimal one in the box, is found. The expansion point of the linearization is restricted to be a corner of the box $\boldsymbol{x}$, this means that even knowing a feasible solution x' in the box, it is not certain that the method is capable of building an inner region around this solution.

In this work we make a twofold contribution. First, we propose AbsTaylor, a quick upper bounding algorithm which also uses Taylor-based linearizations for constructing inner regions related to the problem (1). Thanks to the application of a linearization with *absolute values*, it overcomes the difficulty presented by XTaylor, i.e., it can use any point inside the box as the expansion point. Through experiments we show that, when the midpoint of the box is selected as the expansion point, AbsTaylor finds, in average, four times more inner regions than its predecessor. Also we show that, when IbexOpt uses an upper bounding technique based on AbsTaylor, it reports a significant reduction in the size of the search tree, specially when the precision of the optimizer is set to not-so-low values (e.g., $\epsilon = 10^{-2}$ or $\epsilon = 10^{-4}$).

Second, we propose IterativeTaylor, a simple local search algorithm which explores inner regions by iteratively applying AbsTaylor. The procedure is similar to the followed by trust region algorithms: it starts from an initial feasible solution x_c and constructs an inner region around x_c by using AbsTaylor. If a better solution can be found inside this region, then this solution becomes the expansion point of the next iteration. The process is repeated until a terminal criteria is reached. Integrated into a branch & bound optimizer, IterativeTaylor improves even more the performance of the optimizer. Unlike existent upper bounding methods:

- IterativeTaylor is quite simple: it performs first-order relaxations of the problem and uses the simplex algorithm for finding promising solutions inside the convex regions. It does not require to compute second derivatives and does not use the KKT conditions as in the case of primal-dual interior point methods.
- IterativeTaylor finds a feasible solution in each iteration of the search because it only works with completely feasible regions.[3] Other methods, like trust region algorithms, the GRG method or interior point methods, do not ensure that the current candidate is a feasible solution, but only an approximated one.

[3] Providing that an initial feasible solution is given. Otherwise the same method may find an initial feasible solution.

– AbsTaylor is able to construct inner regions around any feasible solution, making it a perfect component for iterative local search algorithms (such as IterativeTaylor). XTaylor, another approach, guarantees success in finding inner regions *only* if we match a corner of the box (the expansion point) with a feasible solution.

The paper is organized as follows. Section 2 provides basic notions related to interval arithmetic. In Sect. 3 we describe our Taylor-based linearization which is compared to the previous proposal. Section 4 describes our local search algorithm and reports the experimental results. Section 5 presents our conclusions and the future work.

2 Background

In this section we introduce some basics concepts related to interval arithmetic and extraction of inner regions.

2.1 Intervals

An **interval** $\boldsymbol{x}_i = [\underline{x}_i, \overline{x}_i]$ 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. The size or **width** of $\boldsymbol{x}_i$ is defined as $\text{wid}(\boldsymbol{x}_i) = \overline{x}_i - \underline{x}_i$. $\text{mid}(\boldsymbol{x}_i)$ denotes the midpoint of $\boldsymbol{x}_i$, where $\text{mid}(\boldsymbol{x}_i) = \frac{\overline{x}_i + \underline{x}_i}{2}$. The **hull** of two intervals is defined as $\text{hull}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left[\min(\underline{x}_i, \underline{x}_j), \max(\overline{x}_i, \overline{x}_j)\right]$. A **box** $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ represents the Cartesian product of intervals $\boldsymbol{x}_1 \times \ldots \times \boldsymbol{x}_n$. The **size** of a box is $\text{wid}(\boldsymbol{x}) = \max_{\boldsymbol{x}_i \in \boldsymbol{x}} \text{wid}(\boldsymbol{x}_i)$. Interval arithmetic defines the extension of unary and binary operators, for instance:

$$
\begin{aligned}
\boldsymbol{x}_1 + \boldsymbol{x}_2 &= [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2] \\
\boldsymbol{x}_1 - \boldsymbol{x}_2 &= [\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2] \\
\boldsymbol{x}_1 * \boldsymbol{x}_2 &= [\min(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2), \max(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2)] \\
\log(\boldsymbol{x}_1) &= [\log(\underline{x}_1), \log(\overline{x}_1)]
\end{aligned}
$$

A function $f : \mathbb{IR}^n \to \mathbb{IR}$ is **factorable** if it can be computed in a finite number of simple steps, using unary and binary operators. $\boldsymbol{f} : \mathbb{IR}^n \to \mathbb{IR}$ is said to be an **extension** of a real factorable function f to intervals if:

$$\forall \boldsymbol{x} \in \mathbb{IR}^n, \quad f(\boldsymbol{x}) \supseteq \{f(x), x \in \boldsymbol{x}\}.$$

The optimal image $\boldsymbol{f}_{opt}$ is the sharpest interval containing the image of $f(x)$ over $\boldsymbol{x}$. There are several kinds of extensions, in particular, the **natural extension** $\boldsymbol{f}_{nat}$ corresponds to mapping a real n-dimensional function f to intervals by using interval arithmetic.

2.2 Extracting convex feasible regions

The XTaylor method [9] extracts inner regions by using a first order interval Taylor form of the function vector g. The interval Taylor form guarantees that for all $x \in \boldsymbol{x}$ and $j \in \{1, ..., m\}$:

$$g_j(x) \in g_j(x') + \sum_{i}^{n} \boldsymbol{J}_{ij} \cdot (x_i - x'_i), \tag{2}$$

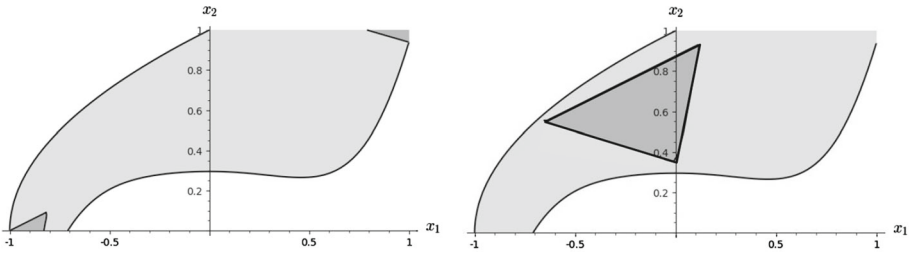

Fig. 1 Inner linearizations by using the interval Taylor form. (left) Inner polytopes generated by XTAYLOR (dark gray) inside the feasible region (light gray); (right) inner polytope generated by ABSTAYLOR using the midpoint (dark gray) inside the feasible region (light gray)

where $\boldsymbol{J}$ is the interval Jacobian matrix of g in the box $\boldsymbol{x}$, i.e., any element $\boldsymbol{J}_{ij}$ in the matrix corresponds to an interval overestimation of the image of the partial derivative $\frac{\partial g_j}{\partial x_i}(x)$ over $\boldsymbol{x}$. $x' \in \boldsymbol{x}$ corresponds to the expansion point.

Due to $\boldsymbol{J}_{ij}$ being an interval, the approximation is not convex unless the expansion point is a corner of the box $\boldsymbol{x}$. The XTAYLOR method proposes to use a corner of the box $x^c \in \boldsymbol{x}$ as the expansion point and the linear approximation becomes:

$$g_j(x) \leq h_j(x) = g_j(x^c) + \sum_i^n J'_{ij} \cdot (x_i - x_i^c), \tag{3}$$

where, $J'_{ij} = \overline{J_{ij}}$ if $x_i^c = \underline{x}$, otherwise $J'_{ij} = \underline{J_{ij}}$.

The set of solutions satisfying $h(x) \leq 0$ denotes a convex inner region or *inner polytope* which is subset of the set of solutions satisfying $g(x) \leq 0$.

Example 1 Let consider the following system of constraints:

$$g_1(x) = x_1^5 + 0.5\cos(x_1) + \sin(x_2) - 2x_2 - 0.2 \leq 0$$
$$g_2(x) = -x_1 + x_2^2 - 1 \leq 0$$

in the box $\boldsymbol{x} = [-1, 1] \times [0, 1]$. Applying XTAYLOR with each of the four corners of the box as the expansion point, we only obtain two non-empty inner regions. By using $x^c = (\underline{x_1}, \underline{x_2})$ we obtain:

$$h_1(x) = 5.4207x_1 - x_2 + 4.4909 \leq 0$$
$$h_2(x) = -x_1 + 2x_2 - 1 \leq 0$$

And by using $x^c = (\overline{x_1}, \overline{x_2})$ we obtain:

$$h_1(x) = -0.4207x_1 - 1.4596x_2 + 1.792 \leq 0$$
$$h_2(x) = -x_1 \leq 0$$

Figure 1-left shows in light gray the feasible region related to the original constraint $g(x) < 0$ and in dark gray the convex inner regions generated by XTAYLOR.

In order to find a promising solution inside the polytope, the authors propose to use the simplex algorithm for minimizing a linearization of the objective function, i.e., $\min_{x \in \boldsymbol{x}} f(x') + \sum_i^n \text{mid}(\boldsymbol{J}_{ij}) \cdot (x_i - x'_i)$, subject to the constraint $h(x) \leq 0$.

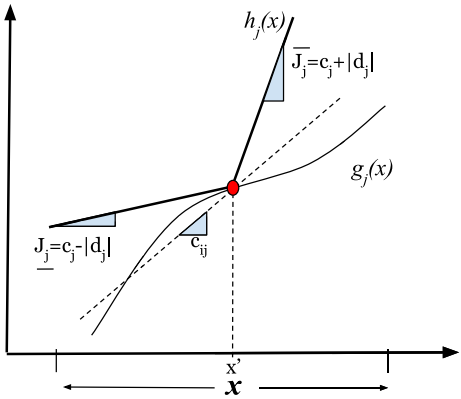

Fig. 2 Example of a Taylor-based linearization with absolute values for an univariate function $g_j(x)$

Note that the XTaylor method requires that the expansion point be in a corner of the box. In this work we propose AbsTaylor, a new method for extracting inner regions which can use any point inside $\boldsymbol{x}$ as the expansion point.

3 AbsTaylor: an inner linearization based on absolute values

In this section we propose a simple method for extracting inner regions which may use any point in the box (e.g., the midpoint) as the expansion point. To do that, we first decompose the interval partial derivatives into two values: $\boldsymbol{J}_{ij} = c_{ij} + \boldsymbol{d}_{ij}$, where the real value c_{ij} is the midpoint of the interval derivative and $\boldsymbol{d}_{ij} = \boldsymbol{J}_{ij} - c_{ij}$ (note that $\overline{d_{ij}} = -\underline{d_{ij}}$).

Then, considering the relation (2), we propose the following Taylor-based inner linearization with absolute values:

$$g_j(x) \leq h_j(x) = g_j(x') + \sum_{i}^{n} \left(c_{ij} \cdot (x_i - x_i') + |d_{ij}| \cdot |x_i - x_i'| \right), \tag{4}$$

where $x' \in \mathbb{R}^n$ can be any point inside the box $\boldsymbol{x}$, for instance $x' = \text{mid}(\boldsymbol{x})$, and $|d_{ij}| = \overline{d_{ij}} = -\underline{d_{ij}}$. Figure 2 illustrates an example for an univariate function. Triangles represent the slope of the corresponding line segments. The function $h_j(x)$, consisting on the two thick segments above $g_j(x)$, starts with a slope equal to $\underline{J_j}$ which becomes $\overline{J_j}$ after the expansion point. The dotted line with slope equal to c_j represents the function $h_j(x)$ without the term $|d_j| \cdot |x - x'|$.

Example 2 Let consider the same system and box of the Example 1. By applying (4), and using the midpoint of the box as the expansion point, we obtain the following linearization:

$$g_1 \leq 2.5x_1 + 2.9207|x_1| - 1.2298x_2 + 0.2298|x_2 - 0.5| + 0.4944 \leq 0$$
$$g_2 \leq -x_1 + x_2 + |x_2 - 0.5| - 1.25 \leq 0$$

The polytope obtained through this process can be seen in Fig. 1-right. Note that in this case the obtained polytope is much larger than the ones generated by XTaylor.

Similar to that proposed in [9], for finding a promising feasible solution in the inner region, we propose to minimize a linearization of the objective function subject to the constraint system generated by (4), i.e.,

$$\begin{aligned} &\min_{x\in\boldsymbol{x}} \; f(x') + \sum_{i}^{n} \text{mid}(\boldsymbol{J}_{ij}) \cdot (x_i - x'_i) \\ &\text{s.t} \quad g_j(x') + \sum_{i}^{n} \left(c_{ij} \cdot (x_i - x'_i) + |d_{ij}| \cdot |x_i - x'_i|\right) \leq 0. \end{aligned} \tag{5}$$

In order to deal with the absolute value functions, we replace the expressions $|x_i - x'_i|$ by auxiliary variables u_i. Then, as the partial derivative of the constraint functions is positive w.r.t. the auxiliary variables (i.e., $|d_{ij}| \geq 0$), we add the constraints $u_i \geq x_i - x'_i$ and $u_i \geq -(x_i - x'_i)$ and we solve the following equivalent linear program by using the simplex algorithm:

$$\begin{aligned} &\min_{x\in\boldsymbol{x}} \; f(x') + \sum_{i}^{n} \text{mid}(\boldsymbol{J}_{ij}) \cdot (x_i - x'_i) \\ &\text{s.t} \quad g_j(x') + \sum_{i}^{n} \left(c_{ij} \cdot (x_i - x'_i) + |d_{ij}| \cdot u_i\right) \leq 0, \quad \forall j = 1 \ldots m \\ &\qquad u_i \geq x_i - x'_i, \quad \forall i = 1 \ldots n \\ &\qquad u_i \geq -(x_i - x'_i), \quad \forall i = 1 \ldots n. \end{aligned} \tag{6}$$

Experiments

The proposals were implemented as upper bounding procedures in the global optimizer IbexOpt [1]. In IbexOpt, L-Smear [22] is used for selecting the next variable to bisect. A series of state-of-the-art methods are used for filtering: ACID(HC4) [23,24] plus a linear relaxation based contractor combining two relaxation methods (AF2 [10] and XNewton [25]). By default, IbexOpt uses a XTaylor-based upper bounding method which first extracts an inner region (using a random corner of the box as the expansion point) and then, for finding promising solutions inside this region, minimizes a linearization of the objective function by using the **S**implex algorithm (XTaylor&S). The optimizer also uses an additional upper bounding method based on the extraction of feasible boxes [9]. If the previous upper bounding methods fail then the optimizer simply tries with the midpoint of the box.

Our AbsTaylor&S strategy consists in replacing XTaylor by AbsTaylor in the upper bounding procedure. As the inner regions found by AbsTaylor are not necessarily better, in solution quality, than the regions found by XTaylor, we implemented TwiceTaylor&S, a strategy which applies XTaylor&S and then AbsTaylor&S. Remark that for these experiments, AbsTaylor always uses the midpoint of the box as the expansion point.

The experiments were run on a computer with an Intel Core i7-4700MQ CPU 2.40GHz and 8GB RAM, running on Ubuntu 16.04. The set of instances were selected from the COCONUT benchmarks for global optimization.[4] We selected all the instances solved by the default strategy in a time comprised between 2 and 3600 seconds when $\epsilon = 10^{-6}$ (44 instances). Details of these instances are shown in Table 1.

Definition 1 **Gain in relative time** We define as **relative time** $t_r(a, b, \pi)$ the ratio between the mean time ($\bar{t}$) taken by a strategy a and the sum of mean times taken by the strategies a and b in solving an instance π, i.e., $t_r(a, b, \pi) = \frac{\bar{t}(a,\pi)}{\bar{t}(a,\pi)+\bar{t}(b,\pi)}$. Thus, the **gain in relative time** of a strategy a w.r.t. a strategy b is given by $\frac{\sum_{\pi\in\Pi} t_r(b,a,\pi)}{\sum_{\pi\in\Pi} t_r(a,b,\pi)}$, where Π is the set of the considered instances.

[4] http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html.

Table 1 Details of the benchmark instances used in the experiments

Benchmark	n	m	Convex	Benchmark	n	m	Convex
ex2_1_9	11	2	?	ex14_2_3	7	10	?
ex5_4_3	17	14	?	ex14_2_7	7	10	?
ex6_1_1	9	7	?	hhfair	30	26	?
ex6_1_3	13	10	?	himmel16	19	22	?
ex6_2_6	4	2	?	hydro	32	25	?
ex6_2_8	4	2	?	immun	22	8	yes
ex6_2_9	5	3	?	launch	39	29	?
ex6_2_10	7	4	?	linear	25	21	?
ex6_2_11	4	2	?	meanvar	9	3	yes
ex6_2_12	5	3	?	ramsey	34	23	?
ex7_2_3	9	7	?	srcpm	40	28	yes
ex7_2_4	9	5	?	batch	47	74	yes
ex7_2_8	8	4	?	dipigri	7	4	?
ex7_2_9	10	7	?	disc2	28	23	?
ex7_3_4	13	18	?	dixchlng	10	5	?
ex7_3_5	14	16	?	himmelbk	24	14	?
ex8_4_4	18	13	?	hs088	33	32	?
ex8_4_5	16	12	?	hs093	7	3	?
ex8_5_1	7	5	?	hs113	11	4	yes
ex8_5_2	7	5	?	hs119	17	9	?
ex8_5_6	7	5	?	odfits	10	6	yes
ex14_1_7	11	18	?	pentagon	6	12	?

The convexity information was extracted from the supplementary material of [3] (instances without convexity information are denoted with (?))

Table 2 reports a comparison of the different strategies. Results for three different values of ϵ are reported. Each cell shows the spent CPU time in seconds (top) and the number of generated nodes (bottom) by a strategy on the given instance for the corresponding value of ϵ.

The last rows of each column correspond to the gain (in relative time and nodes) of each strategy w.r.t. XTaylor&S considering the whole set of instances. Due to some stochastic mechanisms used by IbexOpt,[5] each strategy was run 5 times on each instance and the average results are reported in the table. Only instances with large time differences between the strategies are reported (i.e., instances such that the quotient between the largest and the smallest time for some precision is larger than 2). We highlight in bold the best CPU time and minimum number of nodes reached for some strategy for each instance and precision. Results for pairs instance-precision where all the strategies spent less than 0.5 s are denoted with a dash (–).

Compared to XTaylor&S, AbsTaylor&S reports better results when ϵ is large. For instance, when $\epsilon = 10^{-2}$, the gain in relative time (resp. in nodes) is 1.30 (resp. 1.48). Significant reductions in both, time and nodes, are reported in several instances (e.g., ex8_4_4,

[5] E.g., similarly to XTaylor, XNewton generates linear relaxations of the constraints by choosing *random* corners of the box as expansion points.

ramsey, srcpm and hs119). However, when $\epsilon = 10^{-6}$ the method behaves similar to XTAYLOR&S as we get almost the same results in terms of CPU time with a marginal improvement in the size of the search tree.

On the other hand, despite the additional cost in CPU time of performing more linearizations and simplex runs, TWICETAYLOR&S outperforms XTAYLOR&S and ABSTAYLOR&S for large values of ϵ. When $\epsilon = 10^{-6}$, TWICETAYLOR&S reports an important reduction in the number of nodes (gain of 1.13 w.r.t. XTAYLOR&S), however this gain is mitigated by the relatively expensive cost of the procedure.

Effectiveness of ABSTAYLOR

Additionally, we evaluated the effectiveness of ABSTAYLOR and XTAYLOR by counting the number of times each method contributes to find new upper bounds inside the TWICETAYLOR approach. We observed that, in average, ABSTAYLOR contributes to find new upper bounds 63% of the times and XTAYLOR only a 24%. We also counted the number of times each strategy was successful in finding a feasible region in a node of the search tree. ABSTAYLOR was successful in 20% of the cases while XTAYLOR was successful only in 5% of them (we did not consider 17 instances where both strategies were always successful).

4 ITERATIVETAYLOR: an iterative local search method based on inner regions

In this section we propose ITERATIVETAYLOR a simple iterative method for finding promising feasible solutions in a box $\boldsymbol{x}$. Starting from an initial point x_p (e.g., the midpoint of the box $\boldsymbol{x}$), ITERATIVETAYLOR first attempts to find a candidate feasible solution x_c in $\boldsymbol{x}$ by using ABSTAYLOR&S with x_c as the expansion point of the linearization (see Algorithm 1).

If it is successful, then the algorithm performs a loop while the candidate solution is improved. In each iteration, the size of $\boldsymbol{x}$ is reduced and a new feasible solution x_c is searched in the new box by applying ABSTAYLOR and the simplex method. Note that the previous found solution x_p is used as the expansion point in the current iteration. The algorithm continues until the cost of the new solution x_c does not improve the previous one more than ϵ.

The function reduce_box($\boldsymbol{x}, x_p, \alpha$) moves the box $\boldsymbol{x}$ by centering it in the point x_p and reduces its size: the size of each interval is set to a fraction of the previous size, i.e., $\text{wid}(\boldsymbol{x}_i) \leftarrow \alpha \cdot \text{wid}(\boldsymbol{x}_i)$ for all $i = 1 \ldots n$, with $\alpha \in [0, 1[$ an user-defined parameter. The proportional reduction of the box size forces the worst-case complexity in the number of steps of the whole procedure to be logarithmic.

Note that XTAYLOR cannot be used for finding inner regions in the while-loop because x_p is not a corner of the box but the midpoint.

Experiments

As a second series of experiments, we implemented ITERATIVETAYLOR as an upper bounding procedure in the branch & bound optimizer IbexOpt. We added an additional condition in line 3 for entering the loop: the first candidate solution x_c have to improve the best solution found by the optimizer so far. The experiments setup is the same as the first series of experiments.

Table 2 CPU times and number of nodes when the best solution found so far reaches a precision of 10^{-2}, 10^{-4} or 10^{-6} for each strategy

Benchmarks		XTaylor&S			AbsTaylor&S			TwiceTaylor&S		
		$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
ex6_1_1	time	1.44	16.0	**28.4**	0.74	11.8	29.2	**0.72**	**9.65**	31.4
	#nodes	342	4891	8681	154	3011	7924	**151**	**2958**	**7855**
ex6_2_6		–	**2.43**	**13.0**	–	3.24	67.3	–	3.45	71.0
			1857	**11058**		2375	57288		2506	57199
ex7_2_4		1.07	9.82	18.2	**0.44**	**5.7**	**15.5**	0.51	6.66	16.9
		471	3492	7237	**173**	2312	6106	237	**2193**	**6073**
ex7_2_8		0.97	6.90	15.4	0.25	**2.80**	**13.1**	**0.16**	3.53	14.8
		457	2666	6626	93.6	**979.2**	5661	**78.4**	1187	**5454**
ex7_2_9		9.25	18.9	28.5	**3.86**	**11.2**	**19.6**	5.8	13.9	23.6
		8124	11171	13683	**2090**	**4623**	**6662**	4146	6458	8722
ex8_4_4		20.0	216	409	9.18	**151**	**323**	**7.90**	156	351
		1352	14393	28213	509	9976	23881	**446**	**9630**	**22934**
ex8_4_5		0.69	6.62	1197	1.38	4.55	1111	**0.61**	**2.54**	**668**
		83.0	986	275096	127	588	273744	**78.0**	**309**	**116161**

Table 2 continued

Benchmarks	XTaylor&S			AbsTaylor&S			TwiceTaylor&S		
	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
hydro	6.79	**20.6**	58.1	**1.74**	30.1	**54.8**	3.56	28.5	56.3
	220	**674**	1582	**57.2**	876	1639	106	866	**1552**
immun	**37.4**	**39.7**	**40.1**	138	211	244	43.9	48.7	50.3
	8663	**8788**	**8867**	33310	65582	127169	9906	10026	10043
meanvar	0.92	2.71	2.95	0.61	**1.95**	**2.36**	**0.21**	2.66	3.08
	445	1145	1271	218	960	1092	**85.6**	**953**	**1044**
ramsey	2.84	7.62	17.8	**0.10**	**4.06**	16.4	0.11	4.24	**16.0**
	79.2	192	458	**4.00**	**112**	385	**4.00**	115	**344**
srcpm	9.89	52.0	75.4	5.40	37.5	55.1	**3.17**	**23.9**	**47.6**
	477	2314	3813	181	1579	2623	**111**	**882**	**1778**
dipigri	1.31	7.06	13.3	0.60	5.40	9.16	**0.32**	**5.15**	**8.98**
	866	4100	7012	302	**2611**	4817	**166**	2682	**4807**
hs119	16.8	37.1	41.6	**0.53**	**27.7**	**32.5**	0.92	29.5	37.1
	1174	2246	2825	**40.8**	**1912**	2493	68.8	1944	**2479**
Gain (time)				1.30	1.14	1.05	1.42	1.20	1.02
Gain (nodes)				1.48	1.15	1.03	1.47	1.24	1.13

```
1 procedure IterativeTaylor (f, g, x, x_p, ϵ, α); out: x_c
2     x_c ← AbsTaylor&S(f, g, x, x_p);
3     while x_c exists and f(x_c) < f(x_p) − ϵ do
4         x_p ← x_c;
5         x ← reduce_box(x, x_p, α);
6         x_c ← AbsTaylor&S(f, g, x, x_p);
7     return x_c
```

Algorithm 1: The IterativeTaylor algorithm for finding promising solutions

IterativeTaylor versus TwiceTaylor&S

Table 3 reports the results obtained by TwiceTaylor&S and IterativeTaylor using the best two values of α obtained through an unreported series of experiments.[6] The last rows report the gain in relative time and nodes of each strategy and precision w.r.t. TwiceTaylor&S. We have highlighted in bold the best results. Results where all the strategies spent less than 0.5 s are denoted with a dash (–).

First note that in both cases, i.e., when $\alpha = 0.95$ or $\alpha = 0.5$, we obtain important gains in time and nodes. The best results are obtained when $\alpha = 0.5$. This is probably due to Algorithm 1 converges quickly for small values of α. When $\epsilon = 10^{-6}$, IterativeTaylor($\alpha = 0.5$) outperforms TwiceTaylor&S with gains in time and nodes of 1.33 and 1.10 respectively.

Using TwiceTaylor&S for finding the initial candidate

As a last series of experiments, we replaced the call to AbsTaylor&S by a call to TwiceTaylor&S in line 2 of Algorithm 1. In this way we have more chances of finding an initial feasible candidate solution x_c. The new variant (IterativeTaylor2) reports a gain in relative time w.r.t. IterativeTaylor of 1.20, 1.09 and 1.16 for the precision values 10^{-2}, 10^{-4} and 10^{-6} respectively. In number of boxes, the gains were 1.22, 1.11 and 1.08 respectively.

Figure 3 shows a performance profile [26] which compares the strategies with $\epsilon = 10^{-6}$. Each curve reports the proportion of instances solved by the corresponding strategy in less than or equal to *factor* times the best reported CPU time. Note that all the proposals outperform XTaylor&S, the current default strategy used by IbexOpt. IterativeTaylor2 reports the best results in the comparison.

Finally, Fig. 4 reports an informative comparison among IbexOpt, using IterativeTaylor2, and other well-known global optimizers. The results reported by other solvers were obtained from the supplementary material of [3].[7] Each curve reports the proportion of instances solved by the corresponding strategy in less than a certain amount of time measured in seconds. The sample of benchmarks instances used in this comparison correspond to all the not-reported-convex NCOP instances of the COCONUT webpage (130 instances).

From the figure we can see that IbexOpt, using IterativeTaylor2, is rather competitive with the best global optimizers (Baron and Antigone) in problems spending more than 300 seconds of CPU time. Note that, with the exception of IbexOpt, all the compared solvers are

[6] The other tested values of α were 0.1, 0.2, 0.7, 0.8, 0.9 and 0.99.

[7] They were run on a computer with an Intel Xeon X5650 2.67 GHz processor with 24 GB RAM on a 64 bit Linux version.

Table 3 Spent CPU time and number of generated boxes reported by TwiceTaylor&S and IterativeTaylor for different values of ϵ

Benchmarks		TwiceTaylor&S			IterativeTaylor $\alpha = 0.95$			IterativeTaylor $\alpha = 0.5$		
		$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
ex6_2_6	time	–	3.45	71.09	–	0.78	14.8	–	**0.76**	**11.4**
	#nodes		2506	57198		**544**	11091		**544**	**11090**
ex6_2_9		–	1.91	40.5	–	3.71	38.2	–	**1.17**	**35.8**
			1158	28782		2506	**26961**		**796**	27546
ex6_2_11		–	6.43	28.2	–	2.12	24.3	–	**0.07**	**20.8**
			4807	24522		1914	24442		**56.4**	**24045**
ex7_2_3		**2.74**	6.66	8.83	3.36	**3.36**	**7.92**	4.27	7.06	8.95
		2689	4818	5676	3117	**3173**	**5693**	3933	5738	6915
ex7_2_4		0.51	6.66	17.0	**0.09**	**1.96**	13.5	0.14	3.17	**11.7**
		237	2193	6073	**34.8**	**652**	4814	56.8	1144	**4809**
ex7_2_8		–	3.53	14.8	–	1.48	12.0	–	**1.39**	**10.3**
			1187	5454		566	4728		**545**	**4284**
ex7_2_9		5.83	13.9	23.6	5.87	10.1	18.1	**1.91**	**7.23**	**15.2**
		4146	6458	8722	4070	5476	7887	**1209**	**2947**	**5093**
ex8_4_4		7.90	156	351	5.07	124	327	**3.89**	**106**	**284**
		446	9630	22933	355	**7782**	**21301**	**256**	7920	21645
immun		**43.9**	**48.7**	**50.4**	120	164	176	49.0	128	134
		9906	**10026**	**10042**	26622	40149	41990	11516	30765	35905
linear		13.5	20.5	23.5	12.9	16.0	19.5	**7.51**	**10.1**	**13.6**
		3134	3349	3422	3037	3197	3266	**2740**	**3081**	**3121**
srcpm		**3.17**	**23.9**	47.6	18.8	42.5	63.9	6.66	27.7	**44.6**
		111	**882**	**1778**	1083	2324	3046	344	1144	1935
batch		1159	1704	1783	**0.76**	1037	1121	7.01	**837**	**853**
		22395	30824	32284	**14.0**	19539	20650	147	**15752**	**17313**

Table 3 continued

Benchmarks	TwiceTaylor&S			IterativeTaylor $\alpha = 0.95$			IterativeTaylor $\alpha = 0.5$		
	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$
dipigri	–	5.15	8.98	–	**2.04**	6.23	–	2.25	**5.27**
		2682	4807		**1073**	3182		1183	**2767**
hs113	15.0	21.3	26.1	8.17	10.4	18.3	**0.13**	**3.18**	**10.9**
	4157	5483	6335	2471	3391	4730	**72.4**	**860**	**3060**
hs119	0.92	29.5	37.1	**0.58**	24.3	34.2	0.86	**4.48**	**23.5**
	68.8	1944	2479	**40.0**	1705	2213	65.6	**316**	**1903**
odfits	2.34	33.5	91.5	1.59	32.5	93.9	**0.75**	**19.1**	**59.4**
	1204	13943	39418	866	15537	39987	**426**	**7324**	**26371**
Gain (times)				1.19	1.22	1.13	1.45	1.48	1.33
Gain (nodes)				1.13	1.12	1.04	1.38	1.41	1.10

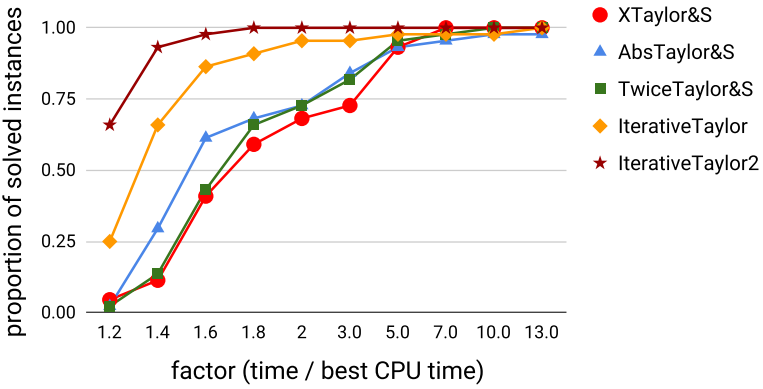

Fig. 3 Performance profile. Comparison between the results reported by XTAYLOR&S and the proposed strategies based on ABSTAYLOR with $\epsilon = 10^{-6}$. For a given strategy, a point (f, p) on the corresponding curve indicates that a proportion p of the instances was solved in less than f times the best reported CPU time

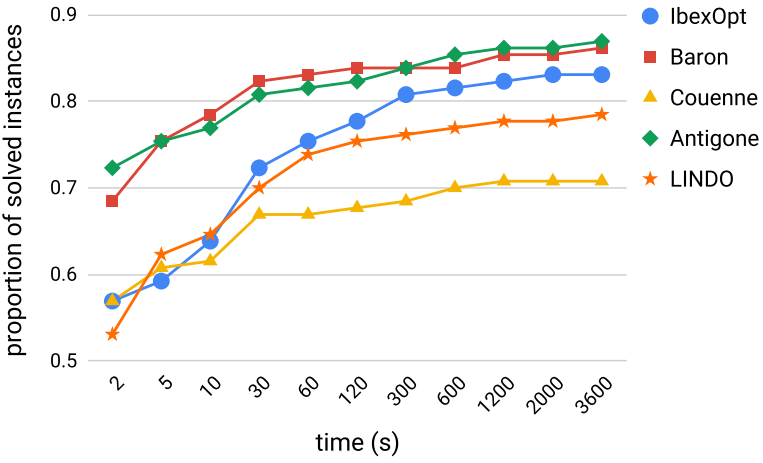

Fig. 4 Performance profile. For a given strategy, a point (t, p) on the corresponding curve indicates that a proportion p of the 130 instances was solved in less than t seconds

not rigorous, i.e., they cannot guarantee their results. Other rigorous solvers, such as GlobSol and Icos, are not competitive at all, therefore, they were not included in the comparison.

5 Conclusions

In this work we present ABSTAYLOR a new inner Taylor-based linearization technique that can be used for finding feasible solutions of NCOPs. Additionally we have presented ITERATIVETAYLOR, a simple iterative local search algorithm which uses inner regions and simplex for finding promising solutions.

Experiments show promising results. On one hand, the combination of our approach ABSTAYLOR and the state-of-the-art XTAYLOR seems to significantly improve the performance of interval branch & bound optimizers, especially when a low precision is required

for the results. Furthermore, IterativeTaylor and IterativeTaylor2 reduce even more the CPU time spent by the optimizer for any required precision.

As a future work we plan to improve the local search algorithm by incorporating a global optimization procedure which searches the optimal solution in the *feasible line segment* joining the current candidate solution with the next one. We also plan to compare IterativeTaylor with other basic iterative approaches such as line search and trust region methods [19,20].

Acknowledgements This work is supported by the Fondecyt Project 1160224. Victor Reyes is supported by the Grant Postgrado PUCV 2018.

References

1. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner regions and interval linearizations for global optimization. In: AAAI Conference on Artificial Intelligence, (2011)
2. Sahinidis, N.V.: Baron: a general purpose global optimization software package. J. Glob. Optim. **8**(2), 201–205 (1996)
3. Misener, R., Floudas, C.A.: Antigone: algorithms for continuous/integer global optimization of nonlinear equations. J. Glob. Optim. **59**, 1–24 (2013)
4. Schrage, L.: Linear, Integer and Quadratic Programming with LINDO. The Scientific Press, Singapore (1986)
5. Belotti, P.: Couenne: a user's manual. Technical report, Lehigh University, Tech. Rep. (2009)
6. Lebbah, Y.: ICOS: a branch and bound based solver for rigorous global optimization. Optim. Methods Softw. **24**(4–5), 709–726 (2009)
7. Kearfott, R.B.: Rigorous Global Search Continuous Problems. Springer, Berlin (1996)
8. Araya, I., Reyes, V.: Interval branch-and-bound algorithms for optimization and constraint satisfaction: a survey and prospects. J. Glob. Optim. **65**(4), 837–866 (2016)
9. 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)
10. Ninin, J., Messine, F., Hansen, P.: A reliable affine relaxation method for global optimization. 4OR **13**(3), 247–277 (2015)
11. Goldsztejn, A., Lebbah, Y., Michel, C., Rueher, M.: Revisiting the upper bounding process in a safe branch and bound algorithm. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming, pp. 598–602. Springer, Berlin (2008)
12. Drud, A.S.: CONOPT—a large-scale GRG code. ORSA J. Comput. **6**(2), 207–216 (1994)
13. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
14. Khajavirad, A., Sahinidis, N.V.: A hybrid LP/NLP paradigm for global optimization relaxations. Math. Program. Comput. **10**(3), 383–421 (2018)
15. Lasdon, L.S., Waren, A.D., Jain, A., Ratner, M.: Design and testing of a generalized reduced gradient code for nonlinear programming. Stanford University CA Systems Optimization Lab, Technical Report (1976)
16. Wright, S.J.: Primal-Dual Interior-Point Methods, vol. 54. SIAM, Philadelphia (1997)
17. Kuhn, H., Tucker, A.: Nonlinear programming. In: Second Berkeley Symposium on Mathematical Statistics and Probability, pp. 481–492 (1951)
18. Nocedal, J., Wächter, A., Waltz, R.A.: Adaptive barrier update strategies for nonlinear interior methods. SIAM J. Optim. **19**(4), 1674–1693 (2009)
19. Yuan, Y.-X.: Recent advances in trust region algorithms. Math. Program. **151**(1), 249–281 (2015)
20. Coleman, T.F., Li, Y.: An interior trust region approach for nonlinear minimization subject to bounds. SIAM J. Optim. **6**(2), 418–445 (1996)
21. Omojokun, E.O.: Trust region algorithms for optimization with nonlinear equality and inequality constraints. Ph.D Dissertation, University of Colorado (1989)
22. Araya, I., Neveu, B.: Ismear: a variable selection strategy for interval branch and bound solvers. J. Glob. Optim. **71**, 1–18 (2017)
23. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising hull and box consistency. In: International Conference on Logic Programming. Citeseer (1999)

24. Trombettoni, G., Chabert, G.: Constructive interval disjunction. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming-CP 2007, pp. 635–650. Springer, Berlin (2007)
25. 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)
26. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)