A standard branch-and-bound approach for nonlinear semi-infinite problems

Antoine Marendet, Alexandre Goldsztejn, Gilles Chabert, Christophe Jermann

A standard branch-and-bound approach for nonlinear semi-infinite problems

Antoine Marendet[a], Alexandre Goldsztejn[b,*], Gilles Chabert[c], Christophe Jermann[d]

[a] *Laboratoire des Sciences du Numérique de Nantes, École Centrales de Nantes, 1 rue de la Noë, Nantes, France*
[b] *Centre National de la Recherche Scientifique, Laboratoire des Sciences du Numérique de Nantes, École Centrales de Nantes, 1 rue de la Noë, Nantes, France*
[c] *IRT Jules Verne, Chemin du Chaffault, 44340 Bouguenais, France*
[d] *Laboratoire des Sciences du Numérique de Nantes, Université de Nantes, 2 Chemin de la Houssinière, Nantes, France*

Abstract

This paper considers nonlinear semi-infinite problems, which contain at least one semi-infinite constraint (SIC). The standard branch-and-bound algorithm is adapted to such problems by extending usual upper and lower bounding techniques for nonlinear inequality constraints to SICs. This is achieved by defining the interval evaluation of parametrized functions and their generalized gradients, by also adapting numerical constraint programming techniques to quantified inequalities, and by introducing linear relaxations and restrictions for SICs. The overall efficiency of our algorithm is demonstrated on a standard set of benchmarks from the literature, in comparison with the best state of the art alternative.

Keywords: Semi-infinite programming, branch-and-bound, relaxation, restriction, constraint programming

1. Introduction

In this paper, we address semi-infinite programs (SIP), which are nonlinear programs (NLP)

$$\min_{\substack{g(x)\leq 0\\ h(x)=0}} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$ and $h_i : \mathbb{R}^n \to \mathbb{R}$ with at least one semi-infinite constraint (SIC) $g_i(x) \leq 0$, i.e.,

$$g_i(x) = \max_{y\in Y} \tilde{g}_i(x, y), \tag{2}$$

*Corresponding author
Email address: alexandre.goldsztejn@ls2n.fr (Alexandre Goldsztejn)

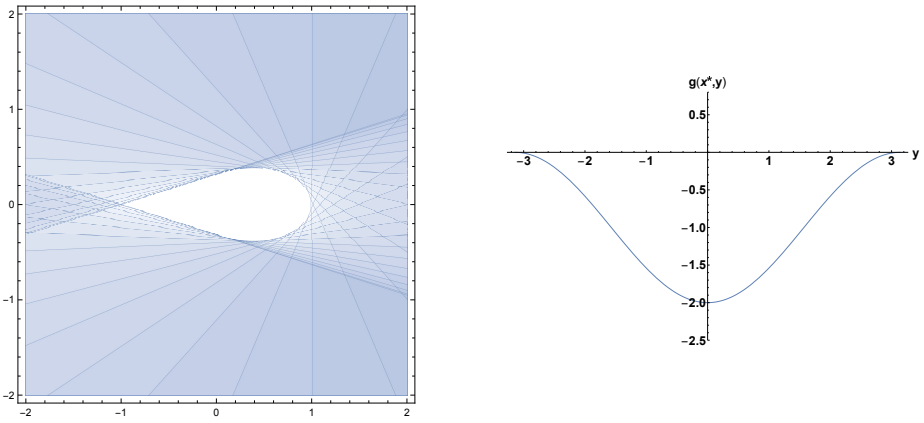

Figure 1: On the left-hand side, the feasible set of the SIC from Example 1 ; its infeasible set, in blue, is approximated by a discretization of $y \in [-\pi, \pi]$, hence the feasible set is the white area. On the right-hand side, $y \mapsto \tilde{g}(x^*, y)$ for $x^* = (-1, 0)^T$.

where $\tilde{g}_i : \mathbb{R}^n \times \mathbb{R}^{m_i} \to \mathbb{R}$ is supposed to be continuously differentiable. We call $x \in \mathbb{R}^n$ the decision variables, and $y \in \mathbb{R}^{m_i}$ the parameters of the SIC $g_i(x) \leq 0$. The maximization problem (2) is called the lower-level program. Enforcing the SIC $g_i(x) \leq 0$ is equivalent to enforcing $\forall y \in Y, \tilde{g}_i(x, y) \leq 0$, hence actually enforcing an inequality constraint for each parameter value inside Y. We restrict our attention to box-constrained lower-level programs: We consider the set Y to be a box $[y] = \{y \in \mathbb{R}^{m_i} : \underline{y} \leq y \leq \overline{y}\}$ for given $\underline{y}, \overline{y} \in \mathbb{R}^{m_i}$ such that $\underline{y} \leq \overline{y}$ (the inequalities are defined component-wise). It contains infinitely many parameter values when the box is not degenerated to a singleton, hence the name SIC. The following example shows a typical SIC.

Example 1. *The SIC* $g(x) := \max_{y \in [y]} \tilde{g}(x, y) \leq 0$, *with* $[y] := [-\pi, \pi]$ *and*

$$\tilde{g}(x, y) = x_1 \cos(y) + x_2(\sin(y) + y) - 1 \tag{3}$$

is equivalent to $\forall y \in [y], \tilde{g}(x, y) \leq 0$. *For a fixed value of* y, *it is linear with respect to* x. *Therefore, it is actually the conjunction of infinitely many linear constraints. Its feasible set is depicted in Figure 1, where a finite subset of the linear constraints are depicted. While* $\tilde{g}(x, y)$ *is smooth, the feasible set of* $g(x) \leq 0$ *presents a "peak" at* $x^* = (-1, 0)^T$. *In fact,* $g(x)$ *is not differentiable at* x^*. *Figure 1 also displays* $\tilde{g}(x^*, y)$ *as a function of* $y \in [-\pi, \pi]$ *and shows that* $\max_{y \in [y]} \tilde{g}(x^*, y) = 0$, *i.e.,* $g(x^*) = 0$, *is attained at two parameter maximizers* $y = -\pi$ *and* $y = \pi$. *Such a non-smoothness due to multiple global maximizers of the lower-level program is a typical feature of SICs, which makes them hard to solve. In particular, the generalized gradient, in the sense of Clarke [17], of* $g(x)$ *has to be considered instead of its usual gradient.*

SIPs have been the topic of intensive studies during the past decades due to their numerous applications, e.g., inventory and logistics, finance, revenue management, queuing networks, machine learning, energy systems and the public

good [24], but also control of robots, eigenvalue computations, mechanical stress of materials, and statistical design [34]. Most of the research in this area has focused on SIPs that are convex with respect to decision variables and have specific dependance with respect to parameters, e.g., convex or linear dependance [6] or concave dependance [35, 5, 55]. In general, proving the feasibility of a SIC for a given point x amounts to solving a global optimization problem. In this situation, dynamically populated discretization of the parameters domains have been proposed, see, e.g., [12, 34], and have given rise to so-called discretization methods [56, 43, 55]. See [34, 55] for reviews on these topics.

On the other hand, the branch-and-bound approach to solving (non-robust) NLPs [51, 22] has been developed to the point where it has become efficient enough to address the global optimization of important applications, e.g., in robotics, control and engineering [36, 10, 13, 28]. Although branch-and-bound algorithms are usually strongly sensitive to the number of variables, they turn out to be useful for very nonlinear small scale problems.This approach has also been successfully extended to larger classes of problems, e.g., multi-objective nonlinear problems [21, 44].

Several branch-and-bound approaches have been proposed to solve SIPs. Upper and lower bounding procedures are proposed in [23], which represent a preliminary step toward a full branch-and-bound algorithm for SIP. A discretization is used for the lower bounding and a convexification of the constraint with respect to parameters is performed for the upper bounding. Another adaptation of the branch-and-bound algorithm is proposed in [11], where a discretization is used for the lower bounding, and a semi formal interval evaluation is used for the upper bounding: The function $\tilde{g}(x, y)$ is evaluated for the interval $[y]$ and kept formal for the decision variable x, leading to a non-smooth restriction of the SIC. In both cases, the upper bounding process requires subdividing the parameters domains. A branch-and-bound algorithm dedicated to quantified quadratic problems has also been proposed in [20], where the problem is non-convex with respect to variables but linear with respect to parameters.

A different approach has been proposed by Mitsos [46] based on Blankenship's method [12]. This algorithm relies on successive global solving of several auxiliary problems, which can be performed by a branch-and-bound algorithm or other global solving methods for particular classes of problems, e.g., linear, convex, polynomial, etc., leading to converging lower and upper bounds. At each iteration k, a finite set of parameter samples $Y_k \subseteq Y$ is used to build relaxations $\tilde{g}(x, y) \leq 0$ for $y \in Y_k$, which are then solved globally using a branch-and-bound algorithm leading to successive minimizers x_k^* of the relaxed problem. Between each iteration, one new parameter sample is added to the set of parameter samples using one maximizer among $\operatorname{argmax}_{y \in Y} \tilde{g}(x_k^*, y)$. This addition tightens the relaxation and forces x_k^* to converge to a minimizer of the SIP as k increases. Meanwhile, upper bounds are computed by solving globally an auxiliary problem where the relaxations $\tilde{g}(x, y) \leq 0$, $y \in Y_k$, are changed into $\tilde{g}(x, y) \leq -\varepsilon$, $y \in Y_k$ for some positive scalar ε. The latter auxiliary problem is neither a relaxation nor a restriction, but allows finding converging upper bounds. It has been recently improved [19] using in addition the "oracle" technique previously

proposed in [58]. A complete comparison of [46, 19] with [23, 11] is difficult, but some clues for a very favorable comparison were provided in [46], supported by some experiments on a small set of benchmarks and different machines. Nevertheless, [19] is currently the best implementation for solving globally non-convex SIPs. However, it presents the drawback of solving globally several instances of the discretized SIP, which can turn out to be computationally very expensive for hard problems.

In this paper, we revisit the adaptation of branch-and-bound algorithms to SIPs by integrating SICs in the process just as regular non-linear constraints. This is done by extending bounding methods used in the branch-and-bound algorithm to SICs: In particular, we 1) show how to perform interval evaluations of SICs and of their generalized gradients, 2) show how to apply numerical constraint programming techniques to SICs, 3) derive rigorous linear relaxations and linear restrictions of SICs, and 4) extend the convergent upper-bounding process proposed in [41]. The necessary bisection of the SIC parameter domains is performed transparently, and helps improving the efficiency of the method. This allows taking full benefits of existing efficient implementations of branch-and-bound algorithms: Experimental results in this paper are obtained using the NLP solver IBEX [14], and they show that the proposed branch-and-bound algorithm is competitive with the distributed implementation of [46, 19] based on BARON [54], hence re-establishing the interest of branch-and-bound algorithms for solving SIPs.

The paper is organized as follows: Section 2 presents some background about interval analysis, numerical constraint solving and branch-and-bound algorithms for NLPs. Section 3 presents the extension to SICs of standard bounding methods required in branch-and-bound algorithms. Finally Section 4 presents experimental results on academic benchmarks from the literature.

2. Background

Branch-and-bound algorithms solve an optimization problem by recursively considering subproblems with smaller domains. Computing upper and lower bounds for these subproblems become simpler as subdomains get smaller, hence eventually providing converging lower and upper bounds. Interval analysis allows computing cheap verified upper and lower bounds of a nonlinear function over a box domain, which converges to the actual range as the size of the domains decreases. Therefore, it is used more or less directly in most implementations of the branch-and-bound algorithm dedicated to NLPs. Basics of interval analysis are briefly recalled in Section 2.1, followed by a generic description of the branch-and-bound algorithm in Section 2.3.

2.1. Interval analysis

Interval analysis (IA) is a branch of numerical analysis born in the 1960's [48]. It replaces computations with real numbers by computations with intervals of real numbers, providing a framework for handling uncertainties and verified computations (see [49, 36, 39] for a survey).

An interval is a closed connected subset of $\mathbb{R}$. Intervals are denoted by bracketed symbols, e.g. $[x] \subseteq \mathbb{R}$. When no confusion is possible, lower and upper bounds of an interval $[x]$ are denoted by $\underline{x} \in \mathbb{R}$ and $\overline{x} \in \mathbb{R}$, with $\underline{x} \leq \overline{x}$, i.e. $[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leq x \leq \overline{x}\}$. Hence, a real number x will be identified with the degenerated interval $[x, x]$. Interval vectors, also called boxes, are equivalently defined as vectors of intervals $[x] = ([x]_1, \ldots, [x]_n) \in \mathbb{IR}^n$, with $[x]_i \in \mathbb{IR}$, or as intervals of vectors $[x] = [\underline{x}, \overline{x}] \in \mathbb{IR}^n$, with $\underline{x}, \overline{x} \in \mathbb{R}^n$. Similarly, interval matrices are equivalently defined as matrices of intervals or intervals of matrices: $[A] = ([a]_{ij})_{1 \leq i \leq n, 1 \leq j \leq m} = [\underline{A}, \overline{A}] \in \mathbb{IR}^{n \times m}$. The lower and upper bound of an interval object are also denoted using the operators inf and sup. The midpoint of an interval or box $[x]$ is $\operatorname{mid}[x] := 0.5(\underline{x} + \overline{x})$. The midpoint is computed approximately using floating point operations, although it is assumed that $\operatorname{mid}[x] \in [x]$ holds [31]. The width of a box $[x]$ is denoted by $\operatorname{wid}[x]$, and is the length of a longest edge, i.e., $\operatorname{wid}[x] = \max_i(\overline{x_i} - \underline{x_i})$. We consider only the maximum norm, so the unit ball B is the cartesian product of intervals $[-1, 1] \times \cdots \times [-1, 1]$.

The convex hull of a set $E \subseteq \mathbb{R}^n$ is denoted by $\operatorname{conv} E$. It is approximated by the interval hull $\bigvee E$ which is the smallest box that contains E (and thus $\operatorname{conv} E$). E.g. $\bigvee\{x \in \mathbb{R}^2 : x_1^2 + x_2^2 \leq 1\} = ([-1, 1], [-1, 1])^T$. The interval hull of two sets E_1 and E_2 is denoted by $E_1 \vee E_2$, and is by definition the smallest box that contains both E_1 and E_2.

A paving $\mathcal{X} = \{[x_1], [x_2], \ldots, [x_k]\} \subseteq \mathbb{IR}^n$ is a finite set of boxes, which allows enclosing a set more accurately than a single box. In the rest of the paper, they are denoted by calligraphic letters. The union $\bigcup \mathcal{X}$ of a paving is the union of the boxes it contains, i.e., $x \in \bigcup \mathcal{X} \iff \exists [x] \in \mathcal{X}, x \in [x]$.

An interval extension of a real function $f : \mathbb{R}^n \to \mathbb{R}$ is an interval function $[f] : \mathbb{IR}^n \to \mathbb{IR}$ that satisfies $[f]([x]) \supseteq f([x])$, where $f([x])$ is the usual extension of a real function on subsets of its domain, i.e. $f([x]) := \{f(x) : x \in [x] \cap \operatorname{dom}(f)\}$. The construction of interval extensions relies on interval arithmetic (IA), which defines how to compute any elementary operation with interval operands. Elementary binary operators $\circ \in \{+, \times, -, \div\}$ are extended to intervals as follows:

$$[\underline{x}, \overline{x}] \circ [\underline{y}, \overline{y}] := \bigvee\{x \circ y : x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}], (x, y) \in \operatorname{dom}(\circ)\}. \tag{4}$$

E.g., $[1, 2] + [3, 5] = [4, 7]$ or $[1, 2]/[-3, 5] = [-\infty, -\frac{1}{3}] \vee [\frac{1}{5}, \infty] = [-\infty, \infty]$. Furthermore, continuous unary elementary functions like exp, log, sin, etc., are also extended to intervals, leading as well to analytical formulas like $\exp[x] = [\exp \underline{x}, \exp \overline{x}]$. Since real numbers are identified with degenerated intervals, IA actually generalizes real arithmetic, and mixed operations like $1 + [1, 2] = [2, 3]$ are interpreted using IA.

The *natural* interval extension of a real function consists in replacing the elementary operations it involves by their interval counterparts. The fundamental theorem of interval analysis states that such an interval evaluation of a real function gives rise to an interval extension of the original real function. E.g. the interval extension of the standard scalar product satisfies $[x]^T[y] \supseteq \{x^T y : x \in$

$[x], y \in [y]\}$. When the expression of a function contains only one occurrence of each variable, its interval evaluation by natural extension is optimal, i.e., it computes the interval hull of the range of the function with no over-estimation. E.g. $[x]^T[y] = \bigvee\{x^T y : x \in [x], y \in [y]\}$. However, if its expression contains several occurrences of some variables, its natural extension is not optimal in general. Nevertheless, under the mild hypothesis that the function is Lipschitz continuous inside the interval arguments, the overestimation of its natural extension decreases proportionally to the width of the interval arguments, which is generally sufficient to enforce the convergence of algorithms although often not enough to avoid some cluster effects.

When interval arguments are small, the *mean-value* interval extension usually gives rise to better enclosures whose overestimation decreases quadratically with respect to the width of the interval arguments. For a differentiable real function $f : \mathbb{R}^n \to \mathbb{R}$, the mean-value extension is defined as $[f](\tilde{x}) + [Df]([x])([x] - \tilde{x}) \supseteq \{f(x) : x \in [x]\}$, where $[f]$ and $[Df]$ are the (usually natural) interval extensions of the function and its derivative, and $\tilde{x} \in [x]$.

An interval extension $[f]$ of a function f is called convergent inside a bounded box $[x_0]$ if for any exhaustive sequence of boxes as defined in [41], i.e., sequences $([x_k])_{k\in\mathbb{N}}$ of nested non-empty boxes whose widths converges to zero, $\mathrm{wid}[f]([x_k])$ also converges to zero. It is well-known that the natural extension is convergent provided that interval extensions of elementary operations involved in the expression are convergent.

Remark 2. *The Minkowski sum between sets somehow generalizes the interval addition to arbitrary sets. In the rest of the paper, we use the Minkowski sum in particular to inflate a set $E \subseteq \mathbb{R}^n$ with a scaled maximum norm ball ϵB, e.g., $E + \epsilon B = \{x + \epsilon y : x \in E, \|y\|_\infty \leq 1\}$.*

2.2. Numerical constraint programming

Numerical constraint programming [8] addresses the problem of finding the solution set $\{x \in [x] \subseteq \mathbb{R}^n : h(x) = 0, g(x) \leq 0\}$ to a constraint system defined by a conjunction of equalities $h(x) = 0$ and inequalities $g(x) \leq 0$ within a given box domain $[x]$.

Though part of the research in this area considers other topics (e.g., search strategies or applications), most of the work in the numerical constraint programming community has concentrated on the definition of efficient contracting operators (contractors in short). A contractor $[C_c] : \mathbb{IR}^n \to \mathbb{IR}^n$ for a constraint $c(x)$ (typically $c(x) \iff h(x) = 0$ or $c(x) \iff g(x) \leq 0$, or conjunctions of such constraints) contracts a domain without loosing any solution, i.e.,

$$x \in [x] \wedge c(x) \implies x \in [C_c]([x]). \tag{5}$$

The simple interval evaluation of a constraint gives rise to a "all-or-nothing" contractor as it either fully rejects the domain or leaves it intact: The tests for equalities $0 \notin [h]([x])$ or for inequalities $[g]([x]) > 0$ allow fully rejecting the domain $[x]$. Stronger typical contractors are: HC4-revise [7] which employs

operator-wise evaluation and projection in order to enforce hull-consistency; BC3-revise [7] which uses univariate Newton steps to find extremal solutions within an interval domain and enforces box-consistency [9]; Octum [15] or MOHC-revise [1] which exploits monotonicity and combines hull and box consistencies. All these contractors usually consider one constraint at a time and must thus be repeated, typically following an AC3-like fix-point propagation principle [9]. Because this propagation mechanism may exhibit slow convergence, heuristic stopping criteria are often employed, e.g., when the obtained contraction drops below a given (relative) threshold called the improvement factor.

More global operators comprise: Peeling (or shaving) operators [18, 42], which iteratively discard slices on the boundaries of an interval domain using local consistency based operators on all the constraints; Constructive interval disjunction (CID) [57, 53] which considers a partition of the domain $[x]$, propagates other contracting operators (usually HC4-revise) onto each part, and takes the hull of the contracted part as the new domain; And X-Newton [2], which generalizes the standard interval Newton [49] dedicated only to square systems of equations, considers linear enclosures of the equalities and inequalities of the problem using interval evaluation of their derivatives and usually solves them using several calls to the Simplex algorithm.

Contractors have proven to be powerful tools for reducing the search space and avoiding large search efforts, allowing to address challenging problems, in particular in control and robotics [36].

2.3. Generic branch-and-bound algorithm for NLPs

This section presents the branch-and-bound algorithm dedicated to solving NLPs (1). We suppose the inequality constraints $g(x) \leq 0$ contain at least bound constraints that define an initial bounded box domain $[x^0]$. This algorithm interleaves lower and upper bounding operations with branching steps.

Given a subdomain $[x] \subseteq [x^0]$, lower bounding consists in finding a lower bound of the objective function for the subproblem restricted to the subdomain $[x]$. It is usually based on tractable relaxations of the subproblem. Upper bounding consists in finding a feasible point, usually lying in the subdomain $[x]$ although this subdomain is sometimes used only as an initial guess to find a feasible point that may lie outside. The objective value of such a feasible point indeed provides a guaranteed upper bound on the minimum of the NLP. Upper bounding is usually based either on tractable restrictions of the subproblem or on local optimization with a feasibility check. Both processes allow proving that a subdomain needs not be explored anymore if its (local) lower bound is greater than the current (global) upper bound.

Branching consists in splitting the current subdomain $[x]$ into two covering non-overlapping subdomains $[x']$ and $[x'']$, i.e., $[x] = [x'] \cup [x'']$ and $[x'] \cap [x'']$ has zero measure. The algorithm will then apply recursively bounding operations and branching steps to both $[x']$ and $[x'']$. Branching is mandatory for the convergence of the algorithm because bounding techniques are pessimistic, but their inaccuracy decreases as the size of the current subdomain decreases.

The convergence of the lower bounding process is usually easily obtained using convergent lower bounding techniques and a search strategy that consists in selecting the current subdomain that has the least computed upper bound among the remaining subdomains to be explored. However, the convergence of the upper bounding process usually requires some regularity assumption.

The contractors defined in the previous section have been included in branch-and-bound algorithms [33, 40, 52, 59] in order to improve the lower bounding process. Instead of simply discarding a subdomain when its lower bound is greater than the current upper bound f^*, it is contracted with respect to the constraints $h(x) = 0$, $g(x) \leq 0$ and $f(x) \leq f^*$. This leads to a more powerful pruning since contracting according to the single constraint $f(x) \leq f^*$ actually encodes the original rejection process using only upper and lower bounds. Other contractors specific to NLPs are: Optimality conditions [40, 50, 29] that discard subdomains that cannot contain any local optima according to first or second order optimality criteria; And monotonicity tests [40, 33] which reduce a domain to one of its bounds if the objective function is proved to monotonically decrease/increase along the corresponding dimension.

3. Extension of bounding methods to SICs

In this section we consider a SIC $g(x) \leq 0$ with

$$g(x) = \max_{y \in [y^0]} \tilde{g}(x, y). \tag{6}$$

The box-domain of the lower-level program is now denoted $[y^0]$ to emphasize the fact that it is the initial lower-level program box-domain. We also consider an initial decision variable domain $[x^0]$, which is supposed to be bounded and will be used for studying the convergence of the bounding processes. Since $\tilde{g}$ is supposed to be continuously differentiable inside $([x_0], [y_0])$, it is also Lipschitz continuous and we denote by $\tilde{L}_0$ a Lipschitz constant for $\tilde{g}$ inside $([x_0], [y_0])$. As a consequence of Theorem 2.1 in [17], g is also Lipschitz, and we denote by L its Lipschitz constant.

We define for such a SIC the main techniques that allow its treatment by a standard branch-and-bound algorithm, namely:

- Interval evaluation of g and its generalized gradient ∂g (Section 3.1);
- Contractors based on linear relaxations and constraint programming applied to a discretization of the SIC (Section 3.2);
- Upper bounding by linear restrictions and directional search (Section 3.3).

Useful parameter values $y \in [y^0]$ are those that maximize $\tilde{g}(x, y)$. This can be interpreted as a parametric maximization problem, where decision variables x now play the role of parameters. We therefore define the constraint parameter maximizer y_x^* for any decision variable value x as follows:

$$y_x^* := \operatorname*{argmax}_{y \in [y^0]} \tilde{g}(x, y). \tag{7}$$

As illustrated by Example 1, y_x^* may contain several maximizers, in which case $g(x)$ may be non-differentiable. In order to perform interval evaluations, this definition is extended to decision variable domains as follows:

$$y_{[x]}^* := \bigcup_{x \in [x]} y_x^*. \tag{8}$$

Then, $g([x]) \subseteq \tilde{g}([x], y_{[x]}^*) = \{\tilde{g}(x, y) : x \in [x], y \in y_{[x]}^*\}$. This inclusion is not an equality in general because the vectors $x \in [x]$ and $y \in y_{[x]}^*$ are considered independently in $\tilde{g}([x], y_{[x]}^*)$, but it is the best enclosure we can aim for. It is therefore crucial that the pessimism of this enclosure decreases when the size of the decision variable domain decreases. This is proved by the following proposition, which is provided here as an initial motivation but is not used explicitly in the rest of the paper.

Proposition 3. *We have* $\operatorname{wid} \tilde{g}([x], y_{[x]}^*) \leq \operatorname{wid} g([x]) + \tilde{L}_0 \operatorname{wid}[x]$.

Proof. Consider some arbitrary $x \in [x]$ and $y \in y_{[x]}^*$. Since $y \in y_{[x]}^*$, there exists $x' \in [x]$ such that $y \in y_{x'}^*$, so $\tilde{g}(x', y) = g(x')$. Now, $|g(x') - \tilde{g}(x, y)| = |\tilde{g}(x', y) - \tilde{g}(x, y)| \leq \tilde{L}_0 \|x - x'\|_\infty \leq \tilde{L}_0 \operatorname{wid}[x]$. In other words, every $\tilde{g}(x, y) \in \tilde{g}([x], y_{[x]}^*)$ is distant of at most $\tilde{L}_0 \operatorname{wid}[x]$ of some $g(x') \in g([x])$, which together with $g([x]) \subseteq \tilde{g}([x], y_{[x]}^*)$ proves the statement. □

All the bounding techniques presented in this section require a description of the set $y_{[x]}^*$ for a given box $[x]$. The more accurate this description, the more efficient the bounding process. This description takes the form of a paving $\mathcal{Y}_{[x]} \subseteq \mathbb{IR}^m$, i.e., a finite set of parameter boxes $\mathcal{Y}_{[x]} = \{[y^1], [y^2], \ldots\}$, which satisfies the following requirement:

$$y_{[x]}^* \subseteq \cup \mathcal{Y}_{[x]} \subseteq [y^0]. \tag{9}$$

Remark 4. *The subscript decision variable domain $[x]$ emphasizes the dependence of the paving with respect to the current decision variable domain during the branch-and-bound process. For clarity, only one SIC is considered in this section and $[y^0]$ is not written explicitly with the paving $\mathcal{Y}_{[x]}$, but such a paving is computed and stored for each SIC of a SIP.*

Requirement (9) forces $\mathcal{Y}_{[x]}$ to contain all the parameter values that maximize the SIC value for any decision variable value in the considered decision variable domain $[x]$. It is then straightforward that, given an arbitrary $x \in [x]$,

$$\forall y \in [y^0],\ \tilde{g}(x, y) \leq 0 \iff \forall [y] \in \mathcal{Y}_{[x]},\ \forall y \in [y],\ \tilde{g}(x, y) \leq 0, \tag{10}$$

or equivalently,

$$g(x) = \max_{y \in [y^0]} \tilde{g}(x, y) \leq 0 \iff \max_{[y] \in \mathcal{Y}_{[x]}} \max_{y \in [y]} \tilde{g}(x, y) \leq 0. \tag{11}$$

Updating a valid parameter paving, i.e., that satisfies Requirement (9), is done during the branch-and-bound algorithm. Note that a paving $\mathcal{Y}_{[x]}$ that

is valid for a given box $[x]$ is also valid for all subboxes of $[x]$. Hence, the same paving could in principle be shared all along a given branch of the search tree, since both branching and bounding steps only reduce the decision variable domain $[x]$. However, in order to enforce the convergence of the bounding methods, and therefore of the overall algorithm, the parameter paving has to be refined when the decision variable domain $[x]$ is reduced. This refinement process is described in Section 3.4. It consists in splitting some boxes in the paving and removing boxes that can be proved to not contain any parameter maximizer. Informally, we expect such a refinement process to enforce the paving to converge to active parameter values. Formally, the refinement process is called convergent if for any exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$, with $\cap_{k=0}^{\infty}[x_k] = \{x_\infty\}$, we have both $\lim_{k\to\infty} \operatorname{wid} \mathcal{Y}_{[x_k]} = 0$ with

$$\operatorname{wid} \mathcal{Y}_{[x]} := \max_{[y]\in\mathcal{Y}_{[x]}} \operatorname{wid}[y], \tag{12}$$

and $\cup\mathcal{Y}_{[x_k]}$ converges to $y^*_{x_\infty}$ for the Hausdorff distance, i.e.,

$$\lim_{k\to\infty} \min\{\epsilon \geq 0 : \cup\mathcal{Y}_{[x_k]} \subseteq y^*_{x_\infty} + \epsilon B\} = 0. \tag{13}$$

The condition (13) is sufficient for the convergence of the Hausdorff distance because $\mathcal{Y}_{[x_k]} \supseteq y^*_{x_\infty}$ holds for all $k \in \mathbb{N}$, so $y^*_{x_\infty} \subseteq \cup\mathcal{Y}_{[x_k]} + \epsilon B$ holds trivially. The refinement process presented in Section 3.4 will be proved to be convergent.

Finally, the convergence of the branch-and-bound algorithm dedicated to NLP presented in [41] will be extended to SIP in Section 3.5.

3.1. Interval evaluation of SIC and their generalized gradient

In this section, we define interval extensions of the SIC (6) and its generalized gradient. They are computed using interval extensions $[\tilde{g}]$ and $[\nabla_x \tilde{g}]$ of the function $\tilde{g}$ and its gradient with respect to x evaluated on decision variable domain $[x]$ and its associated parameter paving $\mathcal{Y}_{[x]}$. To this end, given a paving $\mathcal{Y}_{[x]}$, we define

$$\underline{\mathcal{Y}}_{[x]} := \max_{[y]\in\mathcal{Y}_{[x]}} \inf\, [\tilde{g}]([x],[y]) \tag{14}$$

$$\overline{\mathcal{Y}}_{[x]} := \max_{[y]\in\mathcal{Y}_{[x]}} \sup\, [\tilde{g}]([x],[y]), \tag{15}$$

i.e., $\underline{\mathcal{Y}}_{[x]}$, respectively $\overline{\mathcal{Y}}_{[x]}$, is the greatest lower bound, respectively greatest upper bound, of the evaluation of $[\tilde{g}]$ for the current decision variables domain $[x]$ and all parameters domains $[y]$ of the parameter paving $\mathcal{Y}_{[x]}$. Some interval extensions of g and ∂g are given by the following proposition.

Proposition 5. *Consider interval extensions $[\tilde{g}]$ of $\tilde{g}$, and $[\nabla_x \tilde{g}]$ of $\nabla_x \tilde{g}$. Then*

$$[g]([x]) := [\underline{\mathcal{Y}}_{[x]}, \overline{\mathcal{Y}}_{[x]}] \tag{16}$$

$$[\partial g]([x]) := \bigvee_{[y]\in\mathcal{Y}_{[x]}} [\nabla_x \tilde{g}]([x],[y]) \tag{17}$$

are interval extensions of g and ∂g, respectively.

Proof. Consider a box $[x]$ and a point $x \in [x]$. Since $\max_{y\in[y]} \tilde{g}(x,y)$ is in $[\tilde{g}](x,[y]) \subseteq [\tilde{g}]([x],[y])$, we have

$$\inf[\tilde{g}]([x],[y]) \leq \max_{y\in[y]} \tilde{g}(x,y) \leq \sup[\tilde{g}]([x],[y]). \tag{18}$$

Therefore,

$$\max_{[y]\in\mathcal{Y}_{[x]}} \inf[\tilde{g}]([x],[y]) \leq \max_{[y]\in\mathcal{Y}_{[x]}} \max_{y\in[y]} \tilde{g}(x,y) \leq \max_{[y]\in\mathcal{Y}_{[x]}} \sup[\tilde{g}]([x],[y]). \tag{19}$$

Finally, using (11), we obtain $\underline{\mathcal{Y}}_{[x]} \leq g(x) \leq \overline{\mathcal{Y}}_{[x]}$. Since this holds for all $x \in [x]$, we have proved that (16) is actually an interval extension of g.

By Theorem 2.1 in [17] we have $\partial g(x) = \operatorname{conv}\{\nabla_x \tilde{g}(x,y) : y \in y_x^\star\}$. Using Requirement (9), we obtain $\partial g(x) \subseteq \operatorname{conv}\{\nabla_x \tilde{g}(x,y) : y \in \bigcup \mathcal{Y}_{[x]}\}$. Then, using basic set properties and the fact that the interval hull includes the convex hull, we obtain the following inclusions:

$$\partial g(x) \subseteq \operatorname{conv} \bigcup_{[y]\in\mathcal{Y}_{[x]}} \{\nabla_x \tilde{g}(x,y) : y \in [y]\} \tag{20}$$

$$\subseteq \operatorname{conv} \bigcup_{[y]\in\mathcal{Y}_{[x]}} [\nabla_x \tilde{g}]([x],[y]) \tag{21}$$

$$\subseteq \bigvee_{[y]\in\mathcal{Y}_{[x]}} [\nabla_x \tilde{g}]([x],[y]). \tag{22}$$

Therefore (17) is an interval extension of $\partial g(x)$. □

It is crucial for the convergence of the branch-and-bound algorithm that the interval extension (16) is convergent. The following proposition provides a sufficient condition for this, and will be used in Section 3.5 to prove the convergence of the lower bounding process of the overall algorithm.

Proposition 6. *Suppose that the interval extension $[\tilde{g}]$ is convergent and that the refinement process of the parameter paving is convergent. Then the interval extension* (16) *is convergent.*

Proof. Consider an arbitrary exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$ and the corresponding parameter pavings $\mathcal{Y}_{[x_k]}$. Define $[y_k]$ as the maximizer of the problem (15). From (14) we see that $[g]([x_k]) \subseteq [\tilde{g}]([x_k],[y_k])$ obviously holds. By Definition (12) we have $\operatorname{wid}[y_k] \leq \operatorname{wid} \mathcal{Y}_{[x_k]}$, and since the parameter paving refinement process is supposed convergent, we have $\operatorname{wid} \mathcal{Y}_{[x_k]}$ converges to zero. Therefore, $\operatorname{wid}[y_k]$ also converges to zero. Finally, because the interval extension $[\tilde{g}]$ is supposed convergent, we have $\operatorname{wid}[\tilde{g}]([x_k],[y_k])$ converges to zero and so does $\operatorname{wid}[g]([x_k])$. □

This is illustrated in the following example.

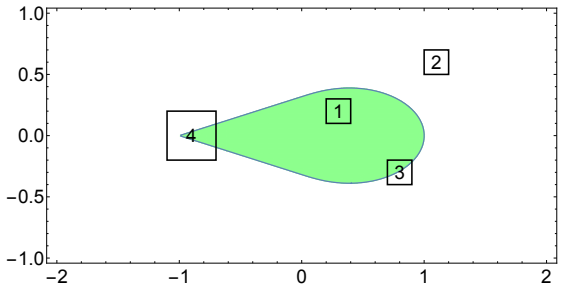

Figure 2: In light green, feasible set of the SIC of Example 7. Four boxes are shown: A feasible box $[x^1]$, an infeasible box $[x^2]$, two boundary boxes $[x^3]$ and $[x^4]$.

	$[x^1]$	$[x^2]$	$[x^3]$	$[x^4]$
$[g]^1([x])$	$[-2.65, 0.65]$	$[-5.10, 3.10]$	$[-3.56, 1.56]$	$[-2.93, 0.93]$
$[g]^{100}([x])$	$[-0.72, -0.18]$	$[0.41, 1.01]$	$[-0.22, 0.28]$	$[-0.95, 0.75]$
$[\partial g]^1([x])$	$[-1, 1]$ $[-4.15, 4.15]$	$[-1, 1]$ $[-4.15, 4.15]$	$[-1, 1]$ $[-4.15, 4.15]$	$[-1, 1]$ $[-4.15, 4.15]$
$[\partial g]^{100}([x])$	$[-1, 0.91]$ $[0.86, 3.21]$	$[0.30, 0.73]$ $[1.43, 2.21]$	$[0.48, 0.93]$ $[-1.95, -0.74]$	$[-1, -0.98]$ $[-3.21, 3.21]$

Table 1: Results of interval evaluation of the function and its generalized gradient from Example 7.

Example 7. *We consider the SIC from Example 1:*

$$\forall y \in [-\pi, \pi], x_1 \cos(y) + x_2(\sin(y) + y) - 1 \leq 0. \tag{23}$$

Its feasible set is depicted in Figure 2, where four boxes are considered. The interval evaluations $[g]^k$ and $[\partial g]^k$ given in Table 1 are computed using the paving $\mathcal{Y}_{[x]}$ obtained by splitting $[y^0]$ into k sub-intervals of equal width, and filtering them following the rules given in Section 3.4, which maintain Requirement (9)*.*

We can see that the more accurate is the paving $\mathcal{Y}_{[x]}$ the sharper are the interval extensions. In particular, the interval evaluation $[g]^{100}([x^1])$ is negative, and therefore $[x^1]$ is proved to be feasible; the interval evaluation $[g]^{100}([x^2])$ is positive, and therefore $[x^2]$ is proved to be infeasible.

The generalized gradient enclosure (17) readily allows using first order rejection tests [29] to reject decision variable domains that are proved not to contain any local minimizer, and hence no global one. This is illustrated by the following example.

Example 8. *Suppose we minimize the objective $f(x) = x_1$ subject to the SIC* (23)*, so the global minimizer $x^* = (-1, 0)$ is the only local minimizer. Consider the box $[x^3]$ and the generalized gradient enclosure $[\partial g]^{100}([x^3])$ provided in Table 1. Then the first order rejection tests proposed in [29] consists in forming the interval matrix*

$$\begin{pmatrix} [\partial g]([x^3]) & [\partial f]([x^3]) \end{pmatrix} = \begin{pmatrix} [0.48, 0.93] & 1 \\ [-1.95, -0.74] & 0 \end{pmatrix} \tag{24}$$

and checking whether it is full rank or not. Here, the interval matrix (24) *is clearly full rank, therefore* $[x^3]$ *is proved not to contain any local minimizer and can be rejected.*

Finally, when a SIC has to be evaluated at a point $\tilde{x} \in [x]$, i.e., at a degenerated interval $[\tilde{x}, \tilde{x}] \subseteq [x]$, using the paving $\mathcal{Y}_{[x]}$ induces a strong pessimism. This happens for example for finding feasible points, see Section 3.3.1. In this case, the paving $\mathcal{Y}_{[x]}$ is temporarily refined to $\mathcal{Y}_{[\tilde{x},\tilde{x}]}$ using the refinement process described in Section 3.4. The refined paving satisfies Requirement (9), i.e.,

$$\operatorname*{argmax}_{y \in [y^0]} \tilde{g}(\tilde{x}, y) \subseteq \mathcal{Y}_{[\tilde{x},\tilde{x}]}, \tag{25}$$

and therefore allows obtaining a sharp interval evaluation of $g(\tilde{x})$.

3.2. Contractors

In the previous section, we have seen that the test $[g]([x]) > 0$ based on the SIC interval evaluation can be used to reject an infeasible domain, hence defining an all-or-nothing contractor for SIC. It is well known that this simple contractor is not efficient, although sufficient to enforce the convergence of the lower bounding process in the branch-and-bound algorithm. The first order rejection test based on the interval evaluation of the generalized gradient is also an all-or-nothing contractor, which turns out to be quite efficient in decreasing the cluster effect, but which is still not enough on its own.

It is standard to build relaxations of SICs by sampling the parameter domains: We build contractors for SICs by using standard contractors dedicated to nonlinear inequality constraints (e.g., HC4, XTaylor, cf. Section 2.2) applied to relaxations of the form $\tilde{g}(x, \tilde{y}) \leq 0$ for a finite set of samples $y \in [y^0]$. In the context of SIPs, this usage of standard contractors applied to relaxations by sampling is strengthened here by two key features: First, the choice of the parameter values $\tilde{y}$ is obviously critical to obtain efficient contractions. We take advantage of the paving $\mathcal{Y}_{[x]}$ to choose effective values $\tilde{y}$ of parameters by considering the midpoints of each box in $\mathcal{Y}_{[x]}$. Since the paving is maintained during the search in such a way that it converges to $y^*_{[x]}$ in the sense of (12) and (13) (see Section 3.4 for the details of the refinement process, and the proof of its convergence), these samples will also converge to optimal parameter values, leading to near-optimal contractions when the decision variables domain is small enough. Second, each parameters sample gives rise to one SIC relaxation, which are all included inside the constraint propagation algorithm.

The previous relaxation scheme can finally be enhanced using Blankenship's method [12], which allows discovering useful parameter values to build relaxations. Indeed, the minimization of the objective minoration subject to the constraint's linear relaxations performed during the constraint propagation outputs both a lower bound on the objective and the corresponding minimizer x_{relax}. Blankenship proposes to use this relaxation minimizer to discover useful parameter values by maximizing the contraint with respect to parameters for the

decision variables fixed to x_{relax}. In our algorithm, this auxiliary parameter maximization problem is performed by simply evaluating $[g](x_{\text{relax}})$, which consists in refining the initial paving $\mathcal{Y}_{[x]}$ to $\mathcal{Y}_{[x_{\text{relax}},x_{\text{relax}}]}$ as in (25). Provided that the maximizer is regular, the usage of the interval Newton operator in the paving refinement process results in tiny parameter boxes, one of them containing $y^*_{x_{\text{relax}}}$. Their midpoints are used as additional parameter values to build relaxations. Finally, the Blankenship process outputs one parameter value, while several of them are usually needed to build an accurate relaxation. Therefore, the computed Blankenship vectors are stored in a queue of size $2n$, this size being motivated by the fact that there are generically at most n active constraints at a minimizer. This queue is updated following the branches of the search tree, and the Blankenship parameter values are used for constraint propagation together with the midpoints of the boxes from the parameter paving.

3.3. Upper bounding

In this section, we extend three methods for finding feasible points of NLPs to SICs: The midpoint interval evaluation (Section 3.3.1), the corner linear restriction [3] (Section 3.3.2) and the directional search [41] (Section 3.3.3). Those three methods output a feasible point x_{feas} when they succeed. In this case, performing a simple dichotomous line-search between the non-feasible optimal solution of the linear relaxation x_{relax} and the feasible solution of the directional linear restriction and evaluating

$$[g]\big((1-\lambda)x_{\text{relax}} + \lambda x_{\text{feas}}\big) \tag{26}$$

allows cheaply discovering better feasible points.

3.3.1. Midpoint evaluation

The most obvious way of finding a feasible point for the SIC (6) inside the decision variable domain $[x]$ is to perform its interval evaluation at one point of this domain, usually the midpoint. Computing $[y] := [g](\text{mid}[x])$ is done using Proposition 5 and $\sup[y] \leq 0$ is a sufficient condition for $\text{mid}[x]$ to be feasible. The midpoint evaluation technique is presented here only for completeness, since it is far less efficient than the subsequent methods.

Remark 9. *The feasibility of several constraints is obtained by checking them independently.*

3.3.2. Corner linear restriction

The interval evaluation $[d] = [\nabla g]([x])$ of the gradient of a differentiable function $g(x)$ allows building a piecewise linear upper-bounding function of $g(x)$ using the so-called interval centered form:

$$g(x) \leq \sup\Big(g(\tilde{x}) + [d]^T(x-\tilde{x})\Big), \tag{27}$$

which is valid for an arbitrary expansion point $\tilde{x} \in [x]$. The right-hand-side of (27) is actually piecewise linear because the expression of the interval product

$[d]^T(x-\tilde{x})$ depends on the sign of each element of $x-\tilde{x}$, and when these signs are fixed it gives rise to a linear function with respect to x. This piecewise linear restriction (27) therefore becomes linear if the expansion point $\tilde{x}$ of the centered form is chosen to be a corner of the domain [3]. For example, if the expansion point is the lower bound of the domain, i.e., $\tilde{x} = \underline{x}$, then the right hand side of (27) becomes $g(\underline{x}) + \overline{d}^T(x-\underline{x})$, that is, a linear upper-bounding function.

The following theorem extends this process to SICs. Given $\alpha \in \{0,1\}^n$ we define the α-corner of a box $[x]$ by $x_\alpha = \inf[x] + \operatorname{diag}(\alpha)\operatorname{wid}[x]$. The opposite corner of x_α is obviously the $(1-\alpha)$-corner.

Theorem 10. *Let $[x] \in \mathbb{IR}^n$, $[y] \in \mathbb{IR}^m$, $\alpha \in \{0,1\}^n$ and x_α be the corresponding corner of $[x]$. Define $\overline{z}^{[y]} = \sup\,[\tilde{g}](x_\alpha,[y])$ and $d^{[y]}_{1-\alpha}$ as the $(1-\alpha)$-corner of $[d]^{[y]} = [\nabla_x \tilde{g}]([x],[y])$. Then, for all $x \in [x]$*

$$\max_{y\in[y]} \tilde{g}(x,y) \le \overline{z}^{[y]} + \left(d^{[y]}_{1-\alpha}\right)^T(x-x_\alpha). \tag{28}$$

Proof. The centered form in x_α of g is expressed as

$$\tilde{g}(x,y) \in [\tilde{g}](x_\alpha,y) + \left([\nabla_x \tilde{g}]([x],y)\right)^T(x-x_\alpha) \tag{29}$$

$$\subseteq [\tilde{g}](x_\alpha,[y]) + \left([\nabla_x \tilde{g}]([x],[y])\right)^T(x-x_\alpha). \tag{30}$$

Since we are interested in the upper bound of the right-hand side, we must determine an upper bound of $\left([\nabla_x \tilde{g}]([x],[y])\right)^T(x-x_\alpha) = \left([d]^{[y]}\right)^T h$, where $h = x - x_\alpha$. For each component i, h_i is positive if $\alpha_i = 0$ and negative if $\alpha_i = 1$. Let $P = \{i : \alpha_i = 0\}$ and $N = \{i : \alpha_i = 1\}$. Then

$$\left([d]^{[y]}\right)^T h = \sum_{i\in P}[d_i]h_i + \sum_{i\in N}[d_i]h_i \tag{31}$$

$$\le \sum_{i\in P}\overline{d_i}h_i + \sum_{i\in N}\underline{d_i}h_i \tag{32}$$

$$= \left(d_{1-\alpha}\right)^T h. \tag{33}$$

Since this is valid for all $y \in [y]$,

$$\max_{y\in[y]} \tilde{g}(x,y) \le \overline{z}^{[y]} + \left(d^{[y]}_{1-\alpha}\right)^T(x-x_\alpha), \tag{34}$$

which concludes the proof. □

The following corollary then follows from considering the SIC (6) and its equivalent expression (10).

Corollary 11. *For an arbitrary $[x] \in \mathbb{IR}^n$ and an arbitrary corner x_α of this box, the polyhedron*

$$\{x \in [x] : \overline{z}^{[y]} + \left(d^{[y]}_{1-\alpha}\right)^T(x-x_\alpha) \le 0,\ [y] \in \mathcal{Y}_{[x]}\} \tag{35}$$

where $\overline{z}^{[y]}$ and $d^{[y]}_{1-\alpha}$ are defined as in Theorem 10, is a restriction of the feasible set of the SIC (6).

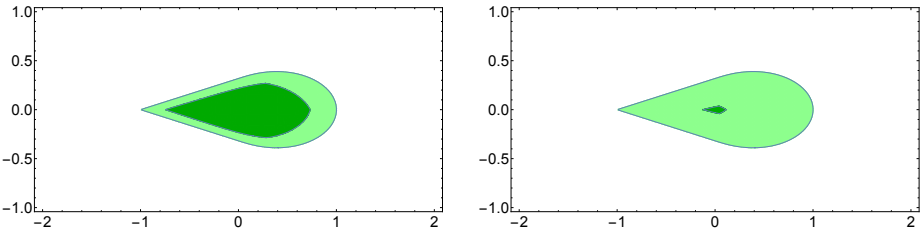

Figure 3: In light green, the feasible set of the SIC of Example 7. In dark green, the restriction obtained using Corollary 11.

Remark 12. *A restriction for several constraints is obtained by intersecting their restrictions, i.e., considering all constraints in* (35).

Example 13. *We consider again the SIC of Example 7 and apply Corollary 11 using a homogeneous paving of* $[y^0]$ *containing* 50 *boxes. The expansion point is the lower bound of the domain, i.e.,* $\tilde{x} = \underline{x}$. *Corollary 11 gives rise to the restrictions depicted in Figure 3: On the left-hand side graphic, the domain is* $[x] = ([-1,1],[-1,1])^T$*; on the right hand side graphic, the domain is* $[x] = ([-3,3],[-3,3])^T$*, which gives rise to a sensibly smaller feasible restriction.*

This example illustrates a drawback of the corner linear restriction: The larger the domain, the farer the expansion point from the feasible set, and thus the cruder the restriction. It may happen that this restriction is actually empty, which makes it useless in this case.

3.3.3. Directional search

Another approach to discover feasible points of NLP was proposed in [41], which is proved to provide convergent upper bounds in the context of NLP under some mild constraint qualification condition. It consists in choosing a finite set of vectors $G \subseteq \mathbb{R}^n$ that is supposed to approximate the gradients of the active constraints at some arbitrary point in the decision variable domain. Then a direction $u \in \mathbb{R}^n$ that potentially points toward feasibility is obtained by solving the linear problem

$$u = \operatorname*{argmin}_{u \in B} \big(\max_{d \in G} d^T u \big), \tag{36}$$

i.e., u minimizes the worst case ascent with respect to each constraint. As noted in [41], this problem is equivalent to $\min_{u \in \mathbb{R}^n, y \in \mathbb{R}} y$ subject to $-1 \leq u_i \leq 1$ and $d^T u \leq y$ for all $d \in G$, a standard linear problem. Finally, a heuristic is used to find a feasible point by performing a discretized line search in the direction u starting from an arbitrary point in the current decision variable domain. We now extend this approach to SICs, and its convergence in the context of SICs will be proved in Section 3.5.

In [41], the vectors in G are chosen to be the gradients of the potentially active constraints, i.e., whose interval evaluations contains zero, evaluated at

the midpoint of the decision variable domain. In the context of SICs, each box of the parameter domain paving can be considered as a potentially active constraint. Furthermore, any point inside the decision variable domain can be chosen instead of the midpoint. Therefore we define

$$G = \{\nabla_x \tilde{g}(\hat{x}, \hat{y}_{[y]}) : [y] \in \mathcal{Y}_{[x]}\}, \tag{37}$$

where $\hat{x}$ is an arbitrary point in $[x]$ and $\hat{y}_{[y]}$ is an arbitrary point in $[y]$. The direction u is then computed as in [41] by solving (36). Starting from an arbitrary base point $\tilde{x} \in [x]$ (in practice we choose $\tilde{x} = x_{\text{relax}}$ which is foreseen to become close to the minimizer), the directional search consists in finding $\bar{t} \geq 0$ such that $g(\tilde{x} + \bar{t}u) < 0$. Unlike [41] where a discretization process in the direction u is used, we propose here to use an approximate linear model: We define $\bar{t}$ using an approximate directional derivative α of $g(\tilde{x} + tu)$ at $t = 0$ defined by

$$\alpha = \max_{d \in G} d^T u, \tag{38}$$

where G is the set of vectors defined in (37). We expect $\alpha < 0$, otherwise the line search is foreseen not to discover any feasible point. The SIC is then evaluated at the base point to obtain the value $g(\tilde{x})$, as explained in Section 3.1, and for an arbitrary $\sigma \in (0, 1)$ we define $\bar{t}$ as the solution of the affine equation $g(\tilde{x}) + \alpha\sigma t = 0$, that is

$$\bar{t} = -\frac{g(\tilde{x})}{\alpha\sigma}. \tag{39}$$

The shifting factor σ is fixed during the search to ensure convergence (in fact it may vary but should stay bounded away from zero), a reasonable value is $\sigma = 0.9$. If $g(\tilde{x}) > 0$ then $\bar{t}$ is positive and $g(\tilde{x} + \bar{t}u)$ is evaluated for checking feasibility. In the other case, the base point is feasible and no directional search is performed.

Remark 14. *When several constraints $g_i(x) \leq 0$ are involved, we consider the equivalent constraint $g(x) := \max_i g_i(x) \leq 0$, which is also a SIC if one of the $g_i(x) \leq 0$ is a SIC. In practice, each constraint parameter pavings are independently used to build G in* (37)*, and the largest $\bar{t}$ is used, or equivalently, the maximum of all $g_i(\tilde{x})$.*

3.4. Branching decision and parameter domains

The parameter domain paving $\mathcal{Y}_{[x]}$ is used in all operations related to SICs. It has to enclose accurately $y^*_{[x]}$ and therefore has to be refined when the decision variable domain $[x]$ is updated. This happens in two steps of the branch-and-bound algorithm: First, when $[x]$ is contracted to a new box $[x']$ the paving $\mathcal{Y}_{[x]}$ simply has to be refined. Second, when $[x]$ is bisected into two new boxes $[x']$ and $[x'']$, the paving $\mathcal{Y}_{[x]}$ has to be copied to two new pavings $\mathcal{Y}_{[x']}$ and $\mathcal{Y}_{[x'']}$, which are refined with respect to their respective new domains.

Refining a paving with respect to an updated domain is necessary in order to enforce it to converge to $y^*_{[x]}$ when the width of the decision domain $[x]$ converges to 0. To this end, boxes of the paving have to be split and useless boxes have to be removed. This is done in four distinct steps:

1. To enforce the convergence, the parameter box that has the largest width is split at the midpoint of a longest edge.
2. Each box of the paving is tested for potential splitting. The aim is to split boxes only when necessary in order to prevent the paving from containing too many boxes. We have tested several criteria for deciding whether a box should be split or not, and the most robust is the following: Given a parameter box $[y]$, we split at the midpoint of a longest edge and obtain two boxes $[y']$ and $[y'']$. We then perform the three interval evaluations $[z] = [\tilde{g}]([x],[y])$, $[z'] = [\tilde{g}]([x],[y'])$ and $[z''] = [\tilde{g}]([x],[y''])$. Finally, we replace $[y]$ in the paving by $[y']$ and $[y'']$ if one of the ratios $\frac{\|\operatorname{wid}[z']\|}{\|\operatorname{wid}[z]\|}$ or $\frac{\|\operatorname{wid}[z'']\|}{\|\operatorname{wid}[z]\|}$ is less than a given threshold, in practice 0.8. Crudely speaking, the box $[y]$ is split only if this improves significantly the interval evaluation of the constraint.
3. A parameter box $[y]$ such that $\sup[\tilde{g}]([x],[y]) < \underline{\mathcal{Y}}_{[x]}$ cannot contain any maximizer of $\tilde{g}(x,y)$ for any $x \in [x]$, and can therefore be rejected.
4. Five rules[1] are then applied to each box $[y]$ of the paving $\mathcal{Y}_{[x]}$ to reduce or reject it, keeping the inclusion $\cup\mathcal{Y}_{[x]} \supseteq y^*_{[x]}$ valid:
   - If $[\nabla_{y_i}\tilde{g}]([x],[y]) < 0$ and $\underline{y}_i^0 < \underline{y}_i$ then $[y]$ can be rejected.
   - If $[\nabla_{y_i}\tilde{g}]([x],[y]) < 0$ and $\underline{y}_i^0 = \underline{y}_i$ then $[y_i]$ can be replaced by the degenerated interval $\underline{y}_i$.
   - If $[\nabla_{y_i}\tilde{g}]([x],[y]) > 0$ and $\overline{y}_i < \overline{y}_i^0$ then $[y]$ can be rejected.
   - If $[\nabla_{y_i}\tilde{g}]([x],[y]) > 0$ and $\overline{y}_i = \overline{y}_i^0$ then $[y_i]$ can be replaced by the degenerated interval $\overline{y}_i$.
   - If $[y] \subseteq \operatorname{int}[y^0]$ then we apply an interval Newton operator to the system $\nabla_y g([x],y) = 0$.

This refinement process works like a simplified branch-and-bound algorithm that maximizes the function $\tilde{g}([x],y)$ with respect to y. Encapsulating this simplified maximization process as a constraint refinement during the search allows both including SICs transparently within the branch-and-bound algorithm, hence taking benefit of its overall efficiency, and using meaningful parameter values in the construction of relaxations and restrictions. This section is ended by proving that this refinement process is convergent.

Proposition 15. *The refinement process consisting of the refinement steps 1 and 3 is convergent provided that the interval extension $[\tilde{g}]$ is convergent.*

Proof. Consider an exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$ generated by the algorithm. Denote by x_∞ the limit of this sequence, i.e., $\cap_{k=0}^{\infty}[x_k] = \{x_\infty\}$. Each new iterate $[x_k]$ has gone through at least one parameter paving refinement.

[1] The first four tests are similar to standard monotonicity tests [40, 33], which were already used in the context of SICs in [30] in a simpler form.

As a consequence, the usual split strategy used at step 1 enforces $\operatorname{wid} \mathcal{Y}_{[x_k]}$ to converge to zero, hence (12) holds.

We now prove (13). Choose $y_\infty \in y^*_{x_\infty}$ and consider an exhaustive sequence of boxes $([y_k])_{k\in\mathbb{N}}$ such that $[y_k] \in \mathcal{Y}_{[x_k]}$ and $\cap[y_k] = \{y_\infty\}$ (such a sequence exists because $y_\infty \in \cup\mathcal{Y}_{[x_k]}$ by (9) and updating a paving consists only in splitting or contracting boxes). Now assume, by way of contradiction, that (13) is false, that is, there exists $\overline{\epsilon} > 0$ such that $\cup\mathcal{Y}_{[x_k]}\backslash(y^*_{x_\infty} + \overline{\epsilon}B) \neq \emptyset$ for all $k \in \mathbb{N}$. Since $\cup\mathcal{Y}_{[x_k]}$ are nested, nonempty and compact by construction, so are $\cup\mathcal{Y}_{[x_k]}\backslash \operatorname{int}(y^*_{x_\infty} + \frac{\overline{\epsilon}}{2})$, where int notes the interior of a set (the difference of a closed set by an open set is closed). Therefore, by Cantor's intersection theorem we have

$$\bigcap_{k=0}^{\infty} \cup\mathcal{Y}_{[x_k]}\backslash \operatorname{int}(y^*_{x_\infty} + \frac{\overline{\epsilon}}{2}) \neq \emptyset. \tag{40}$$

Pick $\tilde{y}_\infty$ in this set and an exhaustive sequence of boxes $([\tilde{y}_k])_{k\in\mathbb{N}}$ such that $[\tilde{y}_k] \in \mathcal{Y}_{[x_k]}$ and $\cap_{k=0}^{\infty}[\tilde{y}_k] = \{\tilde{y}_\infty\}$. Since $\tilde{y}_\infty \notin y^*_{x_\infty}$ we have $\tilde{g}(x_\infty, y_\infty) > \tilde{g}(x_\infty, \tilde{y}_\infty)$. The interval extension of $\tilde{g}$ being convergent, there exists $K \in \mathbb{N}$ such that $\inf[\tilde{g}]([x_K], [y_K]) > \sup[\tilde{g}]([x_K], [\tilde{y}_K])$ and Step 3 rejects the parameter box $[\tilde{y}_K]$. Therefore, the exhaustive sequence of boxes $([\tilde{y}_k])_{k\in\mathbb{N}}$ cannot exist, a contradiction. □

The refinement steps 1 and 3 are sufficient for enforcing the refinement process to be convergent. However, the remaining refinement steps are critical for the efficiency of the algorithm.

Remark 16. *Since only active constraint are to be considered in the branch-and-bound process, we replace the condition* $\sup[\tilde{g}]([x], [y]) < \underline{\mathcal{Y}}_{[x]}$ *in Step 3 by* $\sup[\tilde{g}]([x], [y]) < 0$. *This does not impact the correctness of the algorithm but slightly reduces the size of the parameter paving and therefore the overall computational timings.*

3.5. Convergence of the algorithm

The convergence of the algorithm is proved for one SIC $g(x) \leq 0$, but the proof holds more generally if several constraints $g_i(x) \leq 0$ are involved by considering one constraint $g(x) := \max_i g_i(x) \leq 0$, as in Section 3.3.3. This constraint is a SIC provided that one of the constraints $g_i(x) \leq 0$ is a SIC, and satisfies all the properties of SICs. In practice, one may also consider extended parameters of the form $(i, y) \in \bigcup_{k=1}^{m}\{k\} \times \mathbb{R}^{m_k}$ with corresponding parameter domains $Y = \bigcup_{k=1}^{m}\{k\} \times [y^k]$, integers denoting indices of constraints. The universal quantification over this extended parameter domain enforces the universal quantifications over all SIC independently and the parameter pavings to converge independently (non-SIC constraints can be included using singleton parameter domains). Obviously, the convergence of the branch-and-bound algorithm depends on strategy for the choice of the next decision variable domain to be proceeded in the search tree and the bissection strategy. We use the standard strategies, which are also used in [41]: The selected decision variable domain

that has the smallest objective lower bound and bisect the decision variable domain at the midpoint of its largest edge.

3.5.1. Convergence of the lower bounding process

The constraint propagation on the constraints $g(x) \leq 0$ and $f(x) \leq f^*$, where f^* is the current upper bound, enforces a decision variable domain $[x]$ to be rejected if $[g]([x]) > 0$ or $[f]([x]) > f^*$. This corresponds to M-dependent lower bounding procedure used to fathoming in [41]. This procedure is convergent provided that both interval extensions $[g]$ and $[f]$ are convergent, and monotone because the interval evaluation is monotone[2]. As a consequence, Proposition 4.2 of [41] applies and shows that the branch-and-bound lower bound process is convergent.

3.5.2. Convergence of the upper bounding process

The convergence of the upper bound process proposed in [41] has to be adapted, mainly because in the context of SICs, active constraints cannot be identified exactly (they are real values of parameters and there may be infinitely many active constraints). The convergence of the upper bounding process is proved under a typical Mangasarian-Fromovitz constraint qualification (MFCQ): We require that every global minimizer x_* has a direction u_* where g has a strictly negative directional derivative.

We suppose that the feasible set is nonempty. We make the generic assumption that the problem has a unique global minimizer[3], denoted by x_*, and that this global minimizer satisfies the MFCQ. We define

$$\alpha_* := \max_{d \in G_*} d^T u_*, \tag{41}$$

with $G_* = \{\nabla_x g(x_*, y) : y \in y^*(x_*)\}$, so that $\partial g(x_*) = \operatorname{conv} G_*$ and α_* is the directional derivative of g in the direction u_*. The MFCQ then reads $\alpha_* < 0$.

In the rest of the section, we consider an exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$ produced by the branch-and-bound algorithm that converges to a global minimizer x_* (such a sequence exists because the lower bounding process is convergent, see the proof of Proposition 4.2 in [41]). For clarity, we define $\mathcal{Y}_k := \mathcal{Y}_{[x_k]}$. We furthermore define G_k to be the set of vectors (37) for the box $[x_k]$, i.e.,

$$G_k = \{\nabla_x \tilde{g}(\hat{x}_k, \hat{y}_{[y]}) : [y] \in \mathcal{Y}_k\}, \tag{42}$$

where $\hat{y}_{[y]}$ is an arbitrary point in $[y]$ and $\hat{x}_k$ is an arbitrary point in $[x_k]$. Let $x_k \in [x_k]$ be the base-point of the directional search. As the exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$ converges to x_*, so do $(x_k)_{k\in\mathbb{N}}$ and $(\hat{x}_k)_{k\in\mathbb{N}}$.

[2]Monotonicity is actually not required in the proof given in [41], as the branch-and-bound algorithm enforces monotonicity of the lower bounds of subdomains.

[3]In the non generic case where there are several global minimizers, one may either assume the MFCQ for each of them, or conduct a finer analysis to prove the branch-and-bound algorithm actually accumulates to each of them, one of them being MFCQ-qualified.

We need the following three lemmas. Lemma 17 shows that the gradients in G_k converges to the gradient in G_* for the Hausdorff distance provided that the parameter paving refinement process is convergent.

Lemma 17. *Suppose that the parameter paving refinement process is convergent. Then $G_k \subseteq G_* + \epsilon_k B$ and $G_* \subseteq G_k + \epsilon_k B$ with $\lim_{k\to\infty} \epsilon_k = 0$.*

Proof. Let $\hat{\mathcal{Y}}_k$ be the set of parameters used to compute G_k, i.e.,

$$G_k = \nabla_x \tilde{g}(\{\hat{x}_k\} \times \hat{\mathcal{Y}}_k), \tag{43}$$

where the standard evaluation of a function over a set is used. Then we have $\hat{\mathcal{Y}}_k \subseteq \cup\mathcal{Y}_k \subseteq y^*(x_*) + \epsilon'_k B$, the second inclusion holding because the parameter paving refinement process is supposed convergent. Let $w_k := \operatorname{wid} \mathcal{Y}_{[x_k]}$, which converges to zero since the refinement process is supposed convergent. Now we also have $y^*(x_*) \subseteq \cup\mathcal{Y}_k \subseteq \hat{\mathcal{Y}}_k + w_k B$ (because $[y] \subseteq \hat{y} + \operatorname{wid}[y] B$ for an arbitrary $\hat{y} \in [y]$). As a consequence,

$$d_H(y^*(x_*), \hat{\mathcal{Y}}_k) \leq \max\{\epsilon'_k, w_k\}, \tag{44}$$

which converges to zero. Since $d_k := \|x_* - \hat{x}_k\|$ converges to zero, so

$$d_H(\{x_*\} \times y^*(x_*), \{\hat{x}_k\} \times \hat{\mathcal{Y}}_k) \leq \max\{\epsilon'_k, w_k, d_k\} \tag{45}$$

also converges to zero. Finally, $\nabla_x \tilde{g}$ is continuous in the compact set $[x^0] \times [y^0]$, it is therefore uniformly continuous, and its set extension is (uniformly) continuous for the Hausdorff distance. Therefore

$$d_H(\nabla_x \tilde{g}(\{x_*\} \times y^*(x_*)), \nabla_x \tilde{g}(\{\hat{x}_k\} \times \hat{\mathcal{Y}}_k)) \tag{46}$$

also converges to zero. Since (46) is exactly $d_H(G_*, G_k)$, this proves the statement. □

Lemma 18 provides some convergence property for an approximate linear model of a Lipschitz function.

Lemma 18. *Let $(u_k)_{k\in\mathbb{N}}$ with $\|u_k\| \leq 1$, $(x_k)_{k\in\mathbb{N}}$ converges to x_* and $(t_k)_{k\in\mathbb{N}}$ converges to 0. Suppose that $G_* \subseteq G_k + \epsilon_k B$ with $\lim \epsilon_k = 0$. Define $g_k(t) = g(x_k + t u_k)$ and $\alpha_k = \max_{d\in G_k} u_k^T d$. Then $g_k(t_k) \leq g_k(0) + \alpha_k t_k + \epsilon'_k t_k$ with $\lim \epsilon'_k = 0$.*

Proof. By the mean-value theorem for Lipschitz function [16], $g_k(t_k) \in g_k(0) + t_k u_k^T \partial g(x_k + s_k u_k)$ for some $s_k \in [0, t_k]$. Considering the worst case we obtain

$$g_k(t_k) - g_k(0) \leq t_k \max_{d\in\partial g(x_k + s_k u_k)} u_k^T d. \tag{47}$$

Since the generalized gradient is set-valued upper hemicontinuous [16] and $x_k + s_k u_k$ converges to x_*, we have $\partial g(x_k + s_k u_k) \subseteq \partial g(x_*) + \epsilon''_k B$ with $\lim \epsilon''_k = 0$. We obtain the following upper bounds:

$$g_k(t_k) - g_k(0) \leq t_k \max_{d\in\partial g(x_*) + \epsilon''_k B} u_k^T d \leq t_k \max_{d\in\partial g(x_*)} u_k^T d + t_k n\, \epsilon''_k, \tag{48}$$

where $u_k^T d \leq n$ because both have maximal norm less than 1. Maximizing a linear function $v \to u_k^T d$ over $\partial g(x_*) = \operatorname{conv} G_*$ is equivalent to maximizing it over G_*. Using furthermore $G_* \subseteq G_k + \epsilon_k B$ we obtain

$$g_k(t_k) - g_k(0) \leq t_k \max_{d \in G_k + \epsilon_k B} u_k^T d + t_k \epsilon_k'' \leq t_k \max_{d \in G_k} u_k^T d + t_k n \left(\epsilon_k + \epsilon_k''\right). \quad (49)$$

This ends the proof with $\epsilon_k' := n(\epsilon_k + \epsilon_k'')$, which converges to 0. □

Lemma 19 provides some limit property on optimization problems with Lipschitz cost function and outwardly convergent feasible sets.

Lemma 19. *Let $h : E \to \mathbb{R}$ be Lipschitz, $P_k \subseteq E$ and $P_* \subseteq E$ such that $P_k \subseteq E \cap \left(P_* + \epsilon_k B\right)$ and $\lim_{k\to\infty} \epsilon_k = 0$. Then*

$$\limsup_{k\to\infty} \max_{u \in P_k} h(u) \leq \max_{u \in P_*} h(u). \quad (50)$$

Proof. We have

$$\max_{u \in P_k} h(u) \leq \max_{u \in E \cap \left(P_* + \epsilon_k B\right)} h(u) = \max_{\substack{u \in P_* \\ \|\delta\| \leq \epsilon_k \\ u+\delta \in E}} h(u+\delta) \leq \max_{u \in P_*} h(u) + L_h \epsilon_k, \quad (51)$$

the last inequality holding because h is L_h Lipschitz. The statement follows taking the superior limits of the two sequences. □

The following theorem shows the convergence of the upper bounding process defined Section 3.3.3.

Theorem 20. *Suppose that the parameter paving refinement process is convergent. Suppose furthermore that the feasible set is nonempty, so that the exhaustive sequence of boxes $([x_k])_{k\in\mathbb{N}}$ that converges to the MFCQ-qualified unique global minimizer x_* is proved to exist. Pick an arbitrary directional search base point $\tilde{x}_k \in [x_k]$ and suppose that for all $k \in \mathbb{N}$ the inequality $g(\tilde{x}_k) > 0$ holds[4]. Let $\sigma \in (0,1)$ and define $m_k(u) := \max_{d \in G_k} d^T u$, $\alpha_k := \min_{u \in B} m_k(u)$ and $u_k = \operatorname{argmin}_{u \in B} m_k(u)$, so $\alpha_k = m_k(u_k)$ and α_k and u_k correspond to (38) and (36) respectively. Then there exists $K \geq 0$ such that for all $k \geq K$ we have both $2\alpha_k \leq \alpha_*$ and $g(\tilde{x}_k + \bar{t}_k u_k) < 0$ with $\bar{t}_k$ defined as in (39), i.e., $\bar{t}_k := -\frac{g(\tilde{x}_k)}{\alpha_k \sigma}$, which is well defined since $\alpha_* < 0$. As a consequence, $\tilde{x}_k + \bar{t}_k u_k$ is feasible and converges to x_*.*

Proof. Define $\alpha_{k*} := m_k(u_*)$. Since α_k it minimizes m_k for $u \in B$ we have $\alpha_k \leq \alpha_{k*}$. The function $d \to d^T u_*$ is Lipschitz and, by Lemma 17, we have $G_k \subseteq G_* + \epsilon_k B$ with $\lim \epsilon_k = 0$. Therefore Lemma 19 proves

$$\limsup_{k\to\infty} \max_{d \in G_k} d^T u_* \leq \max_{d \in G_*} d^T u_*, \quad (52)$$

[4]The other case $g(\tilde{x}_k) \leq 0$ happens, e.g., when no constraint is active at the globale minimizer. Defining $\bar{t}_k := \max\{0, -\frac{g(\tilde{x}_k)}{\alpha_k \sigma}\}$ instead of $\bar{t}_k := -\frac{g(\tilde{x}_k)}{\alpha_k \sigma}$ in the statement allows finding a feasible point in this case too. The convergence proof skips this detail for clarity.

that is, $\limsup_{k\to\infty} \alpha_{k*} \leq \alpha_*$. Together with the previously proved inequality $\alpha_k \leq \alpha_{k*}$ we obtain $\limsup_{k\to\infty} \alpha_k \leq \alpha_*$. Since $\alpha_* < 0$, there exists K' such that $\alpha_k \leq \frac{\alpha_*}{2}$ holds for $k \geq K'$. In particular, α_k is bounded away from zero for $k \geq K'$.

Since α_k is less than and bounded away from 0 and $g(\tilde{x}_k)$ converges to zero, $\bar{t}_k$ is the quotient of a numerator that converges to zero and a denominator that is bounded away from zero. Therefore $\bar{t}_k$ converges to 0. Furthermore by Lemma 17 we have $G_* \subseteq G_k + \epsilon_k B$ with $\lim_{k\to\infty} \epsilon_k = 0$. Therefore, all hypothesis of Lemma 18 are satisfied so $g_k(\bar{t}_k) \leq g_k(0) + \alpha_k \bar{t}_k + \epsilon'_k \bar{t}_k$ with $\lim \epsilon'_k = 0$. Using the expression of $\bar{t}_k$ we obtain

$$g_k(\bar{t}_k) \leq g_k(0) - \alpha_k \frac{g(\tilde{x}_k)}{\alpha_k \sigma} + \epsilon'_k \frac{g(\tilde{x}_k)}{|\alpha_k|\sigma} = g_k(0)\big(1 - \frac{1}{\sigma} + \frac{\epsilon'_k}{|\alpha_k|\sigma}\big). \tag{53}$$

Finally, since $g_k(0) = g(\tilde{x}_k) > 0$ by hypothesis, and $\frac{1}{\sigma} > 1$, $|\alpha_k| \geq \frac{|\alpha_*|}{2} > 0$ and $\lim \epsilon'_k = 0$, there exists $K \geq K'$ such that for all $k \geq K$ we have $g_k(\bar{t}_k) < 0$. Finally, since $\tilde{x}_k$ converges to x_* and $\bar{t}_k$ converges to zero and $\|u_k\| \leq 1$, we have $\tilde{x}_k + \bar{t}_k u_k$ converges to x_*. □

4. Experiments

In this section, we present experimental evidence of the interest of the algorithm we have introduced in the previous section.

Our implementation of this algorithm is written in C++ using the interval solving library Ibex [14], based on the interval arithmetic library GAOL [32]. The solver for LP subproblems is SoPlex 3.1.1 [61]. The program is compiled with G++ 5.4.0 with the flags `-std=c++11 -O3 -DNDEBUG`. The tests are run on an Intel Xeon E3-1280 v6 @ 3.90GHz running Ubuntu 16.04.3. The source code is available and can be installed as a plugin of the Ibex library called `sip`. Problem models are available in the `benchs` subdirectory of the plugin.

In the bisection process of the parameter paving, parameter boxes are not bisected further if their diameter is smaller than 10^{-10}. Variables are bisected as long as it is possible.

We first propose in Section 4.1 an experiment inspired from [29] focusing on the *clustering effect* around optima of SIPs. Next in Section 4.2.2, we propose a general comparison to the state-of-the-art alternative from [19], which is to our knowledge the best general SIP solving algorithm to date[5]. Finally, in Section 4.4 we present the resolution of a recently proposed difficult SIP model for telescope design.

4.1. Clustering effect for SIPs

When solving NLPs using an interval-based branch-and-bound method, it is usual to obtain an accumulation of small boxes around the optima that cannot

[5] [19] presents comparison to [46], and [46] presents comparisons to [23, 11].

be rejected efficiently because they are both almost feasible and almost optimal. The number of such boxes typically grows with the prescribed precision, making the resolution inefficient in case this cluster effect is not correctly handled. See [51, 60, 38] for more details on the cluster effect. The following experiment demonstrates simultaneously that there exists a clustering effect around the optima of SIPs similar to the one observed for NLPs, and that our port of the rejection test based on first-order optimality conditions with generalized gradients (see Section 3.1) and our linear relaxation-based lower-bounding technique (see Section 3.2) handle this clustering effect.

To this end, we consider the following scalable SIP:

$$\min f(x) := \sum_{i=1}^{i=n} x_i \tag{54}$$

$$s.t. \quad g(x) := \max_{y\in[-1,1]} \sum_{i=1}^{i=n} x_i^2 + y\,(x_1+1) - n \leq 0. \tag{55}$$

This is the same problem as in Section 5.2 of [29], where the original nonsmooth constraints $g(x) := \sum_{i=1}^{i=n} x_i^2 + |x_1+1| - n \leq 0$ is now replaced by its equivalent semi-infinite expression (55).

The clustering effect when solving the initial non-smooth NLP problem is studied in [29] by varying two solving parameters: The dimension of the problem n, and the solving precision ϵ_x. An asymptotic formula of the number of boxes of size ϵ_x generated by the algorithm in the configuration where it uses only constraint propagation without linear relaxations as lower bounding process is given in [29]:

$$O\big(\epsilon_x^{-\frac{n-2}{2}}\big). \tag{56}$$

This number becomes extremely high for small values of ϵ_x even for relatively small vales of n. It was confirmed in [29] that the solving time is approximately proportional to this number of boxes. We study the clustering effect when solving our SIP adaptation (55) with the same parameters. The results are depicted in Figure 4. The upper-left diagram in Figure 4 shows that the solving time follows quite accurately the cluster effect model (56), which are displayed in gray-dashed lines, demonstrating the existence and impact of the clustering effect for SIPs. In contrast, the results in the other diagrams show the positive impact of the techniques we have introduced in our branch-and-bound in order to address this effect, and their complementarity. These results are in line with those in [29].

4.2. Comparative benchmarking

We now propose a comparison of the performance of our branch-and-bound method with the Hybrid, Blankenship-based, approach in [19]. The GAMS implementation of this method is that provided by the authors as supplementary

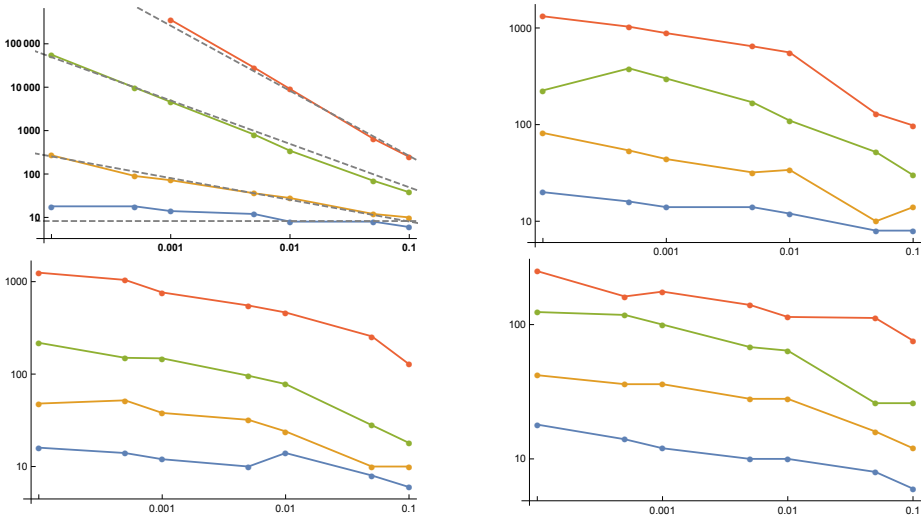

Figure 4: Solving time in seconds (in ordinate) depending on the precision (in abscissa) and the dimension (n=2 in blue, n=3 in yellow, n=4 in green, n=5 in red). Upper-left: branch-and-bound without any anti-clustering technique. Upper-right: branch-and-bound with the lower-bounding technique described in Section 3.2. Lower-left: branch-and-bound with the first-order rejection technique described in Section 3.1. Lower-right: branch-and-bound with both techniques.

material with their paper[6]. We have measured its computation times on our computer, with the parameters specified in [19], however with a more recent version of GAMS 28.2.0 [25] and BARON 19.7.13 [54]. Settings of ibexopt-sip are the same as [19]: both an absolute and relative termination criterion set to 10^{-3}.

4.2.1. Benchmark

All the problems in our benchmark are from the Mitsos SIP test set [45]. Problems 2, 5, 6, 7, 8, 9, H, N, S, 4_3, 4_6 are simple problems with only one quantified constraint, at most 6 variables and at most 2 quantified parameters. The objective function is linear for 8, 9, H, N, 4_3, 4_6 and polynomial for 2, 6 and 7. The SIC is linear in the variables for problems 7, 8, 9, 4_3, 4_6 implying the gradient of the constraint does not depend on the variables, dramatically increasing the accuracy of linearizations.

Problems Dn_1_n_2_n_3 are derived from a design centering problem described in [45], where n_1, n_2, n_3 are three integers: n_1 describes the containing set, n_2 the type of objective function (linear or quadratic) and n_3 the number of circles used in the problem. A graphic representation of the problems and their solutions can be found in [46]. The number of quantified constraints is

[6]Electronic supplementary material on `https://link.springer.com/article/10.1007/s10898-016-0476-7`

$n_1 n_3$, with one quantified parameter each. There are also several non quantified constraints. The variables represent the coordinates and radius of the circles, therefore there are 3 decision variables if $n_3 = 1$ and 6 if $n_3 = 2$.

4.2.2. Results

Table 2 shows computation times for the compared algorithms. Column Problem is the problem name as presented in the previous section, column Hybrid GAMS is the computation time of the Hybrid algorithm from [19], column Hybrid Ibex is the computation time of the Hybrid algorithm using an IBEX implementation, and column ibexopt-sip is the computation time of our algorithm implemented with IBEX. Hybrid GAMS uses the same parameters as in [19], with $r^g = 1.2$. Hybrid Ibex uses IbexOpt as the sub-problems solver, with $\varepsilon^{g,0} = 0.1$, $r^g = r^{LLP} = 1.5$, $\varepsilon^{UBD,0} = \varepsilon^{LBD,0} = \varepsilon^{RES,0} = \varepsilon^{LLP,0} = 10^{-4}$ and $l_{\max} = 20$. The best solving time for each problem is emphasized (boldface).

Problem	Hybrid GAMS	Hybrid Ibex	ibexopt-sip
2	**0.01**	0.10	**0.10**
4_3	0.10	0.05	**0.02**
4_6	0.78	0.43	**0.06**
5	0.08	0.11	**0.03**
6	1.10	0.26	**0.02**
7	**0.05**	0.17	0.09
8	0.95	1.41	**0.35**
9	0.20	$\infty^{(\star)}$	**0.07**
H	0.53	0.27	**0.01**
N	0.12	0.04	**0.01**
S	4.28	0.60	**0.09**
D101	0.16	0.13	**0.03**
D102	22.48	21.60	**1.26**
D111	0.14	0.15	**0.02**
D112	14.40	3.01	**0.24**
D201	0.74	0.42	**0.09**
D202	13.43	10.21	**2.28**
D211	1.37	0.48	**0.10**
D212	36.29	8.68	**1.41**
DP	0.36	0.71	**0.01**
Total	97.62	49	6.26

Table 2: Comparison between the hybrid algorithms and Ibexopt-sip. Timings are in seconds. 0.01s means that the solving time is less than 0.01s.
$^{(\star)}$ Problem 9 cannot be solved within the time limit with Hybrid Ibex unless the *anticipated upper bounding* feature of IbexOpt is deactivated (this feature accelerates the branch-and-bound algorithm by limiting the cluster effect but prevents the algorithm to find new parameters values for Problem 9). In that case, the solving time is 0.017s.

We can see that on most instances, our algorithm fares better than Hybrid. Surprizingly, problem 9 is solved quite efficiently by our method while its optimum is associated to a continuous set of corresponding parameters values, instead of a discrete set for the other problems. This theoretically reduces the efficiency of our parameter filtering strategy, but this difficulty is in fact overcome by the linear restrictions and the Blankenship heuristic in our algorithm.

4.3. Detailed analysis

In this section, numerical experiments are performed to analyze the influence of the different features exposed in this article for lower bounding and upper bounding. For each identified feature, the number of problems solved in less than x seconds is measured. Results are shown in Figure 5 and Figure 6. As deactivating specific features can compromise the convergence of the algorithm on some problems, a timeout has been set to 100 seconds.

4.3.1. Pruning and lower bounding

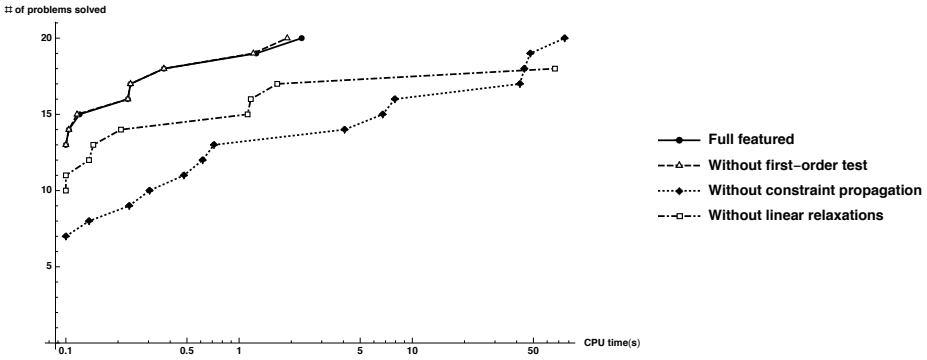

Figure 5: Number of problems from the Mitsos Sip test set solve in less than x seconds, with a timeout set to 100s, when deactivating specified lower-bounding feature.

As shown in Section 3.2, our algorithm uses three separate approaches to pruning and lower bounding : linear relaxations of constraints, constraint programming and a first-order optimality test. As can be seen in Figure 5, deactivating constraint propagation or linear relaxations leads to severe performance losses. For example, problems 8 and 4_6 cannot be solved in less than 100s without relaxations and D2_0_2 solving time goes from 2.28s to 48s when deactivating constraint propagation. More generally, D$n_1n_2n_3$ greatly benefit from constraint propagation.

On the contrary, the first-order test has almost no effect on performance: most problems do not show any difference, except D2_0_2 which shows a 17% degradation in computation time without the test. But as discussed in Section 4.1, it is very useful on problems prone to clustering, thus its inclusion in the final algorithm. This also highlights a deficiency of the Mitsos SIP test set, which does not contain problems prone to clustering.

4.3.2. Upper bounding

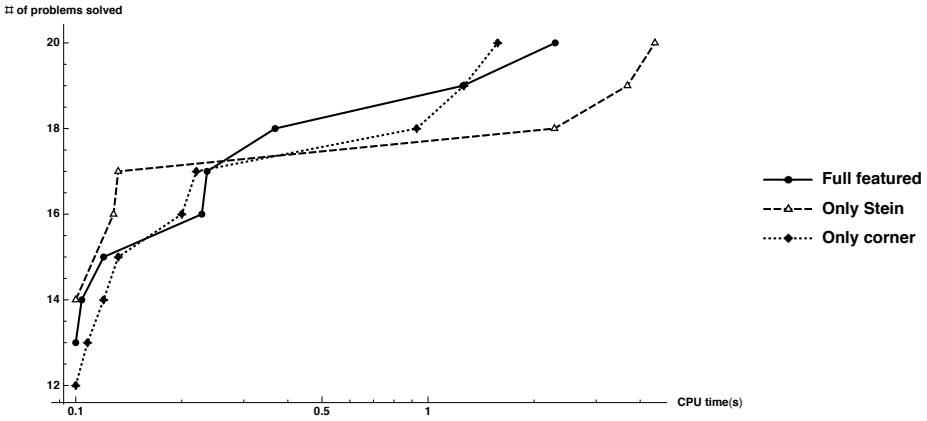

Figure 6: Number of problems from the Mitsos Sip test set solve in less than x seconds, with a timeout set to 100s, when deactivating specified upper-bounding feature.

Feasible points are all found by the line search procedure described in Section 3.3.3. There are two main approaches to find a search direction: one is the generalization to SIPs of Stein's directional search approach (Section 3.3.3), and the other is the use of corner linear restrictions (Section 3.3.2). Figure 6 shows that corner restrictions can greatly increase search speed. On the whole problem set, the corner strategy in enough to solve the bench, however the Stein strategy is necessary to enforce the convergence of the algorithm and only slightly degrades performance on this bench, especially on D202, which goes from 2.30s to 1.58s when removing Stein. Removing the corner strategy leads to severe performance degradation, especially on D212, which goes from 0.23s to 2.28s, but also on Problem 8, D102, D202, and D212. Overall the performance loss when deactivating the corner strategy is 120%. It is interesting to note that for D202 the combination of both strategies is the best strategy.

4.4. Telescope design problem

The following SIP appears in [4]:

$$\text{Minimize } t, \tag{57}$$

$$\text{subject to } -t \leq \frac{J_1(\pi\alpha)}{\alpha} \sum_{k=1}^{m} a_k \cos\left(\frac{2\pi}{d} u_k \alpha\right) \leq t, \ \forall \alpha \in [\alpha_{\min}, \alpha_{\max}], \tag{58}$$

$$u_{k+1} - u_k \geq d, \text{ for } k = 1, ..., m-1, \tag{59}$$

$$\sum_{k=1}^{m} a_k = \frac{1}{2}, \tag{60}$$

$$a_k \geq 0, \text{ for } k = 1, ..., m, \tag{61}$$

where J_1 is the first-order Bessel function of the first kind. In theory, we need to implement this Bessel function for intervals. On a bounded domain, it is however possible to precompute Taylor coefficients in order to obtain a sufficient computational precision. We used the following approximation:

$$\frac{J_1(\pi\alpha)}{\alpha} = \sum_{p=0}^{9}(-1)^p \left(\frac{\pi}{2}\right)^{2p+1} \frac{1}{p!(p+1)!}x^{2p} + O(x^{2p}) \tag{62}$$

which is the power series of order 18. The maximum error on the evaluation of $\frac{J_1(\pi\alpha)}{\alpha}$ and its derivative on the interval $[0.25, 0.75]$ using this approximation is less than 10^{-12}. Another solution would be to use the Arb interval library [37] which provides interval arithmetic for Bessel functions but linking [37] with IBEX is problematic. From [4], $\alpha_{\min} = 0.25$ and $\alpha_{\max} = 0.75$, $m = 4$ and $d = 1$, which gives 9 decision variables and two SICs with a scalar parameter. We have $\alpha \in [0.25, 0.75]$, and we chose $u \in [0, 5]^m$ and $a \in [0, 1]^m$. This problem was solved in 265 seconds with ibexopt-sip. Results are shown in Table 3. The minimizer value is similar and the optimum is improved by 15% with respect to [4]. We cannot use GAMS with Baron to solve this problem, because Baron does not solve problems with trigonometric functions. Hybrid Ibex could not solve this problem or even find a feasible point in 1 hour.

Variable	[4]	ibexopt-sip
$t^\star$	0.00289906377301	0.00246502446622
$\bar{a}^\star$	$(1.0000, 0.7277, 0.36344, 0.1044)$	$(1.0000, 0.7269, 0.3634, 0.1036)$
$u^\star$	$(0.5054, 1.5167, 2.5299, 3.5510)$	$(0.5044, 1.5134, 2.5241, 3.5406)$

Table 3: Numerical solutions to the telescope design problem presented in [4], with $\bar{a}^\star = a^\star/a_1$.

5. Conclusion

A new branch-and-bound algorithm for semi-infinite problems has been proposed, extending methods already used in NLP solvers to semi-infinite constraints (SICs). In particular, interval evaluation, first-order rejection test based on generalized gradients interval evaluation, numerical constraint propagation and linear relaxations and restrictions have been extended to SICs. These bounding techniques rely on a paving that approximates the maximal parameter values of each SIC, refined during the search. Experiments have been performed on standard benchmarks from the literature: First the proposed bounding techniques handle efficiently the cluster effect: Linear relaxations and the first order rejection test are actually complementary. Second, our implementation of our branch-and-bound algorithms outperforms the currently best algorithm dedicated to general SIPs proposed by Djelassi and Mitsos [19], implemented by both its authors using GAMS–Baron and ourselves using Ibex.

One advantage of following the standard branch-and-bound algorithm is modularity: Other bounding techniques like [47] may also be included and their

efficiency within a branch-and-bound algorithm can be assessed. In addition, the proposed framework can be readily applied to more general problems, like multi-objective semi-infinite problems [44, 27, 62, 26].

The extension of this framework from SIPs with box-constrained lower-level programs to SIPs with lower-level programs with general nonlinear inequality constraints is straightforward by handling inner and boundary boxes in the parameter paving. Extending it to generalized SIP presents more subtle issues, and is currently under work.

Aknowledgment

This work was partially supported by the French Agence National de la Recherche (ANR) [grant number ANR-16-CE33-0024].

[1] Araya, I., Trombettoni, G., 2010. Exploiting monotonicity in interval constraint propagation. In: In Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI.

[2] Araya, I., Trombettoni, G., Neveu, B., 2012. A contractor based on convex interval taylor. In: Beldiceanu, N., Jussien, N., Pinson, É. (Eds.), Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28 – June1, 2012. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–16.

[3] Araya, I., Trombettoni, G., Neveu, B., Chabert, G., 2014. Upper bounding in inner regions for global optimization under inequality constraints. Journal of Global Optimization 60 (2), 145–164.

[4] Armand, P., Benoist, J., Bousquet, E., Delage, L., Olivier, S., Reynaud, F., 2009. Optimization of a one dimensional hypertelescope for a direct imaging in astronomy. European Journal of Operational Research 195 (2), 519 – 527.

[5] Ben-Tal, A., den Hertog, D., Vial, J.-P., Feb 2015. Deriving robust counterparts of nonlinear uncertain inequalities. Mathematical Programming 149 (1), 265–299.

[6] Ben-Tal, A., Ghaoui, L. E., Nemirovski, A., 2009. Robust Optimization. Princeton Series in Applied Mathematics. Princeton University Press.

[7] Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F., 1999. Revising hull and box consistency. In: Int. conf. on logic programming. MIT press, pp. 230–244.

[8] Benhamou, F., Granvilliers, L., 2006. Chapter 16 - continuous and interval constraints. In: Francesca Rossi, P. v. B., Walsh, T. (Eds.), Handbook of Constraint Programming. Vol. 2. Elsevier, pp. 571 – 603.

[9] Benhamou, F., McAllister, D., Hentenryck, P. V., 1994. CLP(Intervals) Revisited. In: International Symposium on Logic Programming. pp. 124–138.

[10] Berger, N., Soto, R., Goldsztejn, A., Caro, S., Cardou, P., 2010. Finding the maximal pose error in robotic mechanical systems using constraint programming. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J. M., Ali, M. (Eds.), Trends in Applied Intelligent Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 82–91.

[11] Bhattacharjee, B., Green, W. H., Barton, P. I., 2005. Interval methods for semi-infinite programs. Computational Optimization and Applications 30 (1), 63–93.

[12] Blankenship, J. W., Falk, J. E., 1976. Infinitely constrained optimization problems. Journal of Optimization Theory and Applications 19 (2), 261–281.

[13] Caro, S., Chablat, D., Goldsztejn, A., Ishii, D., Jermann, C., 2014. A branch and prune algorithm for the computation of generalized aspects of parallel robots. Artificial Intelligence 211, 34 – 50.

[14] Chabert, G., 2019. Ibex 2.8 : Interval-based explorer, `http://www.ibex-lib.org/`, GitHub project: `https://github.com/ibex-team/ibex-lib`.

[15] Chabert, G., Jaulin, L., 2009. Hull consistency under monotonicity. In: Principles and Practice of Constraint Programming CP 2009. Vol. 5732 of Lecture Notes in Computer Science. Springer, pp. 188–195.

[16] Clarke, F. H., 1981. Generalized gradients of lipschitz functionals. Advances in Mathematics 40 (1), 52 – 67.

[17] Clarke, H. C., 1975. Generalized gradients and their applications. Transactions of the AMS 205, 247–262.

[18] Collavizza, H., Delobel, F., Rueher, M., 1999. Comparing partial consistencies. Reliable Computing 5 (3), 213–228.

[19] Djelassi, H., Mitsos, A., 2016. A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs. Journal of Global Optimization, 1–27.

[20] Domes, F., Goldsztejn, A., 2017. A branch and bound algorithm for quantified quadratic programming. Journal of Global Optimization 68 (1), 1–22.

[21] Fernández, J., Tóth, B., 2009. Obtaining the efficient set of nonlinear biobjective optimization problems via interval branch-and-bound methods. Computational Optimization and Applications 42 (3), 393–419.

[22] Floudas, C., 2013. Deterministic global optimization: theory, methods and applications. Vol. 37. Springer.

[23] Floudas, C. A., Stein, O., 2008. The adaptive convexification algorithm: A feasible point method for semi-infinite programming. SIAM Journal on Optimization 18 (4), 1187–1208.

[24] Gabrel, V., Murat, C., Thiele, A., 2014. Recent advances in robust optimization: An overview. European Journal of Operational Research 235 (3), 471 – 483.

[25] GmbH, G. S., 2017. Gams : General algebraic modeling system, https://www.gams.com.

[26] Goberna, M., Guerra-Vazquez, F., Todorov, M., 2016. Constraint qualifications in convex vector semi-infinite optimization. European Journal of Operational Research 249 (1), 32 – 40.

[27] Goberna, M., Jeyakumar, V., Li, G., Vicente-Prez, J., 2015. Robust solutions to multi-objective linear programs with uncertain data. European Journal of Operational Research 242 (3), 730 – 743.

[28] Goldsztejn, A., Caro, S., Chabert, G., 2016. A three-step methodology for dimensional tolerance synthesis of parallel manipulators. Mechanism and Machine Theory 105, 213 – 234.

[29] Goldsztejn, A., Domes, F., Chevalier, B., 2014. First order rejection tests for multiple-objective optimization. J Glob Optim 58, 653–672.

[30] Goldsztejn, A., Michel, C., Rueher, M., 2008. Efficient Handling of Universally Quantified Inequalities. Constraints 14 (1), 117–135.

[31] Goualard, F., 2014. How do you compute the midpoint of an interval? ACM Trans. Math. Softw. 40 (2), 11:1–11:25.

[32] Goualard, F., 2015. Gaol 4.2: Not just another interval arithmetic library, http://sourceforge.net/projects/gaol.

[33] Hansen, E., Walster, G. W., 2003. Global Optimization Using Interval Analysis - Revised And Expanded. CRC Press.

[34] Hettich, R., Kortanek, K. O., 1993. Semi-infinite programming: Theory, methods, and applications. SIAM Review 35 (3), 380–429.

[35] Houska, B., Diehl, M., 2013. Nonlinear Robust Optimization via Sequential Convex Bilevel Programming. Mathematical Programming, Series A 142, 539?577.

[36] Jaulin, L., Kieffer, M., Didrit, O., Walter, E., 2001. Applied interval analysis: with examples in parameter and state estimation, robust control and robotics. Springer Verlag.

[37] Johansson, F., Aug 2017. Arb: Efficient arbitrary-precision midpoint-radius interval arithmetic. IEEE Transactions on Computers 66 (8), 1281–1292.

[38] Kannan, R., Barton, P. I., 2017. The cluster problem in constrained global optimization. Journal of Global Optimization 69 (3), 629–676.

[39] Kearfott, R. B., 1996. Interval Computations: Introduction, Uses, and Resources. Euromath Bulletin 2 (1), 95–112.

[40] Kearfott, R. B., 1996. Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers.

[41] Kirst, P., Stein, O., Steuermann, P., Jul 2015. Deterministic upper bounds for spatial branch-and-bound methods in global minimization with nonconvex constraints. TOP 23 (2), 591–616.

[42] Lhomme, O., 2005. Quick Shaving. In: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1. AAAI'05. AAAI Press, pp. 411–415.

[43] Lopez, M., Still, G., 2007. Semi-infinite programming. European Journal of Operational Research 180 (2), 491 – 518.

[44] Martin, B., Goldsztejn, A., Granvilliers, L., Jermann, C., 2017. Constraint propagation using dominance in interval branch & bound for nonlinear biobjective optimization. European Journal of Operational Research (EJOR) 260 (3), 934 – 948.

[45] Mitsos, A., 2009. Test set for semi-infinite programs. Tech. rep., Massachusetts Institute of Technology, http://web.mit.edu/mitsos/www/pubs/siptestset.pdf.

[46] Mitsos, A., 2011. Global optimization of semi-infinite programs via restriction of the right-hand side. Optimization 60 (10-11), 1291–1308.

[47] Mitsos, A., Lemonidis, P., Lee, C., Barton, P., 2008. Relaxation-based bounds for semi-infinite programs. SIAM Journal on Optimization 19 (1), 77–113.

[48] Moore, R., 1966. Interval Analysis. Prentice-Hall.

[49] Neumaier, A., 1991. Interval Methods for Systems of Equations. Cambridge University Press.

[50] Neumaier, A., 1996. Second-order sufficient optimality conditions for local and global nonlinear programming. Journal of Global Optimization 9 (2), 141–151.

[51] Neumaier, A., 2004. Complete search in continuous global optimization and constraint satisfaction. Acta Numerica, 271–369.

[52] Neumaier, A., 5 2004. Complete search in continuous global optimization and constraint satisfaction. Acta Numerica 13, 271–369.

[53] Neveu, B., Trombettoni, G., Araya, I., 2015. Adaptive constructive interval disjunction: algorithms and experiments. Constraints, 1–16.

[54] Sahinidis, N. V., 2017. BARON : Global Optimization of Mixed-Integer Nonlinear Programs, `http://archimedes.cheme.cmu.edu/?q=baron`.

[55] Stein, O., 2012. How to solve a semi-infinite optimization problem. European Journal of Operational Research 223 (2), 312–320.

[56] Still, G., Oct 2001. Discretization in semi-infinite programming: the rate of convergence. Mathematical Programming 91 (1), 53–69.

[57] Trombettoni, G., Chabert, G., 2007. Constructive interval disjunction. In: Bessière, C. (Ed.), Principles and Practice of Constraint Programming CP 2007. Vol. 4741 of Lecture Notes in Computer Science. Springer, pp. 635–650.

[58] Tsoukalas, A., Rustem, B., Nov 2011. A feasible point adaptation of the blankenship and falk algorithm for semi-infinite programming. Optimization Letters 5 (4), 705–716.

[59] Van Hentenryck, P., Michel, L., Deville, Y., 1997. Numerica: A Modeling Language for Global Optimization. MIT press.

[60] Wechsung, A., Schaber, S. D., Barton, P. I., 2014. The cluster problem revisited. Journal of Global Optimization 58 (3), 429–438.

[61] Wunderling, R., 1996. Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin.

[62] Zamani, M., Soleimani-damaneh, M., Kabgani, A., 2015. Robustness in nonsmooth nonlinear multi-objective programming. European Journal of Operational Research 247 (2), 370 – 378.