J Glob Optim (2016) 65:837–866
DOI 10.1007/s10898-015-0390-4

Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects

Ignacio Araya[1] · **Victor Reyes**[2]

Received: 28 April 2014 / Accepted: 6 December 2015 / Published online: 29 December 2015

Abstract Interval Branch and Bound algorithms are used to solve rigorously continuous constraint satisfaction and constrained global optimization problems. In this paper, we explain the basic principles behind interval Branch and Bound algorithms. We detail the main components and describe issues that should be considered to improve the efficiency of the algorithms.

Keywords Interval arithmetic · Constraint propagation · Numerical constrained optimization · Numerical constraint satisfaction · Interval-based solver · Branch and Bound

1 Introduction

Interval-based methods have gained considerable interest over the last decade. Two main features distinguish interval-based methods from other methods:

- They account for uncertainties in the parameter values and computation errors over the floating-point numbers.
- They are able to treat the complete search space, thus offering proofs of infeasibility and/or certification of solutions.

Because of these features, these methods are being used in several research areas, including robotics, localization, model qualification, and the design of control systems. In [53,68,81], robots and vehicles obtain useful information from a set of sensors (e.g., odometers, GPS or gyrocompass). Then, this information is transformed into a set of non-linear systems of

✉ Ignacio Araya
ignacio.araya@pucv.cl

Victor Reyes
vreyes@inf.utfsm.cl

[1] Escuela Informática, Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2950, V Región, Chile

[2] Depto. Informática, Universidad Técnica Federico Santa María, Avenida España, 1680, V Región, Chile

constraints. Solving these systems means finding the position, displacement, or required movement for the vehicle/robot. Uncertainties of sensor measurement errors, mechanical parts or floating-point number computation errors are considered by interval-based methods. Interval-based techniques have also been used successfully for localizing nodes in mobile networks and tracking acoustical sources [64,65,91,104]. In designing robust control systems, the quantitative feedback theory (QFT) method has been studied by interval-based techniques in several works (e.g., [92,93,98]). QFT uses the feedback of measurable plant/machinery outputs to generate an acceptable response. Usually, the general problem in QFT is related to how to design the feedback controller and prefilter such that the desired specifications are satisfied in the region of uncertainty. In [98], the authors propose posing the prefilter problem as a constraint satisfaction problem following a set of rules. The problem is then solved using interval-based techniques. The approach is validated using a magnetic levitation system. In [99,100], the authors propose finding reachable sets using interval-based methods, i.e., finding all the trajectories on a hybrid system (involving discrete and continuous dynamics), starting from a possible initial set of states under disturbances and uncertainties in parameter values.

Interval Branch and Bound (B&B) strategies are primarily used to solve continuous constraint systems and to handle constrained global optimization problems. These strategies are *mathematically rigorous*, i.e., they account for the rounding errors that are implied by floating-point operations in their implementations. The story of interval B&B started with interval analysis [90]. Numerous interval-based techniques for numerical unconstrained optimization appear in Ratz's dissertation [102] and are included in the Pascal-XSC solver described in [45]. In [47], Hansen provides an informal description of many techniques and heuristics for use in unconstrained optimization algorithms. In [57], Kearfott proposes a solver containing techniques for the monotonicity test and a simple technique for computing upper bounds. In [101], Ratschek explains the fundamentals for handling inequality constraints, and in [32], Hansen discusses many interval techniques for both inequality and equality constraints. In the mid-1990s, Kearfott designed GlobSol [59], a solver for constrained global optimization problems. Researchers from the constraint programming community designed the constrained global optimization solvers Numerica [120], introducing interval constraint propagation algorithms, Icos [70,71] and IbexOpt [2,118], introducing safe linear relaxations. The optimizer IBBA [83–85,96], integrated constraint propagation and affine arithmetic.

Despite all of the advances made to date in interval-based methods, interval B&B algorithms generally remain less efficient than global optimizers (non-linear programming solvers) from the mathematical programming community, such as BARON [107], Couenne [9] and the recent ANTIGONE [89]. The latter optimizers, however, are non-rigorous, i.e., they cannot offer any guarantee and occasionally miss the best solution due to a lack of rigorous computations over the floating-point numbers. In a recent work [5], a set of rigorous and a set of non-rigorous global optimizers were compared (BARON, Couenne, IbexOpt, IBBA, Icos and GlobSol), considering a set of 74 instances. IbexOpt and BARON offered the best results. Although BARON and Couenne were shown to be better than IbexOpt in simple instances, IbexOpt outperformed Couenne and reached the performance of BARON in the most difficult ones.

In this paper, we explain the basic principles behind interval B&B algorithms and offer a review of their main components. Then, we present a set of issues to be considered to improve the performance of current interval B&B algorithms.

Section 2 provides basic notions about interval arithmetic. In Sect. 3, the constraint satisfaction and constrained global optimization problems are defined. Section 4 describes the general structure of interval-based solvers. Sections 5, 6, 7 and 8 detail the main components

of interval-based solvers, describing the most used methods and discussing related issues. Conclusions are given in Sect. 9.

2 Interval arithmetic: notation and basic definitions

An **interval** $\boldsymbol{x}_i = [\underline{x_i}, \overline{x_i}]$ is the set of real numbers between $\underline{x_i}$ and $\overline{x_i}$. $\underline{x_i}$ denotes the minimum of $\boldsymbol{x}_i$, and $\overline{x_i}$ denotes the maximum of $\boldsymbol{x}_i$. The width of $\boldsymbol{x}_i$ is $\text{wid}(\boldsymbol{x}_i) = \overline{x_i} - \underline{x_i}$. A **degenerated interval** is an interval with width 0. $\text{mid}(\boldsymbol{x}_i)$ denotes the midpoint of $\boldsymbol{x}_i$. A **box** $\boldsymbol{x}$ is the Cartesian product of intervals $\boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n$. Depending on the case, $\boldsymbol{x}$ may also denote a vector of intervals $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. $\mathbb{IR}$ denotes the set of all of the intervals.

A function $f{:}\mathbb{R}^n \to \mathbb{R}$ is **factorable** if it can be computed in a finite number of simple steps, starting with model variables and real constants, using elementary unary and binary operators. Factorable functions are extended to intervals. $\boldsymbol{f}{:}\mathbb{IR}^n \to \mathbb{IR}$ is said to be an **extension** of the factorable function f to intervals. $\boldsymbol{f}(\boldsymbol{x})$ contains the image of f over $\boldsymbol{x}$, i.e., $\boldsymbol{f}(\boldsymbol{x}) \supset \{f(x), x \in \boldsymbol{x}\}$. The **optimal image** ($\boldsymbol{f}_{opt}$) is the sharpest interval containing $\{f(x), x \in \boldsymbol{x}\}$.

Interval arithmetic defines the extension of the classic elementary operators (e.g., $+$, $-$, $*$, $/$, $power$, exp, log, sin, cos,...). For instance, $[a, b] + [c, d] = [a + b, c + d]$, $[log]([a, b]) = [log(a), log(b)]$, etc. The **natural extension** $\boldsymbol{f}_N$ of a factorable function f corresponds to the mapping of f to intervals using the corresponding interval operators. For instance, consider the function $f(x_1, x_2, x_3) = x_1 + (x_2 \cdot x_3)^2 + 4$. The natural extension of f is given by:

$$\boldsymbol{f}_N(\boldsymbol{x}) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 \cdot \boldsymbol{x}_3)^2 + 4$$

If we consider the following domains of variables, $\boldsymbol{x}_1 = [1, 2], \boldsymbol{x}_2 = [2, 4]$ and $\boldsymbol{x}_3 = [-1, 1]$, we can compute the image of the natural extension or **natural evaluation**; that is, $[1, 2] + ([2, 4] * [-1, 1])^2 + 4 = [5, 22]$. In this case, the image is optimal, i.e., $\boldsymbol{f}_N(\boldsymbol{x}) = \boldsymbol{f}_{opt}(\boldsymbol{x})$; however, this is not the general case.

A common way to compute the image of an interval function consists of representing the factorable function as an **expression Direct Acyclic Graph** (expression DAG). Variables and constraints are represented by leaves in the DAG, and the operators are represented by vertices or nodes. Thus, the function or expression corresponds to the root node (see Fig. 1). The computation of the image of $\boldsymbol{f}$ is performed by replacing the variables with their related intervals/domains and applying, recursively, interval arithmetic in each node.

The main reason that the computation of optimal image approximations is difficult is known as the **dependency problem**. The dependency problem occurs in expressions with multiple occurrences of variables. This problem explains, for instance, why the extension to intervals of the expression $x_1 - x_1$ cannot be computed optimally, i.e., $\boldsymbol{x}_1 - \boldsymbol{x}_1 \neq [0, 0]$. The reason is that the interval operators consider each occurrence of a variable as an independent occurrence. Thus, $\boldsymbol{x}_1 - \boldsymbol{x}_1$ is equivalent to $\boldsymbol{x}_1 - \boldsymbol{x}_2$ when the intervals related to x_1 and x_2 are equivalent. If f is continuous in a box $\boldsymbol{x}$ and has *no multiple occurrences of variables*, then the natural extension computes an optimal image approximation of f over $\boldsymbol{x}$.

Several extensions, focused on reducing the dependency problem, have been proposed thus far. Among the most well-known extensions, we find the monotonicity-based extension and the Taylor-based extension. The **monotonicity-based extension** (ME) first computes a lower and upper bound of the gradient of f to identify the *monotonic variables*. **Monotonic variables** are all of the variables x_i such that f is monotonically increasing (or decreasing)

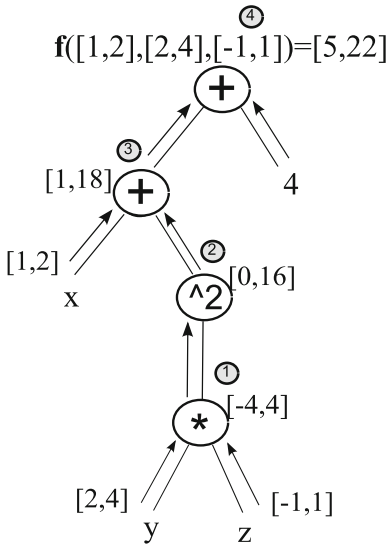

Fig. 1 The evaluation of $[1, 2] + ([2, 4] * [-1, 1])^2 + 4$ performed in the corresponding DAG

w.r.t. x_i. These variables are fixed to the value that maximizes (resp. minimizes) the evaluation of f. Then, an upper bound (resp. lower bound) of the image of f is computed.

$$\boldsymbol{f}_M(\boldsymbol{x}) = \left[\underline{\boldsymbol{f}_N(x_1^-, \ldots, x_n^-, \boldsymbol{w})}, \overline{\boldsymbol{f}_N(x_1^+, \ldots, x_n^+, \boldsymbol{w})}\right]$$

x_i^+ (resp. x_i^-) is the value of the monotonic variable x_i that maximizes (resp. minimizes) f. $\boldsymbol{w}$ is a vector of non-detected monotonic variables. Note that the bounds of the image of f are computed using the natural extension. Any other extension could be used instead.

The *occurrence grouping extension* OG [2] is a variant of ME. It generates a new function f^{og} that increments the number of monotonic variables of $f \cdot f^{og}$ is then evaluated using ME. The increment of monotonic variables in f^{og} implies that OG always computes a better (or equal) image approximation than ME.

Consider, for example, the function $f(x, w) = x^2 - 5x - w^3 + 2w^2 + 14w$ and the variable domains $\boldsymbol{x} = [0, 2]$ and $\boldsymbol{w} = [-2, 1]$. The natural evaluation of f in the domains is $[-39, 53]$. By using an interval gradient, the function is detected to be monotonically decreasing w.r.t. x. Thus, ME computes a sharper interval image: $[-35, 39]$. OG generates a new function, f^{og}, replacing each occurrence of the Non-detected monotonic variable w by a convex linear combination $r_a w_a + r_b w_b + r_c w_c$ such that f^{og} is monotonically increasing w.r.t. w_a and monotonically decreasing w.r.t. w_b:

$$f^{og}(x, w_a, w_b, w_c) = x^2 - 5x - w_a^3 + 2(0.25w_a + 0.75w_c)^2 + 14w_a$$

Finally, by evaluating f^{og} using ME, OG computes a better image approximation of f: $[-26, 16.25]$.

In general, Taylor extensions [63,75] are of the form: $\boldsymbol{f}_{T_d}(\boldsymbol{x}) = \boldsymbol{P}_d(\boldsymbol{x} - \tilde{x}) + \boldsymbol{e}_d$, where $P_d(x)$ is a degree-d polynomial approximation of f over $\boldsymbol{x}$, $\tilde{x}$ is a base point (often the midpoint of $\boldsymbol{x}$), and the interval $\boldsymbol{e}_d$ encompasses the truncation error of the polynomial over $\boldsymbol{x}$. In particular, **the first-order Taylor extension** follows a derivation of the mean value theorem, and it is given by the following definition [90]:

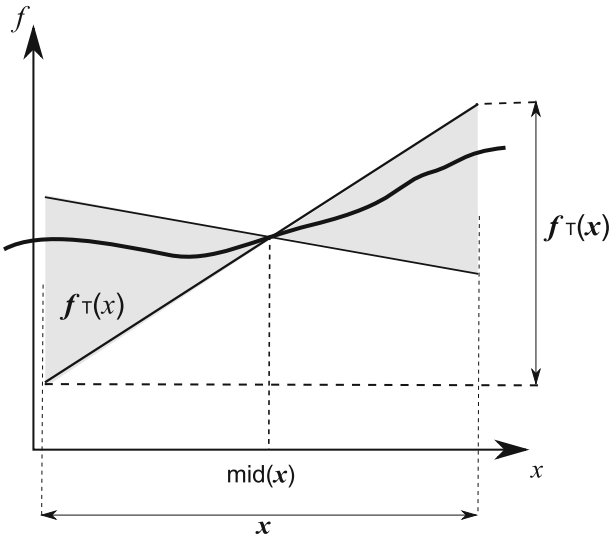

Fig. 2 The linearization of f by using the Taylor extension. $\boldsymbol{f}_T(\boldsymbol{x})$ computes the image of the linearization

$$\boldsymbol{f}_T(\boldsymbol{x}) = f(\text{mid}(\boldsymbol{x})) + \sum_{i=1}^{n} \boldsymbol{a}_i(\boldsymbol{x}_i - \text{mid}(\boldsymbol{x}_i))$$

where $\boldsymbol{a}_i$ is an interval overestimation of the image of $\frac{\partial f}{\partial x_i}$ over $\boldsymbol{x}$. Basically, the extension computes the image of a non-convex interval linearization of f. The linearization is obtained by using the interval gradient of f in the box (see Fig. 2). A better approximation is made by **Hansen's extension** $\boldsymbol{f}_H$ [50]. Hansen's extension can be observed as a recursive variant of the first-order Taylor extension.

$$\boldsymbol{f}_H(\boldsymbol{x}) = f(\text{mid}(\boldsymbol{x})) + \sum_{i=1}^{n} \boldsymbol{a}_i'(\boldsymbol{x}_i - \text{mid}(\boldsymbol{x}_i))$$

where $\boldsymbol{a}_i$ is an interval overestimation of the image of $\frac{\partial f}{\partial x_i}$ over $\boldsymbol{x}^{(i)}$, and the box $\boldsymbol{x}^{(i)}$ is equivalent to $\boldsymbol{x}$ with $n - i$ intervals of zero width:

$$\boldsymbol{x}^{(i)} = \boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_i \times \text{mid}(\boldsymbol{x}_{i+1}) \times \cdots \times \text{mid}(\boldsymbol{x}_n)$$

Another type of Taylor extension is based on *interval slopes* [66]. Given two points x and c in box $\boldsymbol{x}$, the vector $f[x, c] \in \mathbb{R}^n$ is called a slope of f between x and c if $f(x) = f(c) + f[x, c](x - c)$ holds. Consider a point $c \in \boldsymbol{x}$ and $\boldsymbol{f}[\boldsymbol{x}, c] \in \mathbb{IR}^n$ an interval slope that bounds $f[x, c]$ over all $x \in \boldsymbol{x}$. The extension based on interval slopes is given by:

$$\boldsymbol{f}_S(\boldsymbol{x}) = f(c) + \boldsymbol{f}[\boldsymbol{x}, c](\boldsymbol{x} - c)$$

Slopes can be calculated in a similar fashion as the interval derivatives [13,110]. Extensions based on interval slopes usually provide better image approximations than the first-order Taylor extension, particularly if the center of the slope is carefully chosen [78].

3 Problem formulations

In this section, we describe the two problems that will be addressed in this work.

Definition 1 (*Numerical constraint satisfaction problem* (*NCSP*)) A **numerical CSP**, or constraint system $S = (C, x, \boldsymbol{x})$, consists of a vector $x = (x_1, \ldots, x_n)$ of variables varying in a box $\boldsymbol{x} \in \mathbb{IR}^n$ and a set of constraints C. C includes a set of equality constraints $h_1(x) = 0, \ldots, h_p(x) = 0$, where $h_i : \mathbb{R}^n \to \mathbb{R}, \forall i$, and a set of inequality constraints $g_1(x) \leq 0, \ldots, g_q(x) \leq 0$, where $g_j : \mathbb{R}^n \to \mathbb{R}, \forall j$. A **solution** $x' \in \boldsymbol{x}$ of S satisfies all constraints.

When used to compute the set of all of the solutions of an NCSP, interval-based methods generally find a set of *atomic* boxes B_ϵ (i.e., boxes smaller than the required precision), such that all of the solutions to the problem belong to at least one of these boxes. It is possible that there are some boxes in B_ϵ that do not contain any solution to the problem.

Definition 2 (*Numerical Constrained global Optimization Problem* (*NCOP*)) Consider the constraint system $S = (C, x, \boldsymbol{x})$, where x is a vector of n variables. Consider a real function $f_{obj} : \mathbb{R}^n \to \mathbb{R}$. The **numerical constrained global optimization problem**, related to the **objective function** f_{obj} and the system of constraints S, consists of finding:

$$x^* = \min_{x \in \boldsymbol{x}} f_{obj}(x) \text{ s.t. } x \text{ satisfies all of the constraints in } C$$

If $x' \in \boldsymbol{x}$ is the solution to S, then it is said to be **feasible**.

In this case, the objective of the interval-based optimizer is generally to find one ϵ-**optimal** solution, i.e., a feasible point x'[1] such that $f_{obj}(x') - f_{obj}(x^*) \leq \epsilon_{obj}$, where ϵ_{obj} corresponds to the desired precision of the method. Some variants try to find a set of atomic boxes containing all of the ϵ-optimal solutions. Although we refer primarily to the former optimizers in the following, most of the presented methods are applicable to both of them.

4 Interval-based algorithm

In this section, we describe the basic structure of interval-based algorithms for NCOPs and NCSPs. Both types of algorithms are typically based on the B&B strategy. The main procedures and methods, which are covered in the next sections, have been highlighted in bold.

4.1 NCSP algorithm

Consider the NCSP $S = (C, x, \boldsymbol{x})$. Algorithm 1 shows the basic skeleton of an NCSP algorithm. The search tree is implemented using a list of leaves, L. It achieves an exhaustive exploration of the search space with the objective of finding a set of atomic boxes containing all of the solutions. The procedure is initialized with a box, $\boldsymbol{x}$, that contains the initial domain of each variable in the system. The algorithm returns a set of atomic boxes, B_ϵ, that contain all of the solutions to the problem.

In each iteration, a node is selected from L. Usually, the algorithm follows a *depth-first search strategy*. This can be implemented by treating L as a stack; i.e., nodes are removed from and inserted at the front of L. Each node is treated by two methods: branching/bisection and pruning. The branching, or **bisection**, consists of splitting the current box into several

[1] or an atomic box containing a feasible point.

Algorithm 1 `NCSPalgo`$(C, x, \boldsymbol{x}, \epsilon)$; **out:** B_ϵ

```
L ← {x}
while L ≠ ∅ do
  x ←select-and-remove(L)
  (x^r, x^l) ← bisect(C, x, x)
  for all x ∈ {x^r, x^l} do
    x ← pruning(C, x, x)
    if x ≠ ∅ then
      if is-atomic-box?(x, ϵ) then
        B_ϵ ← B_ϵ ∪ {x}
      else
        insert(L, x)
      end if
    end if
  end for
end while
```

sub-boxes (generally two). The new boxes are then treated by the pruning method. Two decisions are made by the bisection method: on which variable/interval to bisect the box and on which value of the domain of the variable to do so. More details about branching/bisection methods are given in Sect. 6.

After branching, the `pruning` procedure treats the new pair of boxes by turn. This procedure is compounded by one or several contraction methods or by **contractors**. Contractors attempt to filter the boxes by eliminating inconsistent values from their bounds without a loss of solutions. If all values in a box are filtered, an empty box is returned. Contractors are studied in more detail in Sect. 5.

After pruning, empty boxes are discarded from L. Non-empty boxes are put in L only if they are not *atomic boxes*; otherwise, they are put in B_ϵ. At the end of the search, B_ϵ contains the set of all solutions to the problem. There are some techniques to show the existence of a unique solution in a box (e.g., interval Newton [90]). They are used to detect atomic boxes in B_ϵ that certainly contain solutions. These techniques work only when certain conditions are satisfied in the box.

Observe the example in Fig. 3. The node/box 1 represents the initial domain of the variables x and y. Each curve in the box represents an equality constraint; Thus, each point of a curve satisfies the corresponding constraint. The problem has two solutions: the two intersections of the curves.

In the first iteration of the algorithm, box 1 is selected. It is bisected on x, and two new boxes are created. Each of them is contracted by using the pruning method. The contraction reduces each box to a smaller white box. These boxes are not atomic; therefore, they are added into the stack L. In the second iteration, box 2 is taken from L. (It is the first box in the stack.) The procedure is repeated for this node; the box is bisected on y, and each of the sub-boxes is pruned. Note that the algorithm has detected that the right box does not contain any solution; therefore, it is discarded. However, the left box is just contracted and put in L. The search continues with box 3 and its entire corresponding subtree before passing on to node 4. When node 4 is treated, two sub-boxes are again generated by the bisection. In this case, the pruning reduces the left box to an atomic one, which is added to B_ϵ. The right box is discarded by the pruning.

Although the depth-first search strategy is usually used due to its constant complexity in memory, some variants have been proposed to quickly discover potential solutions in different areas of the search space [24].

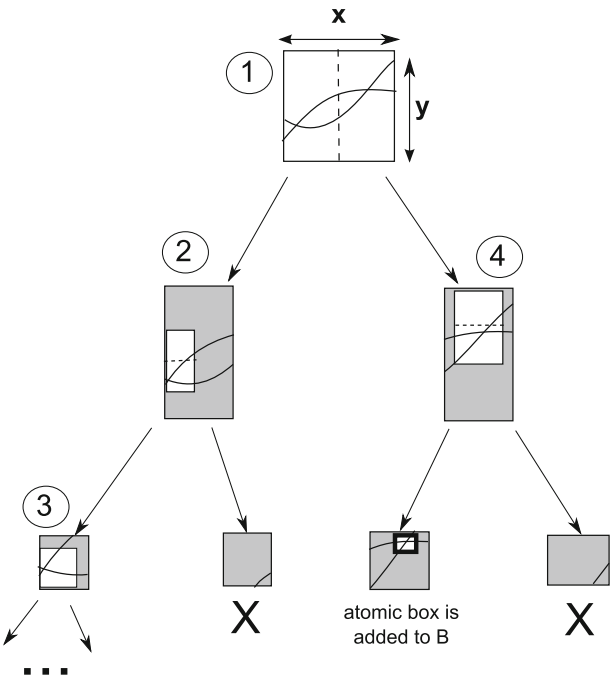

Fig. 3 Example of a search tree for an NCSP problem consisting of two variables and two constraints

A well-known problem related to interval B&B algorithms (NCSP and NCOP algorithms) is known as the *cluster problem* [30,109]. The cluster problem (or cluster effect) consists of the excessive splitting of boxes close to a solution and the failure to remove many boxes not containing the solution. As a consequence, these algorithms slow down considerably once they reach regions close to the solutions. In [109], Schichl and Neumaier discuss the reasons why this problem occurs. They also propose methods for defining exclusion regions (boxes and polytopes) around each solution found. These regions are guaranteed to contain no other solution and thus can be safely discarded. Exclusion regions can also be defined around arbitrary points guaranteed to contain no solution.

Another issue of NCSP algorithms is related to finding regions of solutions when the system is under-constrained. Only a few works [33,38,40,105,108,125] address this problem. In general, if the precision of the solver is set to be too small, the number of atomic boxes containing solutions may overflow the memory of the machine. Furthermore, from such a large number of atomic boxes, it may be very difficult to obtain useful information. When the system becomes highly dimensional, the solution set may be better visualized in other ways (e.g., projection of the solution set over certain parameters/variables [51]). In [122], the authors propose a method for constructing compact approximations of the complete set of solutions. They combine an efficient representation of orthogonal polyhedra [14] with adapted splitting strategies to construct the approximation as unions of boxes. In [40], the authors propose representing the search space using a parallelepiped instead of boxes to extend the power of the interval Newton to under-constrained systems.

4.2 NCOP algorithm

Algorithm 2 details a basic implementation of an NCOP solver. As in the NCSP solver, starting from an initial node, the NCOP solver builds a search tree using a B&B schema. In this case, each node is represented by a pair of variables, $(\boldsymbol{x}, lb)$. $\boldsymbol{x}$ corresponds to the domain space of the variables, and lb is a (generally infeasible) lower bound for the objective function, f_{obj}, in the box $\boldsymbol{x}$ and with the constraints C. In the root node, lb takes the value $-\infty$. The value is improved by the pruning method. LB corresponds to the minimum value of the lower bounds for all leaf nodes. Additionally, UB is the cost of the current best feasible point found during the search. A termination occurs when $UB - LB$ reaches a precision ϵ_{obj}, and a *quasi-optimal solution* x_{UB} with cost $UB = \overline{f_{obj}(x_{UB})}$ is returned.

The same **branching** and **contractors** of the NCSP solver may be used here. An additional constraint related to the objective function may be added to increase the pruning: $f_{obj}(x) \leq UB$. Instead of just adding the constraint, we can add a new variable y with initial domains $\boldsymbol{y} = [lb, UB]$ and the constraint $y = f_{obj}(x)$. In this way, an eventual contraction of the left bound of $\boldsymbol{y}$ implies an improvement of the lower bound corresponding to the node: $lb \leftarrow \underline{y}$. To improve the lower bound, some solvers also compute an approximation of the image using different interval extensions (e.g., [78]).

In unconstrained problems (i.e., $C = \emptyset$), a monotonicity test may be applied to box $\boldsymbol{x}$. If f_{obj} is monotone w.r.t. some coordinates of $\boldsymbol{x}$, then each related interval can be reduced to its respective lower or upper bound. In these problems, it is also possible to add the system of equations $\bigtriangledown f_{obj}(x) = 0$ to restrict the search to the stationary points of f_{obj} in $\boldsymbol{x}$. These additional equations can be used only in boxes strictly contained in the initial box [78].

Algorithm 2 `NCOPalgo`(**in:** $f_{obj}, C, x, \boldsymbol{x}, \epsilon, \epsilon_{obj}$); **out:** x_{UB}, UB, B

$L \leftarrow \{(\boldsymbol{x}, -\infty)\}$; $UB \leftarrow +\infty$; $LB \leftarrow -\infty$
while $L \neq \emptyset$ **and** $UB - LB > \epsilon_{obj}$ **do**
  $(\boldsymbol{x}, lb) \leftarrow$ `select-and-remove`$(\boldsymbol{x}, lb)$
  $(\boldsymbol{x}^l, \boldsymbol{x}^r) \leftarrow$ `bisect`$(C, x, \boldsymbol{x})$

  for all $\boldsymbol{x} \in \{\boldsymbol{x}^l, \boldsymbol{x}^r\}$ **do**
    $(\boldsymbol{x}, lb') \leftarrow$ `pruning`$(f_{obj}, C, x, \boldsymbol{x}, UB)$
    if $\boldsymbol{x} \neq \emptyset$ **and** $lb < UB$ **then**
      $(x', new\_UB) \leftarrow$`upper-bounding`$(f_{obj}, C, x, \boldsymbol{x}, UB)$
      if $new\_UB < UB$ **then**
        $(x^*, UB) \leftarrow (x', new\_UB)$
        remove from L and B_ϵ any pair $(\boldsymbol{x}, lb)$ such that $lb > UB$
      end if
      if `is-atomic-box?`$(\boldsymbol{x}, \epsilon)$ **then**
        $B_\epsilon \leftarrow B_\epsilon \cup \{(\boldsymbol{x}, lb')\}$
      else
        `insert`$(L, (\boldsymbol{x}, lb'))$
      end if
    end if
  end for
  $LB \leftarrow \min_{(\boldsymbol{x}, lb) \in L \cup B_\epsilon} lb$
end while

The **upper bounding** consists of finding feasible solutions in $\boldsymbol{x}$ with costs lower than the current UB. The procedure returns a feasible point or a *certified box* (an

atomic box that certainly contains a feasible solution) plus an upper bound of the image of the objective function over this point/box (new_UB). If $new\_UB < UB$, then UB is updated. Updating UB implies a *global* reduction of the feasible space through the constraint $f_{obj}(x) \leq UB$. This potential reduction in the feasible space makes the upper bounding a crucial component in global optimizers. Cheap local search methods are generally used to find feasible points (e.g., selecting the midpoint of the box [96], applying the Newton method [41,70], extracting convex inner regions [5]), plus some feasibility tests (e.g., the natural evaluation of the candidate point [5,96], the Borsuk proof [70]). More details on the upper bounding techniques are given in Sect. 7.

For the same reason as upper bounding, the **selection of the next box** (procedure `select-and-remove`) is crucial in global optimizers. In each iteration, the algorithm should select the next box that is most likely to improve the UB the most to reduce the feasible space. The most used strategy is the best-first using as a heuristic the box with the lowest lb [5,70,96]. The algorithm hopes to find in this box a feasible point with cost lb, in which case, the search would end. In [18,77], the authors propose a set of original heuristics that also account for the size and feasibility of the box. More details about box-selection techniques are given in Sect. 8.

Nodes resulting in empty boxes after pruning and nodes that do not contain the global optimum (i.e., $lb > UB$) are discarded. If $\boldsymbol{x}$ becomes atomic, then it is added to a list B_ϵ with its associated lower bound. These boxes are not further contracted, though their lower bounds are considered for computing LB. If $\boldsymbol{x}$ is not atomic, then the pair/node is inserted on the list L. At the end of the iteration, LB is updated using the lowest lower bound in L and B_ϵ.

At the end, the solver returns one of the following results depending on the reached termination criteria:

- If $UB - LB \leq \epsilon_{obj}$, the solver returns an ϵ-optimal solution of the problem x_{UB} and its cost UB.
- Otherwise, the solver has most likely not found a solution with a cost that is sufficiently close to the optimal cost. Regardless, it returns a set of atomic boxes, B_ϵ, and the best found solution, x_{UB}. Either B_ϵ contains the optimum of the problem, or x_{UB} is an ϵ-optimal solution with cost UB.
- If neither boxes nor a feasible solution is returned, then the problem is unfeasible.

A simple way to modify the algorithm to find all of the optimal solutions consists of removing the termination criteria $UB - LB > \epsilon_{obj}$. In this way, the algorithm would return a set of atomic boxes B_ϵ containing all of the optimal solutions of the problem plus a feasible solution x_{UB}.

Figure 4 shows an example of the tree search. The node/box 1 represents the initial domain of the variables. The area inside the curve in the box represents the feasible solution space. The small star indicates the location of the optimal solution with cost of -2.5 (this value is not known by the algorithm at the beginning). In the first iteration, node 1 is selected and bisected. Its two children are contracted, and the contractors compute new lower bounds, (-7 and -6), for each child. Then, the `upper-bounding` procedure attempts to find feasible solutions to improve the current upper bound. The method was successful in the left child; it found a new upper bound equal to 10. Both children are added to L. In the second iteration, the node with the lowest upper bound is selected (node 2) and bisected, and its children are pruned. Note that the contractors may remove feasible solutions from the boxes. This is because these methods use the additional constraint related to the current

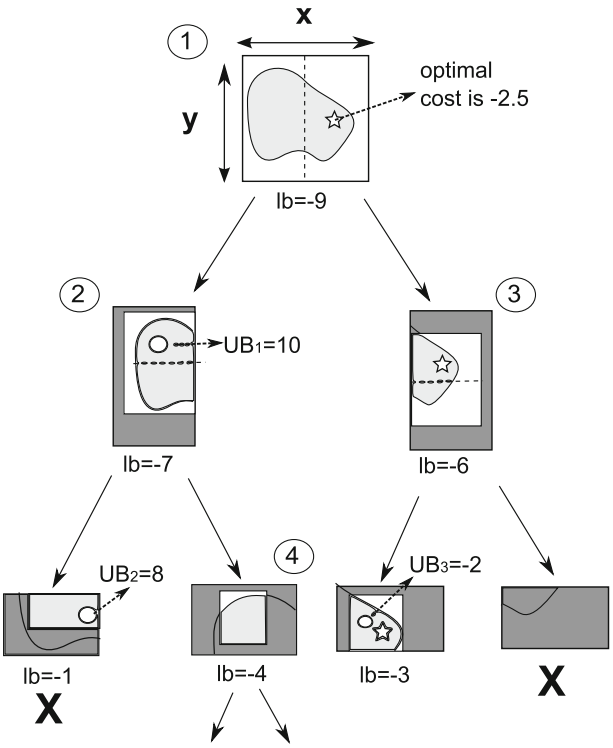

Fig. 4 Example of a search tree for an NCOP problem consisting of two variables

upper bound, i.e., $f_{obj}(x) < 10$. A new upper bound is found in the left child of node 2 ($UB = 8$). In the third iteration, node 3 is selected. After pruning, a new upper bound equal to -2 is found in its left child. Then, the box with $lb = -1$ is removed from L because its lower bound is greater than the current upper bound. The search continues with box 4…

4.3 Advanced features: use of optimality conditions

A way to reduce the cluster effect in NCOP problems is to use *optimality conditions*. Optimality conditions, such as Karush–John or Karush–Kuhn–Tucker conditions [54,55], generally state that for any local optima, a set of constraints must be satisfied; therefore, they can be used to filter the domains of variables. Currently, only a few interval-based optimizers incorporate these constraints to improve the filtering. In [35,50,57], the constraint system is solved using interval Newton, whereas techniques from constraint programming are applied in [44]. In [60], Kearfott states that interval Newton often fails in contracting the box using optimality conditions because the Jacobian matrix is commonly singular. Constraint propagation and relaxation-based contractors have nothing to do with the Jacobian matrix; furthermore, the nature of the optimality conditions is completely different from the nature of the system of constraints. For these reasons, we believe that contractors, distinct from interval Newton, may help to improve the efficiency of NCOP solvers when the optimality conditions are considered.

5 Contractors

Note that one of the main components shared by the NCSP and NCOP solvers is the contractors. These methods attempt to eliminate values from the domains of variables (search spaces) that do not satisfy one or more constraints in the constraint system. Contractors do not lose solutions, i.e., they are mathematically rigorously implemented and do not involve heuristics that could ignore solutions. We distinguish three types of contraction algorithms: contractors from interval analysis, contractors from constraint programming (or CP contractors) and linear relaxation-based contractors.

5.1 Contractors from interval analysis

One of the most effective filtering algorithms used in interval analysis is the extension of the Newton method to intervals (the *interval* Newton method). The objective of the classical Newton method is to find successively better approximations to the zeroes of a real function. Consider a continuous and differentiable univariate function $f(x)$ and its derivative $f'(x)$. If $x^{(l)}$ is an approximation of a solution of the equation $f(x) = 0$, then a better approximation is obtained by:

$$x^{(l+1)} = x^{(l)} - \frac{f(x^{(l)})}{f'(x^{(l)})}$$

The interval Newton method generalizes the procedure to intervals. Consider a univariate function $f(x)$ and an initial interval $\boldsymbol{x}$. The procedure applies successively the contraction step:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} \cap \left(\text{mid}(\boldsymbol{x}) - \frac{f(\text{mid}(\boldsymbol{x}))}{[f'](\boldsymbol{x})}\right)$$

for contracting the interval $\boldsymbol{x}$. If the procedure converges to a fixed point, then it will return an atomic interval containing the only solution of the equation $f(x) = 0$ in $\boldsymbol{x}$. If an iteration of the contraction step returns an empty interval, then there is no solution in $\boldsymbol{x}$. When the iterations do not converge to a fixed point, we cannot predict the presence or absence of solutions in $\boldsymbol{x}$.

The generalization of the method to n variables and n equations is obtained by using the mean value theorem. (further details may be found in [50,90]) to linearize the system. Consider a vector $x = (x_1, \ldots, x_n)$ of variables and a function vector $f = (f_1, \ldots, f_n)$. Using the mean value theorem, it is easy to show that the roots of f in a box $\boldsymbol{x}$ verify the following expression:

$$\exists A \in \boldsymbol{J}(\boldsymbol{x}) \text{ such that } A.y = -f(\text{mid}(\boldsymbol{x})) \tag{1}$$

where $\boldsymbol{J}$ is an interval extension of the Jacobian matrix of f in box $\boldsymbol{x}$. In other words, each element (i, j) of the matrix $\boldsymbol{J}(\boldsymbol{x})$ contains an image approximation of $\frac{\partial f_i}{\partial x_j}(x)$ over box $\boldsymbol{x}$. y is a vector of auxiliary variables: $y = x - \text{mid}(\boldsymbol{x})$. $\text{mid}(\boldsymbol{x})$ is the midpoint of the vector $\boldsymbol{x}$. Consider $\boldsymbol{y}^s$ to be a box that contains all points that satisfy the relation (1). Considering that $x = y + \text{mid}(\boldsymbol{x})$, the contraction step performed by the *multivariate interval Newton* is

$$\boldsymbol{x} \leftarrow \boldsymbol{y}^s + \text{mid}(\boldsymbol{x})$$

There are many variants of the interval Newton method, and they differ primarily in regard to the algorithm that is used to find $\boldsymbol{y}^s$. Some solvers (e.g., Ibex [23]) implement Hansen's

matrix [49] instead of the Jacobian matrix. Hansen's matrix contains sharper coefficients, allowing one to obtain a better linearization of the system. In [80], the author symbolically manipulates the elements of the matrix. Reducing multiple occurrences of variables leads to a better image approximation of the partial derivatives. Soares [113] proposes using affine arithmetic instead of interval arithmetic to maintain the linear dependence among variables. There are several different ways to linearize the system, e.g., the methods of Krawczyk [67], Borsuk [37], and Kantorovich [28,97,116]. Several methods are also used to solve the linear system, e.g., matrix inversion, Gauss–Seidel, Hansen–Bliek [50], Gaussian elimination, and LU. *Preconditioning* [46] is a technique commonly used to improve the precision of the box $\boldsymbol{y}^s$. The preconditioning performs a type of reparation of an ill-conditioned matrix $\boldsymbol{J}(\boldsymbol{x})$ to obtain more precise solutions.

The main strength of the interval Newton contractor is that it is Global, i.e., it considers all of the system to be only one constraint, thereby reducing all of the variable domains simultaneously. However, the main limitations of the interval Newton contractor are (1) it works only with equality constraints, (2) it generally works with *well-constrained* systems (i.e., square systems of independent equations with a finite number of solutions), and (3) it generally converges only when boxes are small enough [48]. In large boxes, due to interval arithmetic operations, the size of the solution $\boldsymbol{y}^s$ may increase enormously, preventing the convergence of the contraction step.[2]

5.2 Constraint-propagation algorithms

The constraint-propagation algorithms are issued from constraint programming over intervals. These algorithms focus on the domain reduction w.r.t. *only one* equation/constraint and are performed by a *revision* procedure. A propagation algorithm, similar to the well-known `AC3` algorithm [74], is then used for propagating the reductions to all of the system until a fixed point is reached.

All propagation algorithms used by interval-based solvers have similar `AC3`-like propagation procedures, which differ primarily in the revision procedure. Generally, revision procedures attempt to enforce partial consistencies over the elements of the domains. Partial consistencies arising from finite-domain CSPs do not seem reasonable when we address constraints on the real numbers (in fact, on the floating-point numbers) due to the very large domains. Hence, new partial consistencies have been defined by the interval community.

Hull consistency [11] considers only the bounds of the intervals. Revision methods enforcing hull consistency should find the minimal interval for a variable x_i containing all consistent values related to a constraint C_j and a box $\boldsymbol{x}$. However, primarily due to the dependency problem, it is generally not possible to implement such methods efficiently.

An intuitive way to filter the domains of variables is to isolate the variable that we want to filter and compute the image of the generated *projection function*. For instance, consider the constraint $x_1 + 2x_2 = 10$, with domains $\boldsymbol{x}_1 = [0, 5]$ and $\boldsymbol{x}_2 = [1, 3]$. If we want to reduce the domain of $\boldsymbol{x}_1$, we can isolate x_1 in this way: $x_1 = 10 - 2x_2$. Then, by using some interval extension (e.g., the natural extension), we can compute an image approximation of the left side of the expression or **projection function** of x_1. The result should then be intersected with the initial domain of $[x_1]$:

$$\boldsymbol{x}_1 \leftarrow \boldsymbol{x}_1 \cap (10 - 2\boldsymbol{x}_2) = [0, 5] \cap [4, 8] = [4, 5]$$

[2] By using appropriate preconditioners, the contractor can be used for contracting *certain coordinates*, even for some large boxes and in some singular cases [56].

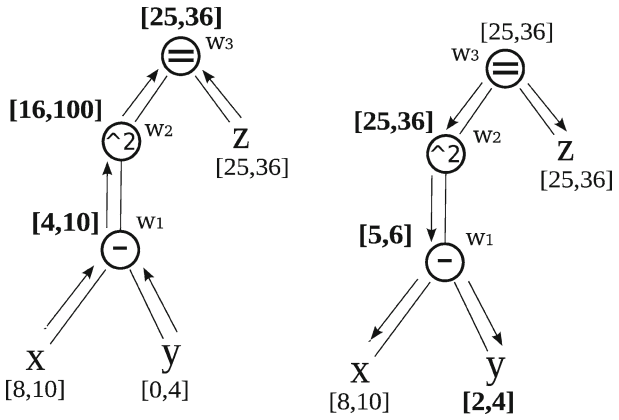

Fig. 5 Evaluation and projection phases of the `HC4-Revise` procedure in the constraint $(x - y)^2 = z$, with domains $\boldsymbol{x} = [8, 10]$, $\boldsymbol{y} = [0, 4]$ and $\boldsymbol{z} = [25, 36]$

Note that as the image of the projection function is computed using interval extensions, this method also suffers from the dependency problem. Thus, it generally cannot compute the hull consistency. A second problem occurs when a variable appears several times in a constraint, and the isolation is generally impossible. In this case, each occurrence can be isolated independently. For instance, consider the constraint $x^3 + 3x = 14$, with domain $\boldsymbol{x} = [0, 3]$. To reduce the domain of the variable, we could consider the projection functions of each occurrence and intersect their image approximations with the interval $\boldsymbol{x}$:

$$[x] \leftarrow \boldsymbol{x} \cap (-3\boldsymbol{x} + 14)^{1/3}$$
$$[x] \leftarrow \boldsymbol{x} \cap \frac{-\boldsymbol{x}^3 + 14}{3}$$

In the example, we obtain $x = [1.70, 2.42]$. After several iterations of the procedure, $\boldsymbol{x}$ converges to the solution ([2, 2]). However, if the projection functions have several occurrences of the variable to project (i.e., the variable appears more than twice in the constraint), due to the dependency problem, the projection will converge to sub-optimal domains. We refer to this particular problem as the **isolation problem**.

The algorithm `HC4-Revise` [11] filters the domains of variables without explicitly generating the projection functions. `HC4-Revise` requires traversing the expression DAG only twice to perform the projection over each occurrence of a variable. The algorithm works in two phases. The *forward* phase is recursively performed from the leaves to the root (see Fig. 5-left). This phase computes, using interval arithmetic, the natural evaluation of the subexpressions represented by the DAG nodes.

Consider, for example, the node $w_2 = w_1^2$. The related interval evaluates $\boldsymbol{w}_1^2 = [4, 10]^2 = [16, 100]$. The *backward* phase traverses the DAG top down, applying in every node a related *projection operator* (see Fig. 5-right). The projection operator reduces the intervals of the nodes, eliminating inconsistent values w.r.t. the corresponding unary or binary basic operator. Consider, for example, the node w_1, corresponding to the sub-expression $w_1 = x - y$ in Fig. 5-right. The projection operator performs the following contraction over y

$$[y] \leftarrow [0, 4] \cap ([8, 10] - [5, 6]) = [2, 4]$$

If an empty interval is obtained during the backward phase, the initial domains do not satisfy the constraint, and an empty box is returned.

Note that HC4-Revise performs an automatic projection over the intervals, using the *natural extension*. A recent work [6] describes a variant of HC4-Revise called MinMax-Revise. It uses a monotonicity-based extension to perform the filtering (more precisely, the OG extension). Just as HC4-Revise does, MinMax-Revise suffers from the isolation problem.

In the global optimization community, procedures similar to HC4 (the propagation algorithm that uses HC4-Revise) have been proposed. These procedures are known as feasibility-based bounds tightening (FBBT) [17,73,83,111]. Commonly, FBBT represents the whole constraint system as one large DAG with several roots. In this DAG, some common subexpressions appearing in several constraints are represented by single subgraphs with several parents.

BC3-Revise [11] and its variants address the isolation problem. To filter an interval related to a variable x_i in a box $\boldsymbol{x}$, BC3-Revise selects a sub-interval/slice $\boldsymbol{s}$ on a bound of $\boldsymbol{x}_i$. Then, through the use of an interval-based extension $\boldsymbol{f}$, it tests whether $\boldsymbol{f}(\boldsymbol{x}')$ satisfies the related constraint. $\boldsymbol{x}'$ is a sub-box of box $\boldsymbol{x}$, where the interval $\boldsymbol{x}_i$ has been replaced by $\boldsymbol{s}$. If the test fails, then $\boldsymbol{s}$ is removed from $\boldsymbol{x}_i$, and another bound slice is tested; otherwise, a smaller slice may be tested. Usually, BC3-Revise performs a dichotomic process to reach the *maximal reduction* on each interval bound. We call left-narrow (resp. right-narrow) the dichotomic process that improves the left (resp. right) bound of the interval. If a precision ϵ is given, then the complexity of the dichotomic process is time $O(1/\epsilon)$. Usually, to accelerate the convergence of BC3-Revise, the univariate interval Newton algorithm is called in each slice after performing the test.

To reduce the dependency problem, constraint-propagation-based algorithms may use sophisticated (but more expensive) interval extensions. For instance, Mohc [6] uses the occurrence-grouping extension, whereas the propagation algorithm proposed in [121] uses several interval extensions, including an affine arithmetic-based one.

5.2.1 The locality problem

In addition to the dependency problem, the constraint-propagation algorithms suffer from the **locality problem**. It is caused by the reduced scope of local consistencies (in discrete and continuous domains). Hull consistency, for instance, guarantees that for each constraint $f(x) \leq 0$ and for each bound of a variable, there exists a vector x_c such that $f(x_c) \leq 0$ is consistent. However, there is no guarantee that x_c is consistent in the other constraints.

Consider, for example, the system $\{x+y = 10; x-y = 0\}$, with domains $\boldsymbol{x} = [0, 10]$; $\boldsymbol{y} = [0, 10]$. The domains are hul consistent: For the value $x = 0$, the vector $(0, 10)$ is consistent in the first constraint, and the vector $(0, 0)$ is consistent in the second one. However, $x = 0$ is not globally consistent because $(0, 10)$ is not consistent in the second constraint, nor is $(0, 0)$ in the first one (the same analysis may be performed for each of the other bounds). Hull consistency may become global when the graph of the constraints has no cycles. However, this is not the general case.

Consider a constraint C and the system S, consisting of the primitive constraints of C. In particular, the filtering reached by HC4-Revise in C is equivalent to the filtering reached by HC4 in S, assuming that both algorithms are carried out until the fixed point [11]. Note that constraint decomposition transforms a dependence problem into a locality problem (and vice versa). Thus, when revision algorithms stronger than HC4-Revise are used (e.g.,

BC3-Revise [21] or Mohc-Revise), it is not advisable to decompose the constraints. These algorithms face and reduce the dependency problem but not the locality problem.

Stronger consistency techniques (particularly kB consistencies) may partially reduce the locality problem (e.g., 3B [72], 3BCID [119]). However, enforcing kB consistencies with $k > 3$ is computationally expensive. Consider the $2n$ subproblems generated by replacing one interval (n in total) with one of its bounds. Thus, a system is **3B-consistent** [72] if each of these $2n$ subproblems is hull consistent. The 3B consistency is similar to SAC for CSP, but it is limited to the bounds of the domains. Algorithms enforcing the 3B consistency (e.g., 3B, ABT [10], 3BCID[3]) generally split the intervals into several sub-intervals (slices) of width w. Then, the *3B(w)-consistency* is enforced, discarding the slices at the bounds by using the underlying revised hull consistency algorithm (e.g., HC4, Mohc, Box).

5.2.2 Issues and perspectives related to constraint-propagation algorithms

In this section, we refer to advanced features and promising ideas that, we think, merit further research. They primarily address the locality and dependence problems.

5.2.3 Common subexpression elimination

Common subexpression elimination (CSE) is an alternative for addressing the locality problem. It consists of searching for identical or common subexpressions and analyzing whether it is worthwhile to replace them with a single auxiliary variable. In [1], it was shown that transforming the original system through use of CSE improves the contraction power of the classical contractor HC4. This behavior may be explained by the reduction performed on the domains related to auxiliary variables (projection information). In the original system, the projection information is lost; however, in the new system, auxiliary variables not only store the projection information but also share it with other constraints. Algorithms such as FBBT and FBPD (forward-backward propagation on DAG [123]) detect *some* common subexpressions on the system and represent them by nodes with several parents in a DAG representation of the whole constraint system. In the work of Vu et al. [123], it is noticeable that merging equivalent nodes of the DAG helps to reduce the solver time by several orders of magnitude. This is due to CSE.

On this particular subject, there remains much to explore. Currently, CSE- and DAG-oriented techniques only replace expressions that are exactly equal. For example, the current techniques do not detect common subexpressions among the expressions $x + y$ and $2x + 2y$; however, according to [1], it is worthwhile to replace $x + y$ with an auxiliary variable v, thus generating the equivalent expressions v and $2v$.

5.2.4 Linear combination of constraints

Combining linearly the constraints to obtain an easier (or better conditioned) problem to solve seems to be a promising but unexploited idea to address the locality problem. The interval Newton, for instance, performs a preconditioning of the system using a preconditioning matrix. This approach, however, works on a linear relaxation of the original system, thus losing important nonlinear information. A few works [21,124] propose applying linear combination techniques to improve constraint-propagation algorithms. Although the results are promising, the application of these methods is restricted to a few types of instances. A

[3] Actually, 3BCID enforces CID consistency [119], a slightly stronger one.

drawback of these methods is that if special care is not taken, the constraints of the new system may involve many more terms than the original set of constraints, thus increasing the dependency problem. We expect that improvements of current techniques for approximating the image of functions will offer promising new opportunities for techniques based on the combination of constraints.

5.2.5 Symbolic preprocessing for reducing the dependency problem

Related to the dependency problem, we believe that automatic symbolic computation mechanisms are missing in most of the currently competitive interval algorithms. The sub-distributivity property of intervals states that $\boldsymbol{x}_1(\boldsymbol{x}_2 + \boldsymbol{x}_3) \subset \boldsymbol{x}_1\boldsymbol{x}_2 + \boldsymbol{x}_1\boldsymbol{x}_3$. In other words, if we have a polynomial with a common factor, then we obtain a narrower image (computed using the natural extension) if this factor is factored out. This is generally true not only for the natural extension but also for most of the interval extensions. Thus, what we should do is relatively obvious: we should factor out the common factors of polynomials to reduce the number of occurrences of variables. Of course, there is no optimal factorized form, but using simple greedy algorithms appears to be sufficient [20,22].

5.3 Linear relaxation-based contractors

There are many approaches that use linear-relaxation methods to solve optimization problems. However, most of them (e.g., the famous Baron [117]), though fast and complete, are not rigorous; i.e., they cannot guarantee their results.

Related to interval-based solvers, several linear relaxation-based contractors have emerged in recent years. They face important drawbacks compared to other interval-based approaches: the locality problem, related to constraint-propagation techniques, and the restrictive application of the interval Newton method and its variants. These methods usually work on a linear relaxation of the entire system or a part of it. The simplex algorithm is then used to narrow the domain of each variable by using the relaxed system i.e., the problem of minimizing x_i (resp. maximizing x_i) is solved to find a new lower bound (resp. upper bound) for each interval $\boldsymbol{x}_i$. Algorithm 3 shows the structure of a typical linear relaxation-based contractor [5,7,71].

Algorithm 3 `PolytopeHull`(**in:** C, x); **out:** $\boldsymbol{x}$

```
for all C_j ∈ C do
  polytope ← polytope Ulinear-relaxation(C_j)
end for
for all x_i ∈ x do
  x_i ← min x_i subject to polytope
  x̄_i ← max x_i subject to polytope
end for
```

The first loop on the constraints builds a *polytope* (i.e., a set of linear constraints), whereas the second loop on the variables contracts the domains, without a loss of solutions, by calling the simplex algorithm twice per variable.

The `linear-relaxation` method transforms a non-linear constraint C_j into a set of linear constraints Cl_j, such that any point satisfying C_j in $\boldsymbol{x}$ also satisfies the set of constraints Cl_j. The heuristics mentioned in [7] indicate in which order the variables should be handled, thus avoiding, in practice, calling the simplex algorithm $2n$ times. The application of simplex

must be mathematically rigorous. A cheap post-processing proposed by [94], which uses interval arithmetic, is commonly used to certify the solution returned by the algorithm.

There are only a few methods, based on linear-relaxation, currently used by interval-based solvers. These methods differ primarily in the linear relaxation technique. Figure 6 shows examples of the three linear-relaxation approaches explained below.

In [61], the authors propose a symbolic preprocessing step to relax nonconvex terms using linear underestimators and/or linear overestimators. The preprocessing first expands the original constraint system and objective function by decomposing the expressions. Intermediate variables are generated to represent intermediate operations.

For instance, consider the NCOP[4]: Minimize $f(x) = (x_1 + x_2 - 1)^2 - (x_1^2 + x_2^2 - 1)^2$, for $x_1 \in [-1, 1]$ and $x_2 \in [-1, 1]$. The preprocessing decomposes the objective function by adding intermediate variables, e.g., $v_3 = x_1 + x_2$, $v_4 = v_3 + 1$, and $v_5 = v_4^2$. The interval related to the variables may be initialized to the natural evaluation of the corresponding expressions, e.g., $\boldsymbol{v}_3 = [-2, 2]$, $\boldsymbol{v}_4 = [-3, 1]$, and $\boldsymbol{v}_5 = [0, 9]$. Then, the expanded equivalent problem is:

$$
\begin{aligned}
&\text{minimize } v_{11}\\
&\text{subject to } x_1 + x_2 - v_3 = 0\\
&\qquad\qquad v_3 - 1 - v_4 = 0\\
&\qquad\qquad v_4^2 - v_5 = 0\\
&\qquad\qquad v_1^2 - v_6 = 0\\
&\qquad\qquad v_2^2 - v_7 = 0\\
&\qquad\qquad v_6 + v_7 - v_8 = 0\\
&\qquad\qquad v_8 - 1 - v_9 = 0\\
&\qquad\qquad -v_9^2 - v_{10} = 0\\
&\qquad\qquad v_5 + v_{10} - v_{11} = 0\\
&\qquad\qquad x_1 \in [-1, 1],\quad x_2 \in [-1, 1]
\end{aligned}
$$

Each equality constraint in the expanded problem is analyzed to determine whether the "$=$" may be replaced by "$\leq$" or "$\geq$" such that the original and the expanded problems have the same set of optimizing points. This process is done automatically following a set of rules. In the example, the third, ninth and tenth constraints would be modified by $v_4^2 - v_5 \leq 0$, $-v_9^2 - v_{10} \leq 0$ and $v_5 + v_{10} - v_{11} \leq 0$, respectively.

Finally, the nonlinear constraints of the new problem are replaced by linear underestimators (for "$\leq$" constraints), overestimators (for "$\geq$" constraints) or both (for equality constraints). For example, the nonlinear inequality $v_4^2 - v_5 \leq 0$ can be replaced by the linear relaxation $(4.5)^2 + 9(v_4 - 4.5) - v_5 \leq 0$. The expression $(4.5)^2 + 9(v_4 - 4.5)$ corresponds to the tangent line to v_4^2 at $v_4 = 4.5$ (the midpoint of the interval $\boldsymbol{v}_4^2$). Each equality is replaced by one underestimator and one overestimator; for instance, $v_1^2 - v_6 = 0$ is replaced by $(0.5)^2 + (x_1 - 0.5) - v_6 \geq 0$ and $x_1 - v_6 \leq 0$. Convex operations can be approximated arbitrarily and closely by generating a large number of tangent lines. For instance, in Fig. 6-left, the constraint $x^2 = 0$ is approximated by one secant line (connecting the endpoints of the image of x^2 in the interval) and three tangent lines.

[4] Example from [61].

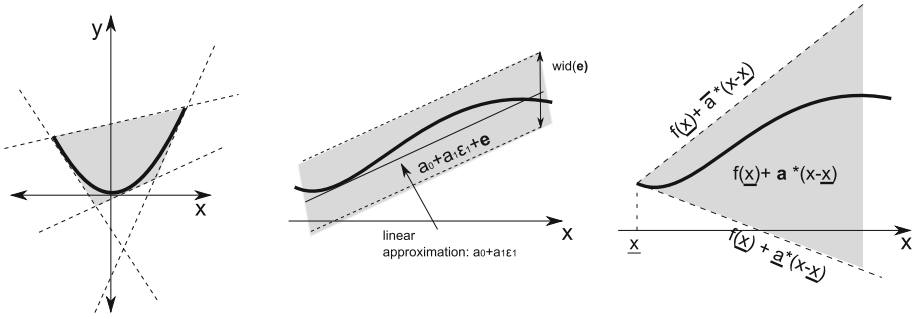

Fig. 6 Examples of linear relaxations related to a function. *Left* one linear overestimator and three linear underestimators applied to the constraint $x^2 = 0$. *Center* arithmetic affine applied to a function. Interval errors are represented by the single interval $\boldsymbol{e}$. *Right* interval Taylor form of a function f. In each case, the *gray area* corresponds to the generated polytope

In [82,86], the authors propose new affine and quadratic extensions for computing efficient and reliable bounds of functions. Extensions based on affine arithmetic are used in interval optimizers such as IBBA and IbexOpt.

Similar to interval arithmetic, computations in affine arithmetic are performed by extending primitive operators to operate on *affine forms* [27]. Affine forms are the representation of quantities in affine arithmetic. They are expressions of the following form:

$$\widehat{a} = a_0 + a_1\varepsilon_i + \cdots + a_p\varepsilon_p$$

where a_i are constants, and ε_i are unknown reals belonging to the interval $\boldsymbol{\epsilon}_i = [-1, 1]$.

Using affine arithmetic, we can construct linear relaxations of a function f over a box $\boldsymbol{x}$ while traversing the expression DAG of f in a bottom-up fashion. First, variables are replaced by their corresponding affine form. That is, each variable x_i is replaced by

$$\text{mid}(\boldsymbol{x}_i) + \frac{\text{wid}(\boldsymbol{x}_i)}{2}\varepsilon_i$$

The natural evaluation of the affine form is equal to interval $\boldsymbol{x}_i$.

Then, in each node corresponding to a linear operator (e.g., $+$ and $-$), the operator is applied to the child affine forms by simply adding or subtracting terms. As a result, we obtain a new affine form, which is attached to the node. Non-linear operators (e.g., $*$, / and unary non-linear functions) are approximated by affine operators, which add a new extra variable related to the non-linear error. For example, the multiplication of two affine forms can be approximated by the following affine form [25]:

$$\widehat{\widehat{a} \times \widehat{b}} := a_0b_0 + \sum_{i=1}^{p}(a_0b_i + a_ib_0)\varepsilon_i + \left(\sum_{i=1}^{p}|a_i| \times \sum_{i=1}^{p}|b_i|\right)\varepsilon_{p+1}.$$

Note that a new extra variable $\varepsilon_{p+1} \in [-1, 1]$ must be added.

At the end, in the root node of expression DAG, the affine arithmetic generates an interval function or, more precisely, a real linear function plus interval errors related to the nonlinear operators (see Fig. 6-center). This interval function corresponds to two linear inequalities, which approximate (overestimate and underestimate) the non-linear function f over box $\boldsymbol{x}$ (consider each variable ε_i, for $i = 1..n$, to be equivalent to $2 * \frac{x_i - \text{mid}(\boldsymbol{x}_i)}{\text{wid}(\boldsymbol{x}_i)}$).

In [4], the contractor `X-Taylor` is proposed. `X-Taylor` linearizes each constraint using an interval Taylor form of them. Unlike the classical Taylor form, which is expanded from

the midpoint of the box (see Fig. 2), `X-Taylor` uses a vertex of the box (see Fig. 6-right), thus creating a convex polytope.

For instance, if we expand the first-order Taylor of a function $f:\mathbb{R}^n \Rightarrow \mathbb{R}$ from the vertex $\underline{x} = (\underline{x_1}, \ldots, \underline{x_n})$ of a box $\boldsymbol{x}$, then we have for all $x \in \boldsymbol{x}$:

$$f(x) \in f(\underline{x}) + \sum_{i=1}^{n} \boldsymbol{a}_i (x_i - \underline{x_i}),$$

where $\boldsymbol{a}_i$ is an interval overestimation of the image of $\frac{\partial f}{\partial x_i}$ over $\boldsymbol{x}$. Replacing the interval $\boldsymbol{a}_i$ with its bounds, we obtain the following two hyperplanes enclosing the function f for all $x \in \boldsymbol{x}$:

$$f(\underline{x}) + \sum_{i=1}^{n} \underline{a_i}(x_i - \underline{x_i}) \leq f(x) \leq f(\underline{x}) + \sum_{i=1}^{n} \overline{a_i}(x_i - \overline{x_i})$$

From each vertex of box $\boldsymbol{x}$, we can analogously construct two hyperplanes enclosing the function f. Thus, we can construct 2^n linear underestimations (respectively overestimations) of f. Because creating a linear relaxation using the 2^n hyperplanes for each function is computationally expensive, several heuristics for selecting some vertices were proposed in [4]. Among them, the heuristic that randomly takes one vertex, and its opposite, offers the best numerical results. Indeed, using more than 2 vertices seems to be counterproductive in practice [4].

5.3.1 Issues and perspectives related to relaxation-based techniques

Non-rigorous algorithms apply more sophisticated relaxation-based techniques to reduce the variable domains (e.g., convexification [117], optimality-based reduction [106], reformulation-linearization technique [89,106,117], outer approximation techniques [31,76,79,87]). A main challenge is making these numerical methods mathematically rigorous. Because these techniques, in general, work with floating-point numbers, they return approximations of the real solution. Using approximations in filtering algorithms results in removing feasible regions and, possibly, the optimal solution. Some techniques have successfully been made mathematically rigorous and have been incorporated into interval solvers. In [94], Neumaier et al. present a technique to compute a safe bound of a linear program based on a standard simplex algorithm. Kearfott proposed a safe implementation of the optimality-based reduction technique in [58]; a reformulation-linearization technique is used in Icos [71]; and safe linearizations are used in several interval solvers [71,96,118].

5.4 Advanced features: adaptive mechanisms for selecting contractors

As times goes by, increasingly more contractors for interval B&B emerge. A typical set of contractors in an interval B&B consists of the following:

- propagation-based contractors: a contractor enforcing hull consistency (e.g., `HC4`, `Box` or `Mohc`), and sometimes contractors enforcing stronger consistencies (e.g., `3B`, `3BCID`).
- convex relaxation-based contractors: different alternatives depending on the convexification technique.
- interval Newton: primarily used for certifying solutions.

The contraction power of these contractors is generally not comparable. For instance, in highly non-linear and non-convex constraints, propagation-based contractors seem to be

more effective, whereas in constraints with a few nonlinear terms, convex relaxation-based contractors offer the best contraction. Mohc offers better contraction than HC4 (particularly when functions are monotonic w.r.t all or several variables) but at higher costs. Similarly, strong consistency contractors are more effective but less efficient than contractors enforcing hull consistency.

An adaptive mechanism for deciding which revision procedure to use is proposed in [6]. Through this mechanism, the algorithm decides, for each constraint, whether it applies a weak but cheap revision method (HC4-Revise) or a stronger but more expensive one (Mohc-Revise). Similarly to works in finite domain CSPs (e.g., see [114,115]), the stronger contractor is applied only when there are high probabilities of filtering the search space using this contractor. Though Mohc suffers a decrease in its contraction power, the adaptive mechanism allows the algorithm to perform better than if any of the revision algorithms is applied alone.

Another adaptive interval-based contractor is proposed in [95]. The algorithm ACID (adaptive 3BCID) partially enforces the CID consistency (similarly to SAC in finite domain CSPs). Instead of performing the procedure on the entire set of variables, ACID dynamically selects the number of variables that will be processed. The variables are sorted by decreasing influence using the SmearSumRel heuristic. In this way, the most promising variables are most likely to be processed. When the number of selected variables is 0, ACID performs a simple constraint propagation (e.g., a call to HC4). The number of processed variables is set through a learning phase. In this phase, the algorithm attempts to find the necessary number of variables to obtain most of the maximal reachable filtering. The learning phase is interleaved several times in the search to auto-adapt the value during the search. ACID is the current state-of-the-art contractor, enforcing strong consistencies used by the IbexOpt solver.

We also believe that some ideas from finite-domain CSPs can be adapted to interval-based algorithms. For instance, in [115], the authors experimentally study the behavior of constraint propagation when different local consistencies are applied. They show that in structured problems, propagations resulting in empty domains form clusters of very close calls to the propagation procedure. Considering this observation, they propose several heuristics to detect clusters to apply strong consistencies in them (high probability of pruning) while applying weak consistencies in the rest of the search (low probability of pruning). In [114], the authors use information gathered from a random probing preprocessing phase to select the propagation method that should be used on each constraint. The information provided by the random probing includes the percentage of fruitful revisions for each constraint, the local consistency responsible for each such revision and the number of value deletions caused by a given propagation method. This information is then used to select the best contractor for each constraint.

In interval B&B, it would also be interesting to identify, during the search, which constraints and/or variables have been *actively used* by the different contractors (e.g., constraints responsible for an empty box or filtered variables). Thus, each constraint/variable should be treated by its most promising related contractors.

6 Bisection methods

The bisection consists of two steps: the selection of the variable(s) to be split and the selection of the value(s) in which the box will be split. Commonly, the box is split at the midpoint of the selected interval.

Related to the variable selection, several algorithms have been proposed to dater [3,26,43,62,69,118,119]. Among the most used algorithms include round robin, largest-first and smear-based selection methods. **Round-Robin** is one of the simplest and most used methods. It does not require any information about the system. If the current k-dimensional box $\boldsymbol{x}$ was obtained by bisecting the interval $[x_i]$, then the Round-Robin method will select the variable x_j, with j= $(i+1)\ mod\ k$. The objective is to refrain from neglecting any variable. One weakness of this strategy is that a *bad* initial ordering of variables can lead to disastrous performance. **Largest-First** [90] (also known as the interval width-oriented rule [26]) simply selects the variable with the largest domain. It is based on the assumption that intervals with large diameters have a greater influence on the function evaluation. In the same manner as the Round-Robin technique, the largest-first method does not omit any variable. **Smear-based methods** [62,102] use information of the system to obtain the variable with the greatest *influence*. The influence of a variable x_i on a function f_j is computed by means of the *smear value*. Consider that the current node is associated with box $\boldsymbol{x}$; the smear value is given by

$$\text{smear}(x_i, f_j) = |\boldsymbol{a}_{ij}| * \text{wid}(\boldsymbol{x}_i)$$

where $\boldsymbol{a}_{ij}$ is an interval overestimation of the range of the partial derivative $\frac{\partial f_j}{\partial x_i}$ in $\boldsymbol{x}$.

Selection methods based on the smear value select the variable that maximizes an aggregation (e.g., sum or maximum) of this value in the whole system. A variant proposed in [118] uses a *relative smear value* instead (for more details, refer to [3]):

$$\text{smear}_{rel}(x_i, f_j) = \frac{\text{smear}(x_i, f_j)}{\sum_{k=1}^{n} \text{smear}(x_k, f_j)}$$

According to the experimentations performed in [2] (74 NCOP and 27 NCSP instances), using the relative smear value (heuristic SmearSumRel) seems to be much more robust than just using the smear value.

6.1 Issues and perspectives related to bisection methods

In finite-domain CSPs, many selection heuristics have been proposed to date (e.g., [8,12,15,16,42,112]). In general, they follow the fail first principle "to succeed, try first where you are most likely to fail". The principle states that we should choose next the variable whose success probability is smallest, to minimize the expected branch length and thereby to minimize the search cost.

In interval-based algorithms, only Smear-based heuristics (and possibly largest first) follow the fail-first principle. They select the variable that, if instantiated, will most reduce the image of the functions. This would maximize the probability that at least one constraint is unsatisfied; therefore, the new subproblem would fail. In [3], we present new heuristics based on the Smear function to better understand its behavior.

Experiments related to finite CSPs conducted by Beck et al. [8] led them to believe that the effect of heuristics on the *branching factor* of the search tree (i.e., the number of children at each node) is also crucial for the performance of the algorithm. Well-performing heuristics in finite CSPs, such as dom [42] and Brélaz [16], select the variable that minimizes the branching factor in the node, i.e., the variable with the smallest domain. Other well-known heuristics combine both principles (failing early and reducing the branching factor), e.g., dom/deg [12] and dom/wdeg [15]. The branching factor is not used in interval-based algorithms because, due to the bisection procedure, there are at most two children in each node. However, we wonder whether there is a way to successfully adapt heuristics such as dom or dom/deg to intervals.

More sophisticated heuristics in finite domains, such as dom/wdeg, *look back* in the current search to detect the most important constraints: Each time a node is discarded, the weights of constraints directly related to this discard are augmented by 1. The criterion for selecting the next variable combines two criteria: the variable with the smallest domain and the variable that appears in the most weighted constraints. Another, more recent, look-back strategy is described in [88]. The authors propose selecting the *most active* variable during the propagation. The activity of a variable is computed during the search, and it is augmented each time some value of a variable domain is filtered. Several works also perform probing, i.e., a set of random searches to retrieve information from the problem (e.g., weight for constraints, activity of variables). This information is then used to select the order in which the variables should be selected [15,88,103].

Related to interval B&B, on the one hand, we think that a concept similar to the domain size of variables could be devised (for instance, the expected number of times a variable *needs to be* bisected). Then, heuristics minimizing the branching factor in finite CSPs (such as dom or Brélaz) could be easily adapted to intervals. On the other hand, once there is a high-quality and robust heuristic (e.g., the SmearSumRel heuristic), we think that the next step will be to incorporate some type of look-back strategy, similar to the ones related above.

7 Upper bounding: issues and perspectives

Upper bounding is a crucial component of NCOP solvers. Improving the current best found cost, UB, implies a reduction of the feasible space through the constraint $f_{obj}(x) \leq UB$. Thus, NCOP solvers should put considerable effort into finding solutions with near-optimal costs. However, once a quasi-optimal solution has been found, any effort spent in finding better solutions becomes useless.

In NCOP solvers, low-cost upper-bounding techniques are generally used. They range from simply trying the midpoint point of the box [96] to more sophisticated methods, such as extracting feasible linear regions to minimize a linearization of the objective function using Simplex [5]. In this direction (i.e., cheap techniques), well-known simple iterative methods, such as the gradient-descent [19] or Frank-Wolfe [36,52] methods, can help to find better solutions without much additional effort.

To validate a candidate solution x_c, interval optimizers perform a feasibility test. IBBA and Ibex-opt evaluate, using interval arithmetic, x_c in each function to validate the related constraints. Equality constraints $f_j(x) = 0$ are relaxed ($-\epsilon_{eq} \leq f_j(x) = 0 \leq \epsilon_{eq}$) to be validated. A few solvers work rigorously without relaxing equalities (e.g., GlobSol, Icos). They validate a tiny box, i.e., they guarantee that in a tiny box $\boldsymbol{x}_c$, there exists a solution for the original problem. An upper-bound cost of this solution is computed by evaluating $\boldsymbol{x}_c$ in the objective function with interval arithmetic. The incorporation of this type of certification technique is required in current state-of-the-art optimizers. In a recent work [60], Kearfott proposes an alternative technique that begins with an *approximately feasible point* (i.e., a point feasible in the relaxed system but unfeasible in the original one) and tries to make it feasible by perturbing it. The technique makes use of Gauss–Newton steps.

Global optimizers from the mathematical programming community generally resort to local minimization using sophisticated algorithms, e.g., Lagrangian relaxation-based methods in BARON [107] or the algorithms SNOPT [39] and CONOPT [29] in ANTIGONE [89]. We believe that new strategies should incorporate this type of sophisticated *but expensive* upper-bounding technique for finding solutions. Calling these methods in each node of the

search tree may be counterproductive because of their expense. Thus, to minimize the overhead, they should incorporate mechanisms for controlling the effort. For instance, they could apply an expensive local minimization technique *only* when a new upper bound has been found by the current techniques or only in the root node. Indicators (e.g., estimates of the proximity of the node to a minimizer point [77]) could also be used to decide whether a node should be exploited.

Consider the following experiment. An NCOP instance π can be transformed into an unfeasible NCSP instance π_u by adding the constraint $f_{obj}(x) < x^*$, where x^* is the lower bound of the optimal solution found by solving π. Solving π_u is almost equivalent (in time and the size of the tree search) to solving π using a super-fast upper-bounding method, which finds an ϵ-optimal solution in the first iteration of the NCOP algorithm. Comparing the CPU time spent for solving π and π_u, we can see the highest gain we could expect by improving the upper-bounding method.

8 Selection of the next box: issues and perspectives

When we address NCSPs, the selection of the next box to be treated is not a problem. Because the algorithm requires finding all solutions of the NCSP, it must treat all of the expanded boxes. Thus, to minimize the memory complexity, algorithms usually use a stack for storing the expanded nodes, performing a depth-first search (memory complexity O(1)). A different strategy is presented in [24]. The authors propose an interval-based B&B solver with the objective of finding potential solutions in different areas of the search space in the early stages of the exploration. Its algorithm first executes a classical depth-first search until a solution is found. Then, a new search strategy is used. The box maximizing the distance to the nearest solution found so far is selected next.

In contrast, the selection of the next box is crucial when we address NCOPs. We should next select the box that is most likely to improve the upper bound by the highest value, thus achieving the largest reduction of the feasible space (note the relation with the upper bounding). Thus, for the same reason that the upper bounding is useless once a near-optimal solution is found, the criteria for selecting the next box become irrelevant at that time, too.

One of the most used criteria is minLB. It consists of simply selecting the box with the lowest underestimation of the objective function ($lb = LB$). In the best case, we could find in this box a better new solution with cost lb, and the search would be finished (because $UB - LB = 0$). However, because interval methods compute large overestimations of images, this result is very improbable. Markot et al. [77] propose several interesting policies for selecting the next box. They range from the previously mentioned minLB to more sophisticated heuristics that try to select a box that (1) has an image under f and most likely contains a solution better than UB and (2) seems to have the largest feasible volume. Although these criteria have been tested only on a solver with no contractors (only a feasibility test is used to discard boxes), they show promising results.

KBFS [34] is a search strategy used in discrete-domain CSPs that generalizes the best-first search. In each iteration, KBFS expands the K best nodes instead of only 1, as the best-first strategy does. The authors show that KBFS is better than best-first when the heuristics for estimating the cost are inadmissible (otherwise, best-first is the best strategy) and have large errors. In this case, best first could fall into large subtrees with suboptimal or no solutions, whereas KBFS could escape from them and explore other possibilities. The KBFS strategy could be used in interval B&B because although minLB is an admissible heuristic, the fact

that we can update the UB in each node and thus the search space turns the best-first into a not-best strategy.

9 Conclusions

Interval B&B algorithms are commonly used to solve constraint satisfaction and global optimization problems. In this paper, two generic interval B&B solvers for treating these types of problems are described. They have important components in common: contractors and bisectors.

Newton-based contractors appear to be a good choice when the system is well constrained and the boxes are sufficiently small. Additionally, they are useful to certify the existence of a unique solution in the box. Relaxation-based contractors are not limited to well-constrained systems; however, the linear relaxations may lose important information related to the original problem. CP contractors are efficient for filtering larger boxes, working directly on the original (nonlinear) system of constraints. However, they have a reduced sight of the entire system; therefore, they suffer from the locality problem. Most recent interval-based libraries (e.g., Ibex [23], Icos [70]) allow one to combine these three approaches in one solver strategy, thus making the results more robust. Related to the bisection methods, only a few works have been proposed thus far, with the smear-based methods (in particular, the SmearSumRel heuristic) representing the most promising ones.

The dependency problem has been an obstacle to interval-based solvers from the very beginning. Although there is still work to do, we think that recent sophisticated CP contractors address the problem in a *satisfactory way*. We observed, however, several other issues that should be taken into account to improve the solver performance in the short term:

- Treating the locality problem with constraint propagation-oriented techniques, such as the linear combination of constraints and common subexpression elimination.
- Incorporating successful contraction techniques from non-rigorous algorithms, such as the reformulation linearization-technique, optimization-based reduction, convexification, and the use of constraints related to optimality conditions oriented to reduce the cluster effect. Most of these techniques must be adapted to intervals by making them mathematically rigorous.
- Incorporating adaptive mechanisms for selecting contractors. The idea here is to increase the contraction effort when it is most likely to filter domains.
- Incorporating look-back strategies to improve the variable selection heuristics.
- Improving the upper bounding techniques in optimization problems by incorporating more sophisticated (and possibly expensive) local search algorithms. To minimize the overhead, they should also incorporate mechanisms to control the effort.
- Improving the next box selection in optimization problems. For instance, combining intensification and diversification criteria in a KBFS-like [34] strategy.

Acknowledgments This work is supported by the Fondecyt Project 1120781.

References

1. Araya, I., Neveu, B., Trombettoni, G.: Exploiting common subexpressions in numerical CSPs. In: Principles and Practice of Constraint Programming (CP 2008), pp. 342–357. Springer (2008)

2. Araya, I., Neveu, B., Trombettoni, G.: An interval extension based on occurrence grouping. Computing **94**(2–4), 173–188 (2012)
3. Araya, I., Reyes, V., Oreallana, C.: More smear-based variable selection heuristics for NCSPs. In: International Conference on Tools with Artificial Intelligence (ICTAI 2013), pp. 1004–1011. IEEE (2013)
4. Araya, I., Trombettoni, G., Neveu, B.: A contractor based on convex interval Taylor. In: Proceedings of CPAIOR, LNCS 7298, pp. 1–16 (2012)
5. Araya, I., Trombettoni, G., Neveu, B., Chabert, G.: Upper bounding in inner regions for global optimization under inequality constraints. J. Glob. Optim. **60**, 145–164 (2014). doi:10.1007/s10898-014-0145-7
6. Araya, I., Trombettoni, G., Neveu, B., et al.: Exploiting monotonicity in interval constraint propagation. In: AAAI (2010)
7. Baharev, A., Achterberg, T., Rév, E.: Computation of an extractive distillation column with affine arithmetic. AIChE J. **55**(7), 1695–1704 (2009)
8. Beck, J.C., Prosser, P., Wallace, R.J.: Trying again to fail-first. In: Recent Advances in Constraints, pp. 41–55. Springer (2005)
9. Belotti, P.: Couenne, a users manual (2013). http://www.coin-or.org/Couenne/
10. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)
11. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising hull and box consistency. In: International Conference on Logic Programming. Citeseer (1999)
12. Bessiere, C., Régin, J.C.: MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In: Principles and Practice of Constraint Programming (CP96), pp. 61–75. Springer (1996)
13. Bliek, C.: Computer methods for design automation. Ph.D. thesis, Massachusetts Institute of Technology (1992)
14. Bournez, O., Maler, O., Pnueli, A.: Orthogonal polyhedra: Representation and computation. In: Hybrid Systems: Computation and Control, pp. 46–60. Springer (1999)
15. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI, vol. 16, p. 146 (2004)
16. Brélaz, D.: New methods to color the vertices of a graph. Commun. ACM **22**(4), 251–256 (1979)
17. Carrizosa, E., Hansen, P., Messine, F.: Improving interval analysis bounds by translations. J. Glob. Optim. **29**(2), 157–172 (2004)
18. Casado, L., Martinez, J., García, I.: Experiments with a new selection criterion in a fast interval optimization algorithm. J. Glob. Optim. **19**(3), 247–264 (2001)
19. Cauchy, A.: Méthode générale pour la résolution des systemes déquations simultanées. C. R. Sci. Paris **25**(1847), 536–538 (1847)
20. Ceberio, M., Granvilliers, L.: Horner's rule for interval evaluation revisited. Computing **69**(1), 51–81 (2002)
21. Ceberio, M., Granvilliers, L.: Solving nonlinear equations by abstraction, Gaussian elimination, and interval methods. In: Frontiers of Combining Systems, pp. 117–131. Springer (2002)
22. Ceberio, M., Kreinovich, V.: Greedy algorithms for optimizing multivariate Horner schemes. ACM SIGSAM Bull. **38**(1), 8–15 (2004)
23. Chabert, G., Jaulin, L.: Contractor programming. Artif. Intell. **173**(11), 1079–1100 (2009)
24. Chenouard, R., Goldsztejn, A., Jermann, C., et al.: Search strategies for an anytime usage of the branch and prune algorithm. In: IJCAI, pp. 468–473 (2009)
25. Comba, J., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proceedings of SIBGRAPI'93—VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pp. 9–18 (1993)
26. Csendes, T., Ratz, D.: Subdivision direction selection in interval methods for global optimization. SIAM J. Numer. Anal. **34**(3), 922–938 (1997)
27. De Figueiredo, L.H., Stolfi, J.: Affine arithmetic: concepts and applications. Numer. Algorithms **37**(1–4), 147–158 (2004)
28. Demidovitch, B., Maron, I., Polonski, V.: Eléments de calcul numérique. Mir (1973)
29. Drud, A.S.: CONOPT: a large-scale GRG code. ORSA J. Comput. **6**(2), 207–216 (1994)
30. Du, K., Kearfott, R.B.: The cluster problem in multivariate global optimization. J. Glob. Optim. **5**(3), 253–265 (1994)
31. Duran, M.A., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math. Program. **36**(3), 307–339 (1986)
32. Eldon, H., William, W.: Global optimization using interval analysis (1992)
33. Faltings, B.V., Lottaz, C., et al.: Collaborative design using solution spaces (2000)

34. Felner, A., Kraus, S., Korf, R.E.: KBFS: K-best-first search. Ann. Math. Artif. Intell. **39**(1–2), 19–39 (2003)
35. Floudas, C.A., Pardalos, P.M.: Encyclopedia of Optimization, vol. 1. Springer Science & Business Media, Berlin (2008)
36. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Nav. Res. Logist. Q. **3**(1–2), 95–110 (1956)
37. Frommer, A., Lang, B.: Existence tests for solutions of nonlinear equations using Borsuk's theorem. SIAM J. Numer. Anal. **43**(3), 1348–1361 (2005)
38. Fünfzig, C., Michelucci, D., Foufou, S.: Nonlinear systems solver in floating-point arithmetic using LP reduction. In: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, pp. 123–134. ACM (2009)
39. Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: an SQP algorithm for large-scale constrained optimization. SIAM J. Optim. **12**(4), 979–1006 (2002)
40. Goldsztejn, A., Granvilliers, L.: A new framework for sharp and efficient resolution of NCSP with manifolds of solutions. Constraints **15**(2), 190–212 (2010)
41. Goldsztejn, A., Lebbah, Y., Michel, C., Rueher, M.: Revisiting the upper bounding process in a safe branch and bound algorithm. In: Principles and Practice of Constraint Programming (CP 2008), pp. 598–602. Springer (2008)
42. Golomb, S.W., Baumert, L.D.: Backtrack programming. J. ACM (JACM) **12**(4), 516–524 (1965)
43. Granvilliers, L.: Adaptive bisection of numerical CSPs. In: Principles and Practice of Constraint Programming (2012), pp. 290–298. Springer (2012)
44. Granvilliers, L., Goldsztejn, A.: A branch-and-bound algorithm for unconstrained global optimization. In: Proceedings of the 14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN) (2010)
45. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: Numerical toolbox for verified computing I (1993)
46. Hansen, E.: Interval arithmetic in matrix computations, Part I. J Soc Ind. Appl. Math. Ser. B Numer. Anal. **2**(2), 308–320 (1965)
47. Hansen, E.: Global optimization using interval analysis: the multi-dimensional case. Numer. Math. **34**(3), 247–270 (1980)
48. Hansen, E.: Bounding the solution of interval linear equations. SIAM J. Numer. Anal. **29**(5), 1493–1503 (1992)
49. Hansen, E.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (1992)
50. Hansen, E., Walster, G.W.: Global Optimization Using Interval Analysis: Revised and Expanded, vol. 264. CRC Press, Boca Raton (2003)
51. Ishii, D., Goldsztejn, A., Jermann, C.: Interval-based projection method for under-constrained numerical systems. Constraints **17**(4), 432–460 (2012)
52. Jaggi, M.: Revisiting Frank–Wolfe: projection-free sparse convex optimization. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp. 427–435 (2013)
53. Jaulin, L.: Localization of an underwater robot using interval constraint propagation. In: Principles and Practice of Constraint Programming (CP 2006), pp. 244–255. Springer (2006)
54. John, F.: Extremum problems with inequalities as subsidiary conditions. In: Studies and Essays Presented to R. Courant on his 60th Birthday (Jan. 8, 1948), pp. 187–204. Interscience, New York (1948)
55. Karush, W.: Minima of functions of several variables with inequalities as side constraints. Ph.D. thesis, Masters thesis, Dept. of Mathematics, University of Chicago (1939)
56. Kearfott, R.B.: Preconditioners for the interval Gauss–Seidel method. SIAM J. Numer. Anal. **27**(3), 804–822 (1990)
57. Kearfott, R.B.: An interval branch and bound algorithm for bound constrained optimization problems. J. Glob. Optim. **2**(3), 259–280 (1992)
58. Kearfott, R.B.: Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. Optim. Methods Softw. **21**(5), 715–731 (2006)
59. Kearfott, R.B.: GlobSol user guide. Optim. Methods Softw. **24**(4–5), 687–708 (2009)
60. Kearfott, R.B.: On rigorous upper bounds to a global optimum. J. Glob. Optim. **59**(2–3), 459–476 (2014)
61. Kearfott, R.B., Hongthong, S.: Validated linear relaxations and preprocessing: some experiments. SIAM J. Optim. **16**(2), 418–433 (2005)
62. Kearfott, R.B., Novoa III, M.: Algorithm 681: INTBIS, a portable interval Newton/bisection package. ACM Trans. Math. Softw. (TOMS) **16**(2), 152–157 (1990)
63. Kearfott, R.B., Walster, G.W.: Symbolic preconditioning with Taylor models: some examples. Reliab. Comput. **8**(6), 453–468 (2002)
64. Kieffer, M.: Distributed bounded-error parameter and state estimation in networks of sensors. In: Numerical Validation in Current Hardware Architectures, pp. 189–202. Springer (2009)

65. Kieffer, M., Walter, E.: Centralized and distributed source localization by a network of sensors using guaranteed set estimation. In: 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings, vol. 4, pp. IV–IV. IEEE (2006)
66. Kolev, L.V.: Use of interval slopes for the irrational part of factorable functions. Reliab. Comput. **3**(1), 83–93 (1997)
67. Krawczyk, R.: Newton-algorithmen zur bestimmung von nullstellen mit fehlerschranken. Computing **4**(3), 187–201 (1969)
68. Kueviakoe, I., Lambert, A., Tarroux, P.: Comparison of interval constraint propagation algorithms for vehicle localization. J. Softw. Eng. Appl. **5**, 157 (2013)
69. Lagouanelle, J.L., Soubry, G.: Optimal multisections in interval branch-and-bound methods of global optimization. J. Glob. Optim. **30**(1), 23–38 (2004)
70. Lebbah, Y.: Icos: a branch and bound based solver for rigorous global optimization. Optim. Methods Softw. **24**(4–5), 709–726 (2009)
71. Lebbah, Y., Michel, C., Rueher, M.: An efficient and safe framework for solving optimization problems. J. Comput. Appl. Math. **199**(2), 372–377 (2007)
72. Lhomme, O.: Consistency techniques for numeric CSPs. In: IJCAI, vol. 93, pp. 232–238. Citeseer (1993)
73. Liberti, L.: Writing global optimization software. In: Global Optimization, pp. 211–262. Springer (2006)
74. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977)
75. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. Int. J. Pure Appl. Math. **4**(4), 379–456 (2003)
76. Maranas, C.D., Floudas, C.A.: Finding all solutions of nonlinearly constrained systems of equations. J. Glob. Optim. **7**(2), 143–182 (1995)
77. Markót, M.C., Fernandez, J., Casado, L.G., Csendes, T.: New interval methods for constrained global optimization. Math. Program. **106**(2), 287–318 (2006)
78. Markót, M.C., Schichl, H.: Bound constrained interval global optimization in the COCONUT environment. J. Glob. Optim. **60**(4), 751–776 (2014)
79. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: part I—convex underestimating problems. Math. Program. **10**(1), 147–175 (1976)
80. Merlet, J.P.: Optimal design for the micro parallel robot MIPS. In: IEEE International Conference on Robotics and Automation, 2002. Proceedings of ICRA'02, vol. 2, pp. 1149–1154. IEEE (2002)
81. Merlet, J.P.: Interval analysis for certified numerical solution of problems in robotics. Int. J. Appl. Math. Comput. Sci. **19**(3), 399–412 (2009)
82. Messine, F.: Extentions of affine arithmetic: application to unconstrained global optimization. J. Univers. Comput. Sci. **8**(11), 992–1015 (2002)
83. Messine, F.: Deterministic global optimization using interval constraint propagation techniques. RAIRO Oper. Res. **38**(04), 277–293 (2004)
84. Messine, F.: A deterministic global optimization algorithm for design problems. In: Essays and Surveys in Global Optimization, pp. 267–294. Springer (2005)
85. Messine, F., Nogarede, B., Lagouanelle, J.L.: Optimal design of electromechanical actuators: a new method based on global optimization. IEEE Trans. Magn. **34**(1), 299–308 (1998)
86. Messine, F., Touhami, A.: A general reliable quadratic form: an extension of affine arithmetic. Reliab. Comput. **12**(3), 171–192 (2006)
87. Meyer, C.A., Floudas, C.A.: Convex envelopes for edge-concave functions. Math. Program. **103**(2), 207–224 (2005)
88. Michel, L., Van Hentenryck, P.: Activity-based search for black-box constraint programming solvers. In: Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, pp. 228–243. Springer (2012)
89. Misener, R., Floudas, C.A.: ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. J. Glob. Optim. **59**(2–3), 503–526 (2013)
90. Moore, R.: Interval Analysis, vol. 60 (1966)
91. Mourad, F., Snoussi, H., Abdallah, F., Richard, C.: Anchor-based localization via interval analysis for mobile ad-hoc sensor networks. IEEE Trans. Signal Process. **57**(8), 3226–3239 (2009)
92. Nataraj, P., Patil, M.D.: Reliable and robust automated synthesis of QFT controller for nonlinear magnetic levitation system using interval constraint satisfaction techniques. In: Constraint Programming and Decision Making, pp. 131–135. Springer (2014)
93. Nataraj, P., Tharewal, S.: An interval analysis algorithm for automated controller synthesis in QFT designs. J. Dyn. Syst. Meas. Contr. **129**(3), 311–321 (2007)
94. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. **99**(2), 283–296 (2004)

95. Neveu, B., Trombettoni, G., et al.: Adaptive constructive interval disjunction. In: International Conference on Tools with Artificial Intelligence (ICTAI), pp. 900–906 (2013)
96. Ninin, J., Messine, F., Hansen, P.: A reliable affine relaxation method for global optimization. 4OR (2014). doi:10.1007/s10288-014-0269-0
97. Ortega, J.M., Rheinboldt, W.C.: Iterative Solution of Nonlinear Equations in Several Variables, vol. 30. Siam, Philadelphia (2000)
98. Patil, M.D., Nataraj, P.: QFT prefilter design for multivariable systems using interval constraint satisfaction technique. J. Control Theory Appl. **11**(4), 529–537 (2013)
99. Ramdani, N., Meslem, N., Candau, Y.: Reachability of uncertain nonlinear systems using a nonlinear hybridization. In: Hybrid Systems: Computation and Control, pp. 415–428. Springer, Berlin (2008)
100. Ramdani, N., Nedialkov, N.S.: Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. Nonlinear Anal. Hybrid Syst. **5**(2), 149–162 (2011)
101. Ratschek, H., Rokne, J.: New Computer Methods for Global Optimization. Horwood, Chichester (1988)
102. Ratz, D.: Automatische ergebnisveri kation bei globalen optimierungsproblemen. Ph.D. thesis, Dissertation, Universit at Karlsruhe (1992)
103. Refalo, P.: Impact-based search strategies for constraint programming. In: Principles and Practice of Constraint Programming (CP 2004), pp. 557–571. Springer (2004)
104. Reynet, O., Voisin, O., Jaulin, L.: Anchor-based localization using distributed interval contractors (2011)
105. Roy, J.M.: Singularities in Deterministic Global Optimization. University of Louisiana at Lafayette (2010)
106. Ryoo, H.S., Sahinidis, N.V.: A branch-and-reduce approach to global optimization. J. Glob. Optim. **8**(2), 107–138 (1996)
107. Sahinidis, N.V.: BARON: a general purpose global optimization software package. J. Glob. Optim. **8**(2), 201–205 (1996)
108. Sam-Haroud, D., Faltings, B.: Consistency techniques for continuous constraints. Constraints **1**(1–2), 85–118 (1996)
109. Schichl, H., Neumaier, A.: Exclusion regions for systems of equations. SIAM J. Numer. Anal. **42**(1), 383–408 (2004)
110. Schichl, H., Neumaier, A.: Interval analysis on directed acyclic graphs for global optimization. J. Glob. Optim. **33**(4), 541–562 (2005)
111. Shectman, J.P., Sahinidis, N.V.: A finite algorithm for global minimization of separable concave programs. J. Glob. Optim. **12**(1), 1–36 (1998)
112. Smith, B.M., Grant, S.A.: Trying harder to fail first. Research report series/University of Leeds, School of Computer Studies LU SCS RR (1997)
113. Soares, R.D.P.: Finding all real solutions of nonlinear systems of equations with discontinuities by a modified affine arithmetic. Comput. Chem. Eng. **48**, 48–57 (2013)
114. Stamatatos, E., Stergiou, K.: Learning how to propagate using random probing. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 263–278. Springer (2009)
115. Stergiou, K.: Heuristics for dynamically adapting propagation in constraint satisfaction problems. AI Commun. **22**(3), 125–141 (2009)
116. Tapia, R.: The Kantorovich theorem for Newton's method. Am. Math. Mon. **78**(4), 389–392 (1971)
117. Tawarmalani, M., Sahinidis, N.V.: Convexification and Global Optimization in Continuous and Mixed-integer Nonlinear Programming: Theory, Algorithms, Software, and Applications, vol. 65. Springer, Berlin (2002)
118. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner regions and interval linearizations for global optimization. In: AAAI, pp. 99–104 (2011)
119. Trombettoni, G., Chabert, G.: Constructive interval disjunction. In: Principles and Practice of Constraint Programming (CP 2007), pp. 635–650. Springer (2007)
120. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: A Modeling Language for Global Optimization. MIT Press, Cambridge (2003)
121. Vu, X.H., Sam-Haroud, D., Faltings, B.: Combining multiple inclusion representations in numerical constraint propagation. In: International Conference on Tools with Artificial Intelligence (ICTAI 2004), pp. 458–467. IEEE (2004)
122. Vu, X.H., Sam-Haroud, D., Silaghi, M.C.: Approximation techniques for non-linear problems with continuum of solutions. In: Abstraction, Reformulation, and Approximation, pp. 224–241. Springer (2002)
123. Vu, X.H., Schichl, H., Sam-Haroud, D.: Using directed acyclic graphs to coordinate propagation and search for numerical constraint satisfaction problems. In: International Conference on Tools with Artificial Intelligence (ICTAI 2004), pp. 72–81. IEEE (2004)

124. Yamamura, K., Kawata, H., Tokue, A.: Interval solution of nonlinear equations using linear programming. BIT Numer. Math. **38**(1), 186–199 (1998)
125. Yannou, B., Simpson, T.W., Barton, R.R.: Towards a conceptual design explorer using metamodeling approaches and constraint programming. In: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, pp. 605–614. American Society of Mechanical Engineers (2003)