A Box-Consistency Contractor Based on Extremal Functions

Gilles Trombettoni, Yves Papegay, Gilles Chabert, Odile Pourtallier

COPRIN INRIA, Université Nice–Sophia, LINA Ecole des Mines de Nantes
{`Gilles.Trombettoni,Yves.Papegay,Odile.Pourtallier`}`@sophia.inria.fr`,
`Gilles.Chabert@emn.fr`

Abstract. Interval-based methods can approximate all the real solutions of a system of equations and inequalities. The `Box` interval constraint propagation algorithm enforces *Box consistency.* Its main procedure `BoxNarrow` handles one function f corresponding to the revised constraint, and one variable x, replacing the other variables of f by their current intervals. This paper proposes an improved `BoxNarrow` procedure for narrowing the domain of x when f respects certain conditions. In particular, these conditions are fulfilled when f is polynomial. f is first symbolically rewritten into a new form g. A narrowing step is then run on the non-interval *extremal functions* that enclose the interval function g. The corresponding algorithm is described and validated on several numerical constraint systems.

1 Motivation

Interval-based solvers can solve systems of numerical constraints (i.e., nonlinear equations or inequalities over the reals). Their reliability and increasing performance make them applicable to various domains such as robotics design and kinematics [9], proofs of conjectures [12], robust global optimization [7, 11] and bounded-error parameter estimation [6].

Two main types of *contraction algorithms* allow solvers to filter variable domains, *i.e.*, to reduce the intervals of each variable, without loss of solutions of the system: interval (numerical) analysis methods, like *Interval Newton* [10], and constraint propagation algorithms from constraint programming. The `HC4` and `Box` algorithms [2, 14] are very often used in solving strategies. They perform a propagation loop and filter the variable domains with a specific *revise* procedure (called `HC4-Revise` and `BoxNarrow`) handling the constraints individually. For every pair (c, x) in the system, where c is the numerical constraint $f(x, y_1, \dots, y_{k-1}) = 0$, the `BoxNarrow` contraction procedure is applied to x by considering the uni-variate constraint: $f_{[Y]}(x) = f(x, [y_1], \dots, [y_{k-1}]) = 0$. That is, $f_{[Y]}$ is a function where each variable $y_i \in Y = \{y_1, \dots, y_{k-1}\}$ of f has been replaced by its interval of variation. The important point is that $f_{[Y]}$ is an interval function: to any $x \in \mathbb{R}$, $f_{[Y]}(x)$ is an interval. Thus, the iterative process run by `BoxNarrow` may be very slow in some cases. The main idea of `PolyBox` is to work with two non-interval functions instead of $f_{[Y]}$. These non-interval functions are obtained by a symbolic manipulation preprocessing that rewrites

f into a new form g for which the two *extremal functions* enclosing $g_{[Y]}$ can be easily extracted. Then, during constraint propagation, `PolyBoxRevise` calls `BoxNarrow` on the two extremal functions of $g_{[Y]}$. This implies a faster contraction. In addition, when $g_{[Y]}$ is a low-degree polynomial, the computation of the new bounds of $[x]$ follows simple evaluations using the real roots of the extremal functions identified analytically.

2 Background

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

Definition 1 (Basic definitions, notations)
$\mathbb{IR}$ *denotes the set of* **intervals** $[v] = [a, b] \subset \mathbb{R}$ *where* a*, also denoted* $\underline{v}$*, and* b*, also denoted* $\overline{v}$*, are floating-point numbers.* $\overline{v} - \underline{v}$ *is the* **size** *of* $[v]$*.*
An **interval vector***, or* **box***,* $[V] = ([v_1], \ldots, [v_n])$ *represents the Cartesian product* $[v_1] \times \ldots \times [v_n]$*. Its size is the maximal size of its components* $[v_i], i = 1, \ldots, n$*.*

Interval arithmetic has been introduced to extend the real arithmetic to intervals [10]. For instance, we have straightforwardly $[v_1]+[v_2] = [\underline{v_1}+\underline{v_2}, \overline{v_1}+\overline{v_2}]$. This allows us to extend real valued functions to intervals. Such an **extension** must be defined so as to be conservative, i.e., $\forall V \in \mathbb{R}^k \;\; f(V) = [f](V)$ and $\forall [V] \in \mathbb{IR}^k \;\; [f]([V]) \supseteq \{f(V),\ V \in [V]\}$.

The **natural extension** $[f]_N$ of a real function f replaces arithmetic over the reals by interval arithmetic. Consider for instance the real function $f(x_1, x_2) = x_1^2 - 2x_1x_2 + x_2^2$. The natural extension $[f]_N$ from $\mathbb{IR}^n$ to $\mathbb{IR}$ is defined by $[f]_N([x_1], [x_2]) = [x_1]^2 - 2[x_1][x_2] + [x_2]^2$. Evaluated on the intervals $[x_1] = [x_2] = [0, 1]$, we obtain $[f]_N([x_1], [x_2]) = [-2, 2]$. Note that the natural extension of f depends upon its symbolic expression, and consequently is not unique. As a matter of fact, f may also be rewritten as $(x_1-x_2)^2$ and yields the natural extension $([x_1] - [x_2])^2$. Note that $([x_1] - [x_2])^2 = [0, 1] = \{f(x_1, x_2),\ x_1 \in [x_1],\ x_2 \in [x_2]\} \subset [f]_N([0, 1], [0, 1])$. This illustrates the **dependency problem** which is a major concern in interval arithmetic. f has **multiple occurrences** of variables that are handled as different variables by interval arithmetic. In general, the dependency problem implies an overestimation of the interval image. It renders NP-hard the problem of finding the optimal interval image of a polynomial [8]. This raises the need to symbolic manipulations of expression before calculations so as to reduce overestimation.

The `PolyBox` algorithm presented in this paper aims at solving nonlinear systems of constraints or Numerical CSPs. An **NCSP** $P = (V, C, [V])$ contains a set of constraints C, a set V of n variables with domains $[V] \in \mathbb{IR}^n$. A solution $S \in [V]$ to P satisfies all the constraints in C. To approximate all the solutions of an NCSP with interval-based techniques, the solving process starts from an initial box representing the search space and builds a search tree, following a *Branch & Contract* scheme. A *Branching* operation **bisects** the current box on one dimension (variable), generating two sub-boxes. At each node of the search tree, **contraction/filtering** algorithms improve the bounds of the current box with no loss of solutions. The process terminates with boxes of size smaller than a given positive ω.

The constraint programming community proposes constraint propagation algorithms that perform a propagation loop like AC3. Contracting optimally a box w.r.t. an individual constraint is referred to as **hull-consistency** problem. Similarly to the optimal interval image computation, due to the dependency problem, hull-consistency is not tractable. The main procedure of our algorithm is compared to two state-of-the-art *revise* algorithms that handle the constraints individually. `HC4-Revise` [2] is known to achieve the hull-consistency of constraints having *no* variable with multiple occurrences, provided that the function[1] is continuous. It traverses twice the tree representing the mathematical expression of the constraint for narrowing all the involved variable intervals. `BoxNarrow` [2, 14] is stronger than `HC4-Revise` [4] and can enforce hull-consistency of a constraint when it contains *one* variable with multiple occurrences. In the general case, it enforces the *Box-consistency* property [2].

Definition 2 *An NCSP* $(X, C, [X])$ *is* **box-consistent** *if every pair* (c, x)*,* $c \in C$*,* $x \in X$ *is box-consistent. Consider the pair* (c, x)*, where the constraint is described by* $c : f(x, y_1, \ldots, y_{k-1}) = 0$*,* $f : \mathbb{R}^k \to \mathbb{R}$*, and the univariate interval function* $f_{[Y]}(x) = f(x, [y_1], \ldots, [y_{k-1}])$*. The pair* (c, x) *is box-consistent (with respect to the natural extension* $[f]_N$*) on the domain* $[x] = [\underline{x}, \overline{x}]$*, if:*
$0 \in [f_{[Y]}]_N([\underline{x}, +])$ *and* $0 \in [f_{[Y]}]_N([-, \overline{x}])$*, where* $[\underline{x}, +]$ *and* $[-, \overline{x}]$ *denote intervals of size one u.l.p.*[2] *at the bounds of* $[x]$*.*

In practice, for every pair (f, x), starting with an interval $[x]$, the `BoxNarrow` procedure returns a reduced interval $[x'] \subseteq [x]$ such that $[\underline{x'}, +]$ (*resp.* $[-, \overline{x'}]$) is the smallest (*resp.* largest) ϵ-solution[3] of the equation $f_{[Y]}(x) = 0$. Existing procedures use a *shaving* principle to narrow $[x]$: "Slices" $[\underline{x}, \underline{x}+\eta]$ (*resp.* $[\overline{x}-\eta, \overline{x}]$) are discarded from $[x]$ if $0 \notin [f_{[Y]}]_N([\underline{x}, \underline{x}+\eta])$ (*resp.* $0 \notin [f_{[Y]}]_N([\overline{x}-\eta, \overline{x}])$). This test sometimes uses a uni-variate interval Newton procedure.

Figure 1 illustrates that $f_{[Y]}$ is an interval function. It also shows the steps followed by `BoxNarrow`. The top (resp. the bottom) side of the figure details the "dichotomic" work performed by `LeftNarrow` (resp. `RightNarrow`) on slices/intervals of decreasing size, starting from $[x]$. For `LeftNarrow`, if the size of the current interval $[l]$ is less than or equal to 1 u.l.p. and $0 \in [f_{[Y]}]_N([l])$, then the procedure returns $[l]$. The last step 17 replaces the interval $[x]$ by the new interval $[\underline{l}, \overline{r}]$. Observe that at the end of `RightNarrow` (step 16), $[r]$ does not contain any zero of the function but an ϵ-zero. The slicing performed by `BoxNarrow` on a variable x limits the overestimation effect on x, but not on the other variables y_i if they also occur several times.

[1] along with *projection functions* used during the second top-down tree traversal...
[2] One Unit in the Last Place is the gap between two successive floating-point numbers.
[3] $x \in \mathbb{R}^n$ is an ϵ-solution of $f(x) = 0$, if $[-\epsilon, \epsilon] \cap f(x) \neq \emptyset$.

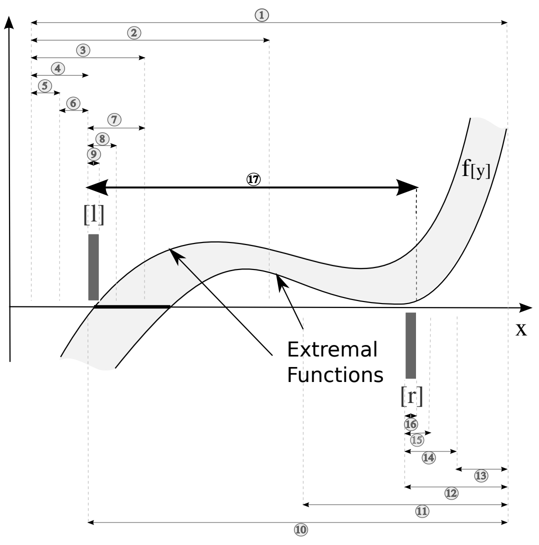

Fig. 1. The `BoxNarrow` procedure. The algorithm returns the interval computed in the step 17. (The additional contraction performed by Newton is not considered.)

3 Description of the `PolyBoxRevise` procedure

The aim of our new `PolyBoxRevise` procedure is to limit the overestimation due to multiple occurrences of variables y_i and to speed up the iterative narrowing process introduced above. Before solving the system, in a preprocessing phase, for every pair (c, x) given by $c : f(x, y_1, \ldots, y_{k-1}) = 0$, we use symbolic manipulation to rewrite f into a new form $g(x, y_1, \ldots, y_{k-1})$. This preprocessing allows the `PolyBoxRevise` procedure to rapidly extract, during constraint propagation, non-interval *extremal functions* that enclose $g_{[Y]}$, before contracting $[x]$.

Symbolic manipulation and extremal functions

For any pair (c, x) and any box $([x], [y_1], \ldots, [y_{k-1}])$, the aim would be to extract the *optimal* extremal real (i.e., non-interval) functions $\underline{h}(x)$ and $\overline{h}(x)$ defined by $\underline{h}(x) = \min_{y_i \in [y_i]} f(x, y_1, \ldots, y_{k-1})$ and $\overline{h}(x) = \max_{y_i \in [y_i]} f(x, y_1, \ldots, y_{k-1})$, $\forall x \in [x]$. Then a necessary and sufficient condition for $x \in [x]$ to satisfy c is $0 \in [\underline{h}(x), \overline{h}(x)]$. Unfortunately, it is generally not tractable to determine $\underline{h}$ and $\overline{h}$ due to the overestimation implied by the dependency problem (see Section 2 and [8]), and we have to be satisfied with functions $\underline{g}$ and $\overline{g}$ such that for any $x \in [x]$ we have $\underline{g}(x) \leq \underline{h}(x) \leq f(x, y_1, \ldots, y_{k-1}) \leq \overline{h}(x) \leq \overline{g}(x), \forall x \in [x], \forall y_i \in [y_i],\ i \in \{1, \ldots, k-1\}$. Now the test $0 \in [\underline{g}(x), \overline{g}(x)]$ is only a necessary condition. We will refer to $\underline{g}$ and $\overline{g}$ as *minimal* and *maximal* extremal functions.

The aim is to *automatically* and rapidly identify extremal functions. This can clearly not be done for any function f and we have restricted our attention to the class of functions that can be described by

$f(x, y_1, \dots, y_{k-1}) = \sum_{i=0}^{d} f_i(y_1, \dots, y_{k-1}).h_i(x)$
where d is a positive integer and h_i has a finite number of zeros in $[x]$ that can be computed exactly. In addition, the sign of $h_i(x)$ is known for any $x \in [x]$. We have in mind elementary functions such as x^i, $\log(x)$ or e^x.

We have used the symbolic manipulation tool `Mathematica` [15] for automatically identifying functions f_i and h_i, and to rewrite them in the most appropriate manner. The procedure `FullSimplify` of `Mathematica` computes automatically several possible forms for every f_i (heuristically) and selects the form minimizing a given criterion. The criterion we have specified is the number of occurrences of each variable y_i. During the solving, like `BoxNarrow`, the `PolyBoxRevise` procedure first replaces, in the new analytic form g, the variables y_i by their domains. We thus obtain $g_{[Y]}(x) = \sum_{i=0}^{d} [f_i]_N([Y]) h_i(x), \quad [Y] = ([y_1], \dots, [y_{k-1}])$. Given the box $[Y]$, the coefficients $[f_i]_N([Y])$, denoted $[c_i]$, are now numerical intervals. Due to the assumptions on h_i, the following two functions $\underline{g_{[Y]}}$ and $\overline{g_{[Y]}}$ are respectively minimal and maximal extremal functions, computed at the bounds of the interval coefficients $[c_i]$:

$$\underline{g_{[Y]}}(x) = \sum_{i=1}^{d} c_i^-(x) h_i(x) \quad \text{and} \quad \overline{g_{[Y]}}(x) = \sum_{i=1}^{d} c_i^+(x) h_i(x)$$
$$\text{with} \begin{cases} c_i^-(x) = \underline{c_i}, \;\; c_i^+(x) = \overline{c_i}, & \text{if } h_i(x) \geq 0 \\ c_i^-(x) = \overline{c_i}, \;\; c_i^+(x) = \underline{c_i}, & \text{if } h_i(x) \leq 0 \end{cases}$$

Example. Consider the function: $f(x, y_1, y_2) = (y_1 + y_2)x^2 + (2y_1y_2)x + sin(y_2)$. For the domains $[y_1] = [0.5, 1]$, $[y_2] = [1, 2]$, we have
$g_{[Y]}(x) = [1.5, 3]x^2 + [1, 4]x + [0.84147, 1]$ and then

$$\underline{g_{[0.5,1],[1,2]}}(x) = \begin{cases} 1.5x^2 + x + 0.84147, & \text{if } x \geq 0 \\ 1.5x^2 + 4x + 0.84147, & \text{if } x \leq 0 \end{cases}$$
$$\overline{g_{[0.5,1],[1,2]}}(x) = \begin{cases} 3x^2 + 4x + 1, & \text{if } x \geq 0 \\ 3x^2 + x + 1, & \text{if } x \leq 0 \end{cases}$$

Remark. $\underline{g_{[Y]}}$ and $\overline{g_{[Y]}}$ are optimal extremal functions of $g_{[Y]}$. However, although f and g are the same, the interval functions $f_{[Y]}$ and $g_{[Y]}$ are different because the replacement of the variables y_i by $[y_i]$, occurring several times in f, produce different overestimations. Hence, $\underline{g_{[Y]}}$ and $\overline{g_{[Y]}}$ constitute only approximate non-interval functions enclosing f. Also, Box-consistencies of $f_{[Y]}$ and $g_{[Y]}$ are not comparable. That is why our contractor starts by calling systematically the cheap `HC4-Revise` procedure on the initial form f before performing the process described below.

For a given $[Y]$, once the extremal functions have been determined, we proceed with the contraction part of `PolyBoxRevise` (during constraint propagation). Starting with an initial interval $[x]$, let us detail how the new and improved left bound $\underline{l}$ ($[l]$ is 1 u.l.p. large) of $[x]$ is determined. (A symetric process is performed for the right bound.) `PolyBoxRevise` first determines with which extremal function to work with. Three cases occur:

1. If $\underline{g_{[Y]}}(\underline{x}) \leq 0$ and $0 \leq \overline{g_{[Y]}}(\underline{x})$: $l = \underline{x}$ (no contraction)
2. If $\underline{g_{[Y]}}(\underline{x}) > 0$: $\underline{g_{[Y]}}$ is selected
3. If $\overline{g_{[Y]}}(\underline{x}) < 0$: $\overline{g_{[Y]}}$ is selected (situation depicted in the left side of Fig. 1)

The smallest root $[l]$ of $\overline{g_{[Y]}}(x) = 0$ in $[x]$ can now be computed using the standard `BoxNarrow` (i.e., `LeftNarrow`) procedure applied to the extremal function selected. The advantage is a faster convergence since `BoxNarrow` is run with a non-interval function.

We have implemented the polynomial case, where $h_i(x) = x^i, i = 0, \ldots, d$. In particular, when the degree d is smaller than 4, instead of using `BoxNarrow` to determine the real roots of $\overline{g_{[Y]}}(x) = 0$, we have used explicit analytical expressions of the roots. For $d = 3$, we have used the Cardano's expressions[4] of the real roots. We have adapted these symbolic methods to manage rounding errors due to floating point calculation by first replacing all the coefficients by a degenerate interval (of null size).

Finally, we have implemented a new procedure `PolyBoxRevise` based on the `Box` algorithm variant called `BC4` [2]. If $f(x, y_1, \ldots, y_{k-1})$ has a single occurrence of x, `PolyBoxRevise` calls `HC4-Revise` (like `BC4` does). Otherwise, it uses the rewritten form g (with appropriate symbolic expressions for the $f_i(y_1, \ldots, y_{k-1})$) of f produced *automatically* by the `FullSimplify` procedure of `Mathematica` [15] in the preprocessing. Four cases occur:

1. f is not polynomial w.r.t. x: the procedure calls `BoxNarrow` (or `HC4-Revise` in a hybrid version because it is less time consuming).
2. $g_{[Y]}(x)$ contains only one occurrence of x: `HC4-Revise` is applied to $g_{[Y]}(x)$.
3. $g_{[Y]}(x)$ has multiple occurrences of x and $d < 4$:
   analytic determination of the smallest root of $g_{[Y]}(x)$ in $[x]$.
4. $g_{[Y]}(x)$ has multiple occurrences of x and $d \geq 4$:
   numerical determination of the smallest root of $g_{[Y]}(x)$ in $[x]$, using `BoxNarrow`.

Remarks. The second case above can be illustrated by an equation of the system `Caprasse` (tested below): $-2x + 2txy - z + y^2 z = 0$ that our symbolic tool rewrites into: $(-2 + 2ty)x + (-1 + y2)z = 0$ (for the contraction of $[x]$ or $[z]$). Observe that the new form makes disappear the multiple occurrences of x and z. The decrease in occurrences of x and z illustrates a successful transformation leading to a gain in CPU time. An equation of the instance `6body` shows a counterproductive transformation of $5(b_1 - d_1) + 3(b_2 - d_2)(b_1 + d_1 - 2f_1) = 0$ into $b_1(5+3(b_2-d_2))+(-5+3b_2-3d_2)d_1+6(-b_2+d_2)f_1 = 0$. Indeed, for obtaining an expanded form on b_1 or d_1, the transformation increases the overestimation because of the additional occurrences of variables a_2, b_2 and d_2.

Comparison with the `Box` algorithm of Numerica

Van Hentenryck, Michel et Deville have also used extremal functions (without using this vocabulary) in their interval-based solver `Numerica` [14]. The principle

[4] G. Cardano. *Ars magna, sive de regulis algebraicis liber unus*, Nuremberg, 1545.

is introduced in one page in a technical article [13]. `Numerica` manages different forms of the handled system, and a separate constraint propagation is run on the system in an entirely expanded form for using extremal functions.

`PolyBox` follows on the contrary a scheme close to `BC4`. It manages a unique system with revise procedures adapted to every pair (f, x), which causes an overestimation smaller than the entirely expanded form used by `Numerica`. In addition, like `BC4`, `PolyBox` also uses `HC4-Revise` when x occurs only once in f. Finally, the analytic solving of low degree polynomials is added.

4 Experiments

Table 1. Results. The entries in the last four columns are the CPU time in second (first row) and the number of nodes in the search tree (second row).

Name	**#var**	**#sol**	`HC4`	`BC4`	`PolyBox--`	`PolyBox`
`Caprasse`	4	18	5.53	37.2	2.34	**2.16**
			9539	6509	2939	2939
`Yamamura1`	8	7	34.3	13.4	5.79	**2.72**
			42383	4041	2231	2231
`Extended Wood`	4	3	**0.76**	1.94	1.34	1.12
			4555	1947	3479	3479
`Broyden Banded`	20	1	> 3600	0.62	0.16	**0.09**
			?	1	1	1
`Extended Freudenstein`	20	1	> 3600	0.19	0.22	**0.11**
			?	121	121	121
`6body`	6	5	**0.58**	2.93	0.73	0.73
			4899	4797	4887	4887
`Rose`	3	18	> 3600	> 3600	**4.00**	**4.10**
			?	?	12521	12521
`Discrete Boundary`	39	1	179	29.5	41.8	**16.1**
			185,617	3279	3281	3281
`Katsura`	12	7	**102**	404	**103**	**104**
			14007	11371	13719	13719
`Eco9`	8	16	**66**	191	71	71
			132,873	125,675	131,911	131,911
`Broyden Tridiagonal`	20	2	470	495	403	**350**
			269,773	163,787	164,445	164,445
`Geneig`	6	10	3657	> 7200	3508	**3363**
			79,472,328	?	4,907,705	4,907,705

We have compared our `PolyBox` algorithm to `BC4` and `HC4`. The symbolic manipulation of all pairs (f, x) is achieved in a fraction of a second in a preprocessing by `Mathematica` [15]. All the contractors have been implemented in the free `Ibex` interval-based `C++` library [3]. To find all the solutions to the tested

NCSPs, the solving strategy bisects the variables in a round-robin way. Between two branching points in the search, constraint propagation (i.e., `PolyBox`, `HC4` or `BC4`) is performed before an interval Newton.

Among the 44 polynomial systems with isolated solutions found in COPRIN's Web page[5], we have selected the 12 instances that are solved by at least one of the 3 strategies in a time comprised between 1 second and 1 hour (on a `Pentium 3 GHz`) and have equations with multiple occurrences of the variables.

Table 1 reports interesting speedups brought by `PolyBox` on these instances. The column `PolyBox--` in the table corresponds to a variant of `PolyBox` in which the low degree polynomials are not handled analytically but by `BoxNarrow`. The additional gain brought by the analytic process is significant in only two NCSPs.

Our first results are promising, so that it should be worthwhile hybridizing `PolyBox` with other algorithms, especially those achieving the Box-consistency or a weaker form of it [1, 5]. An idea would be to keep the rewritten forms only if they are of degrees 2 and 3, and add them as global and redundant constraints in the system for improving the constraint propagation.

References

1. I. Araya, G. Trombettoni, and B. Neveu. Making Adaptive an Interval Constraint Propagation Algorithm Exploiting Monotonicity. In *Proc. CP, LNCS*, 2010.
2. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP, conf. on logic programming*, pages 230–244, 1999.
3. G. Chabert. Ibex – An Interval Based EXplorer. `www.ibex-lib.org`, 2010.
4. H. Collavizza, F. Delobel, and M. Rueher. Extending Consistent Domains of Numeric CSP. In *Proc. IJCAI*, pages 406–413, 1999.
5. A. Goldsztejn and F. Goualard. Box Consistency through Adaptive Shaving. In *Proc. ACM SAC*, pages 2049–2054, 2010.
6. L. Jaulin. Interval Constraint Propagation with Application to Bounded-error Estimation. *Automatica*, 36:1547–1552, 2000.
7. R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer, 1996.
8. V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations.* Kluwer, 1997.
9. J.-P. Merlet. Interval Analysis and Robotics. In *Symp. of Robotics Research*, 2007.
10. R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs N.J., 1966.
11. M. Rueher, A. Goldsztejn, Y. Lebbah, and C. Michel. Capabilities of Constraint Programming in Rigorous Global Optimization. In *NOLTA*, 2008.
12. W. Tucker. A Rigorous ODE Solver and Smale's 14th Problem. *Found. Comput. Math.*, 2:53–117, 2002.
13. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM J. on Num. Analysis*, 34(2), 1997.
14. P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica : A Modeling Language for Global Optimization.* MIT Press, 1997.
15. S. Wolfram. *Mathematica.* Cambridge University Press, fourth edition, 1999.

[5] See `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`. These benchmarks have been proposed by the interval community and some of them correspond to real problems.