An Interval Constraint Propagation Algorithm Exploiting Monotonicity

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni

INRIA, University of Nice-Sophia, CERTIS, France
FirstName.Name@sophia.inria.fr

Abstract. When a function f is monotonic w.r.t. a variable x in a given box, it is well-known that the monotonicity-based interval extension of f computes a sharper image than the natural interval extension does. Indeed, the overestimation due to the variable x with multiple occurrences in f disappears. However, monotonicity has not been exploited in interval filtering/contraction algorithms for solving systems of nonlinear constraints over the reals.
We propose in this paper a new interval constraint propagation algorithm, called MOnotonic Hull Consistency (`Mohc`), that exploits monotonicity of functions. The propagation is standard, but the `Mohc-Revise` procedure, used to filter/contract the variable domains involved in an individual constraint, is novel. This revise procedure uses two main bricks for narrowing intervals of the variables involved in f. One procedure is a monotonic version of the well-known `HC4-Revise`. A second procedure performs a dichotomic process calling interval Newton iterations, close to (while less costly than) the procedure `BoxNarrow` used in the `Box` contraction algorithm.
When f is monotonic w.r.t. every variable with multiple occurrences, `Mohc` is proven to compute the optimal/sharpest box enclosing all the solutions of the constraint (hull consistency). Experiments show that `Mohc` is a relevant approach to handle constraints having *several* variables with multiple occurrences, contrarily to `HC4` and `Box`.

1 Introduction

Interval-based solvers can solve systems of numerical constraints (i.e., equations or inequalities over the reals). They are becoming useful for handling numerical CSPs encountered in dynamic systems defined in robust control or autonomous robot localization [13]. Also, novel applications emerge from various domains such as robotics design and kinematics [17], or proof of conjectures (e.g., the proof of Lorentz's strange attractors detailed in [22]).

Two main types of algorithms allow solvers to remove inconsistent values from the domains of variables. Interval Newton and related algorithms generalize to intervals standard numerical analysis methods [12, 19]. Contraction algorithms issued from constraint programming are also in the heart of interval-based solvers. The constraint propagation algorithms `HC4` [3] and `Box` [23, 3] are very often used by solvers. They perform a propagation loop and filter the variable domains (i.e., improve their bounds) with a specific *Revise* procedure (called `HC4-Revise` and `BoxNarrow`) handling the constraints individually.

In practice, `HC4-Revise` often computes an optimal box enclosing all the solutions to one constraint c when no variable appears twice in c. When one critical variable appears several times in c, `HC4-Revise` is generally *not* optimal. In this case, `BoxNarrow` is proven to compute a sharper box. The new revise algorithm presented in this paper, called `Mohc-Revise`, tries to handle the general case when *several* variables have multiple occurrences in c.

When a function f is monotonic w.r.t. to a variable x in a given box, it is well-known that the monotonicity-based interval extension of f produces no overestimation related to the multiple occurrences of x. `Mohc-Revise` exploits this property to improve contraction/filtering. Monotonicity is generally verified for a few pairs (f, x) at the beginning of the search, but can be detected for more pairs as long as one goes to the bottom of the search tree, handling smaller boxes.

After introducing notations and background in Section 2, Sections 3 and 4 describe the `Mohc-Revise` algorithm, leading to two constraint propagation variants called `LazyMohc` and `Mohc`. Section 5 details related properties. In particular, when f is monotonic w.r.t. every variable with multiple occurrences, `Mohc` (a variant in fact) is proven to compute the optimal/sharpest box enclosing all the solutions of the constraint (hull consistency property). Experiments shown in Section 6 highlight that `Mohc` is a relevant approach to handle constraints having several variables with multiple occurrences.

2 Background

The algorithms presented in this paper aim at solving systems of equations or, more generally, numerical CSPs.

Definition 1 *A* **numerical CSP** *(NCSP)* $P = (X, C, B)$ *contains a set of constraints* C *and a set* X *of* n *variables. Every variable* $x_i \in X$ *can take a real value in the interval* $[x_i]$ *and* B *is the Cartesian product (called a* **box***)* $[x_1] \times ... \times [x_n]$*. A solution of* P *is an assignment of the variables in* X *satisfying all the constraints in* C*.*

Since real numbers cannot be represented in computer architectures, note that the bounds of an interval $[x_i]$ should actually be defined as floating-point numbers. Most of the set operations can be achieved on boxes, such as inclusion and intersection. An operator `Hull` is often used to compute an outer approximation of the union of several boxes. It returns the minimal box including the input boxes.

$\underline{x}$, resp. $\overline{x}$, denotes the lower bound, resp. the upper bound, of $[x]$. `Mid`$([x])$ denotes the midpoint of $[x]$ while `Diam`$([x]) \equiv \overline{x} - \underline{x}$ denotes the diameter, or size, of the interval $[x]$.

To find 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. The tree search **bisects** the current box, that is, **splits** on one dimension (variable) the box into two sub-boxes, thus generating one choice point. At every node of the search tree, filtering (also called **contraction**) algorithms reduce the bounds of the current box. These algorithms comprise

interval Newton algorithms issued from the numerical analysis community [12, 20] along with contraction algorithms issued from the constraint programming community. The process terminates with **atomic boxes** of size at most ω on every dimension.

The contraction algorithm presented in this paper proposes new procedures. They are adaptations of the classical `HC4-Revise` [3] and `BoxNarrow` [23, 3] procedures that take advantage of the monotonicity of functions.

The `HC4` algorithm performs an AC3-like propagation loop. Its revise procedure, called `HC4-Revise`, traverses twice the tree representing the mathematical expression of the constraint for narrowing all the involved variable intervals. An example is shown in Fig. 1.

`Box` is also a propagation algorithm. For every pair (f, x), where f is a function of the considered NCSP and x is a variable involved in f, the a other variables in f are replaced with their interval $[y_1], ..., [y_a]$, and the `BoxNarrow` procedure reduces the bounds of $[x]$ such that the new left (resp. right) bound is the leftmost (resp. rightmost) solution of the univariate equation $f(x, [y_1], ..., [y_a]) = 0$. Existing *revise* procedures use a *shaving* principle to narrow $[x]$: Slices $[s_i]$ inside $[x]$ with no solution are discarded by checking whether $f([s_i], [y_1], ..., [y_a])$ does not contain 0 and by using a univariate interval Newton. `Box` is stronger than `HC4` in that the narrowing effort performed by `Box` on a variable x with multiple occurrences removes the overestimation effect on it. However, it is *not optimal in case the other variables y_i also have multiple occurrences.*

These algorithms are used in our experiments as a sub-contractor embedded in a `3B` algorithm [15] (or a variant `3BCID` [21]). `3B` uses a shaving refutation principle similar to SAC [7]. The main procedure splits an interval into slices. A slice at the bounds is discarded if calling a sub-contractor (e.g. `HC4`) on the resulting subproblem leads to no solution.

The `Mohc` algorithm exploits the monotonicity of functions to better contract intervals. Let us first introduce the well-known monotonicity-based interval extension of f, denoted $[f]_M$ in this article. It appears that the overestimation due to a variable x occurring several times in f disappears when f is monotonic w.r.t. x in a given box. (For the sake of conciseness, we sometimes write that "x is monotonic".) Let us take $f(x) = x^3 - 3x^2 + x$ as an example. The image of $[3, 4]$ calculated by the **natural** interval extension $[f]$ of f (i.e., using interval arithmetic for each primitive operator) yields $[-18, 41]$, which is an overestimation of the optimal image. But the derivative $f'(x) = 3x^2 - 6x + 1$, and $f'([3, 4]) = [3, 30] > 0$. We deduce that f is monotonic (increasing) in $[x] = [3, 4]$. In this case, the image can be optimally obtained at both bounds of $[x]$: $[f]_M([3, 4]) = [f(3), f(4)] = [3, 20]$.

3 The Monotonic Hull-Consistency Algorithm

The MOnotonic Hull-Consistency algorithm (in short `Mohc`) is a new constraint propagation algorithm that exploits monotonicity of functions to better contract a box. The propagation loop is exactly the same AC3-like algorithm performed by the famous `HC4` and `Box`. Its novelty lies in the `Mohc-Revise` procedure handling one contraint individually and described in Algorithm 1.

Algorithm 1 `Mohc-Revise` (**in-out** $[B]$; **in** f, Y, W, ρ_{mohc}, τ_{mohc}, ϵ)

```
HC4-Revise(f(Y,W) = 0, Y, W, [B])
if W ≠ ∅ and ρ_mohc[f] < τ_mohc then
  [G] ← GradientCalculation(f, W, [B])
  (f^og, W) ← OccurrenceGrouping(f, W, [B], [G])
  (f_max, f_min, X, W) ← ExtractMonotonicVars(f^og, W, [B], [G])
  MinMaxRevise([B], f_max, f_min, Y, W)
  MonotonicBoxNarrow([B], f_max, f_min, X, [G], ε)
end if
```

This procedure aims at narrowing the current box $[B]$. It works on a unique equation[1] $f(Y,W) = 0$, in which the variables in Y occur once in the expression f whereas the variables in W occur several times in f.

`Mohc-Revise` starts by a call to `HC4-Revise` (an exception terminating the procedure is raised if an empty box is obtained, proving the absence of solution). If f contains variables with multiple occurrences ($W \neq \emptyset$) and if another condition is fulfilled (see Section 4.1), then five procedures are called to detect and exploit the monotonicity of f.

The `GradientCalculation` function simply computes the gradient of f. More precisely, it computes the partial derivatives w.r.t. every variable w in W having multiple occurrences.

The `OccurrenceGrouping` function is not required in `Mohc-Revise` and can be viewed as an improvement of it. It rewrites the expression f into a new form f^{og} such that the image $[f^{og}]_M([B])$ computed by the monotonicity-based interval extension is sharper than, or at worst equal to, the image $[f]_M([B]) \subset [f]([B])$. This sophisticated function is briefly introduced in Section 4.2.

Using the vector $[G]$, for every variable $w \in W$ (with multiple occurrences), the function `ExtractMonotonicVars` checks whether 0 belongs to the partial derivative $\frac{\partial f^{og}}{\partial w}([B])$. If it does not, it means that f^{og} is monotonic w.r.t. w, so that w is removed from W to be added in X. At the end, X contains variables with multiple occurrences that are monotonic; W contains those that are not detected to be monotonic. (In the following, $[g_i] = \frac{\partial f^{og}}{\partial x_i}([B])$ is the i^{th} component of $[G]$. It denotes the partial derivative on the i^{th} variable x_i in X.) Finally, the function returns the two expressions exploiting, for every variable $x_i \in X$, the monotonicity of f^{og} w.r.t. x_i. f_{min} is the expression f^{og} in which every $[x_i]$ is replaced by $\underline{x_i}$ (resp. $\overline{x_i}$) if f^{og} is increasing (resp. decreasing) w.r.t. x_i. For f_{max}, every $[x_i]$ is replaced by $\overline{x_i}$ (resp. $\underline{x_i}$) if f^{og} is increasing (resp. decreasing) w.r.t. x_i.

The next two routines are in the heart of `Mohc-Revise` and are detailed below. They mainly work with the two functions f_{min} and f_{max}. The procedure `MinMaxRevise` narrows the variables in Y (appearing once in f and thus in f^{og}) and those in W. The procedure `MonotonicBoxNarrow` narrows the monotonic variables in X.

At the end, if `Mohc-Revise` has contracted the interval of a variable in W (more than a user-defined ratio τ_{propag}), then the constraint is pushed into

[1] The procedure can be straightforwardly extended to handle an inequality.

the propagation queue in order to be handled again in a subsequent call to `Mohc-Revise`. Otherwise, we know that a fixpoint in terms of filtering has been reached (under some assumptions). Indeed, nice properties presented in Section 5 explain why `MinMaxRevise` contracts $[Y]$ optimally while `MonotonicBoxNarrow` contracts $[X]$ optimally. The constraint is thus not pushed into the propagation queue.

Note that a variable y in Y, appearing once in f, is handled by `MinMaxRevise`, and not by `MonotonicBoxNarrow`, *even if it is monotonic*. We have made this choice because `MinMaxRevise` is less costly than `MonotonicBoxNarrow` (see Proposition 3) and has *the same power of filtering on* $[y]$ (see Lemma 2).

3.1 The `MinMaxRevise` procedure

Algorithm 2 `MinMaxRevise` (**in-out** $[B]$; **in** f_{max}, f_{min}, Y, W)

```
HC4-Revise(f_min(Y,W) ≤ 0, Y, W, [B])    /* also called MinRevise */
HC4-Revise(f_max(Y,W) ≥ 0, Y, W, [B])    /* also called MaxRevise */
```

`MinMaxRevise` brings a contraction on variables in Y and W. The monotonicity-based interval evaluation yields $[f]_M([B]) = [\underline{[f_{min}]}([B]), \overline{[f_{max}]}([B])]$, where $[f_{min}]$, resp. $[f_{max}]$, denotes the natural extension of f_{min}, resp. f_{max}.
Checking that $0 \in [f]_M([B])$ ($[f]_M([B]) \subset [f]([B])$) amounts in checking that:

$$[f_{min}]([B]) \leq [0,0] \leq [f_{max}]([B])$$

This inequality is thus split into two parts and each of both is used by `HC4-Revise` to narrow intervals of variables in Y and W. Figure 1 illustrates how the first part of `Mohc-Revise` narrows the box of the constraint: $x^2 - 3\,x + y = 0$, with $[x] = [4, 10]$ and $[y] = [-80, 30]$.

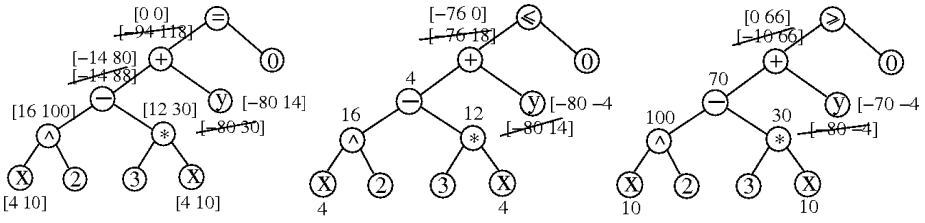

Fig. 1. `HC4-Revise` (**Left**), `MinRevise` (**Center**) and `MaxRevise` (**Right**) applied to $x^2 - 3\,x + y = 0$.

`HC4-Revise` works in two phases (Fig. 1-left). The evaluation phase evaluates every node bottom-up and attachs the result to the node. The second phase, due to the equality node, starts by intersecting the top interval $[-94, 118]$ with 0, and proceeds top-down by applying "inverse" (projection) functions. For instance, since $nplus = nminus + y$, the inverse function of this sum yields the difference $[y] \leftarrow [y] \cap [nplus] - [nminus] = [0,0] - [-14, 80] = [-80, 14]$. The

symmetric operation on *nminus* produces $[-14, 80]$, but this narrowing does not allow a contraction lower in the tree, so that $[x]$ is left unchanged. After this step, `OccurrenceGrouping` detects that the function is monotonic w.r.t. x (trivial case when the derivative is positive so that $f = f^{og}$), hence `ExtractMonotonicVars` puts x in the set X of monotonic variables.

Fig. 1-center shows the first step of `MinMaxRevise`. The tree represents the inequality $f_{min}(4, y) \leq 0$. Calling `HC4-Revise` on this expression produces a new contraction on y because $[x]$ is replaced by $\underline{x} = 4$. On the top of the tree, the narrowing phase intersects $[-76, 18]$ with $[-\infty, 0]$ (inequality), and the first inverse projection function yields $[y] \leftarrow [y] \cap [nplus] - [nminus] = [-76, 0] - [4, 4] = [-80, -4]$. Following the same principle, `MaxRevise` applies `HC4-Revise` to $f_{max}(10, y) \geq 0$ and narrows $[y]$ to $[-70, -4]$ (see Fig. 1-right).

3.2 The `MonotonicBoxNarrow` procedure

Algorithm 3 `MonotonicBoxNarrow` (**in-out** $[B]$; **in** f_{max}, f_{min}, X, $[G]$, ϵ)

for all variable $x_i \in X$ /* x_i contains multiple occurrences and is monotonic */ **do**
  if $\overline{[f_{min}]([B])} < 0$ **and** `applyFmax`$[i]$ **then**
    if $[g_i] > 0$ **then**
      `LeftNarrowFmax`$([x_i], f_{max}^{x_i}, [g_i], \epsilon)$
    else if $[g_i] < 0$ **then**
      `RightNarrowFmax`$([x_i], f_{max}^{x_i}, [g_i], \epsilon)$
    end if
  end if
  if $\underline{[f_{max}]([B])} > 0$ **and** `applyFmin`$[i]$ **then**
    if $[g_i] > 0$ **then**
      `RightNarrowFmin`$([x_i], f_{min}^{x_i}, [g_i], \epsilon)$
    else if $[g_i] < 0$ **then**
      `LeftNarrowFmin`$([x_i], f_{min}^{x_i}, [g_i], \epsilon)$
    end if
  end if
end for

The procedure `MonotonicBoxNarrow` (Algorithm 3) aims at narrowing the interval of every monotonic variable x_i in X. Without loss of generality, assume in the following that x_i is increasing. If the condition $\overline{[f_{min}]([B])} < 0$ *and* `applyFmax`$[i]$, detailed in Section 4.3, is not fulfilled, then the left bound of $[x_i]$ cannot be improved. Otherwise, the `LeftNarrowFmax` procedure is used to improve the left bound of $[x_i]$ with the function $f_{max}^{x_i}$. Also, `RightNarrowFmin` is used to narrow the right bound of $[x_i]$ with the function $f_{min}^{x_i}$ (if $\underline{[f_{max}]([B])} > 0$ *and* `applyFmin`$[i]$). $f_{max}^{x_i}$ and $f_{min}^{x_i}$ denote univariate thick/interval functions depending on x_i. They have similarities with the interval function used in the classical `Box` algorithm. $f_{max}^{x_i}$ and $f_{min}^{x_i}$ are produced by replacing in the f^{og} expression all the variables except x_i with a punctual or an interval value. All the monotonic variables in X, *except* x_i, are replaced with the right bound and all the variables in Y and W are replaced with their current interval in $[B]$.

We detail below how the left bound of $[x_i]$ is improved by the `LeftNarrowFmax` procedure. Note that if `MonotonicBoxNarrow` is called, this requires `MinMaxRevise` be terminated with no failure. This means that `LeftNarrowFmax` will never return an empty interval for $[x_i]$: either $[x_i]$ is left unchanged, or $[x_i]$ is narrowed to a non-empty interval (see cases 1 and 2 in Fig. 4).

3.3 The `LeftNarrowFmax` procedure

This procedure has a close connection with the `LeftNarrow` procedure used by the well-known `Box` algorithm [23, 3]. However, because f is monotonic w.r.t. x_i, the contraction process is faster, that is, it is a true dichotomic, and thus log-time, process.

Algorithm 4 `LeftNarrowFmax` (**in-out** $[x]$; **in** f^x_{max}, $[g]$, ϵ)

if $\overline{[f^x_{max}](\underline{x})} < 0$ /* test of existence */ **then**
  $[l] \leftarrow [x]$
  $size \leftarrow \epsilon \times$ `Diam`$([l])$
  while `Diam`$([l]) > size$ **do**
    $x_m \leftarrow$ `Mid`$([l])$; $z_m \leftarrow \overline{f^x_{max}(x_m)}$ /* $z_m \leftarrow \underline{f^x_{min}(x_m)}$ in `{Left|Right}NarrowFmin` */
    $[l] \leftarrow [l] \cap x_m - \frac{z_m}{[g]}$ /* Newton iteration */
  end while
  $[x] \leftarrow [\underline{l}, \overline{x}]$
end if

Let us illustrate `LeftNarrowFmax` (Algorithm 4) applied to the f^x_{max} function depicted in Fig. 2. Starting with $[l] = [x]$, the goal is to narrow the bounds of $[l]$ for providing a tight approximation of the point L, i.e., the new left bound of $[x]$. `LeftNarrowFmax` provides a sharp enclosure of L and keeps only its left bound at the end (last line of Algorithm 4).

A first existence test checks that $\overline{f^x_{max}(\underline{x})} < 0$, i.e., the point A in Fig. 2-left is below zero. Otherwise, a zero occurs in $\underline{x}$ so that $[x]$ cannot be narrowed, leading to an early termination of the procedure.

A dichotomic process is then run until $Diam([l]) < size$. A classical Newton iteration is iteratively launched from the midpoint x_m of $[l]$, e.g., from the point B in Fig. 2-left, and from the point C in Fig. 2-right.

Graphically, an iteration of a univariate interval Newton [20] intersects $[l]$ with the projection on the x axis of the cone (dotted lines) enclosing all the partial derivatives. Note that the cone forms an angle of at most 90 degrees because the function is monotonic and the derivative $[g]$ is positive[2]. This explains why the diameter of $[l]$ is divided by at least 2 at each iteration. Indeed, if $z_m < 0$ (Fig. 2-left), then the term $-\frac{z_m}{[g]}$ is positive and the dichotomic process will continue in the right side of x_m. If $z_m > 0$ (Fig. 2-right), then $-\frac{z_m}{[g]} < 0$ and one will proceed with the left side of x_m.

[2] Every Newton iteration could recompute a tighter partial derivative $[g]$ although we do not perform it in our implementation.

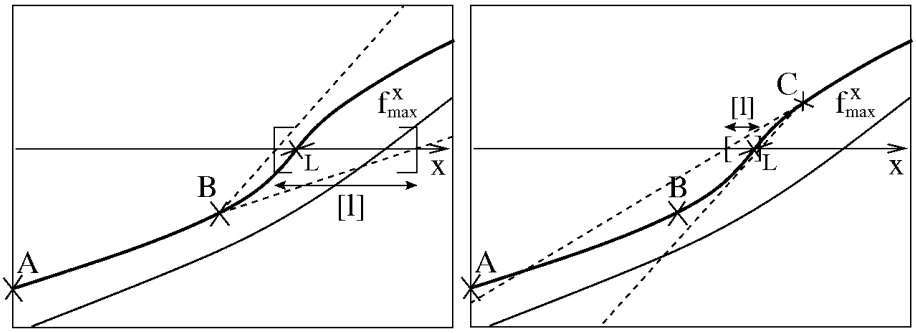

Fig. 2. First two Newton iterations for narrowing the left bound of $[x]$.

Lemma 1. *The procedures* `LeftNarrowFmax`*,* `RightNarrowFmin`*,* `LeftNarrowFmin`*,* `RightNarrowFmax` *terminate and run in time* $O(log_2(\frac{1}{\epsilon}))$*, where* ϵ *is a precision expressed as a ratio of interval diameter.*

Finally note that Newton iterations called inside `LeftNarrowFmax` and `RightNarrowFmax` work with $z_m = \overline{f^x_{max}(x_m)}$, that is, a *punctual* curve (in bold in the figure), and not with a thick function. In the same way, `LeftNarrowFmin` and `RightNarrowFmin` work with $z_m = \underline{f^x_{min}(x_m)}$.

4 Advanced features of `Mohc-Revise`

4.1 The user-defined parameter τ_{mohc} and the array ρ_{mohc}

The user-defined parameter $\tau_{mohc} \in [0, 1]$ allows the monotonicity-based procedures to be called more or less often (see Algorithm 1). For every constraint, $\rho_{mohc}[f]$ tries to predict whether the monotonicity-based treatment that follows is promising: the procedures exploiting monotonicity are called only if $\rho_{mohc}[f] < \tau_{mohc}$. These ratios are computed in a preprocessing procedure called after every bisection (i.e., choice point) on the current box $[B]$, as follows:

$$\rho_{mohc}[f] = \frac{Diam([f]_M([B]))}{Diam([f]([B]))} = \frac{Diam([\underline{[f_{min}]}([B]), \overline{[f_{max}]([B])}])}{Diam([f]([B]))}$$

This ratio thus indicates whether the monotonicity-based image of a function is sufficiently sharper than the natural one. As confirmed by the experiments detailed in Section 6, this ratio is relevant for the bottom-up evaluation phases of `Minrevise` and `Maxrevise`, and also for `MonotonicBoxNarrow` in which a lot of evaluations are performed.

The experiments below show that the parameter τ_{mohc} is useful when `Mohc` is a sub-contractor of `3BCID` [21]. In this case, `Mohc` is called many times between two branching points, so that the CPU time required by the preprocessing procedure filling the array ρ_{mohc} is negligible. This trend would also be true for any other sophisticated contractor calling a constraint propagation sub-contractor, such as `3B` [15], `Quad` [14] or `Box-k` [2]. Future experiments will confirm if tuning a parameter τ_{mohc} is still useful when `Mohc` is called only once between two branching points.

4.2 The `OccurrenceGrouping` function

The motivation is the following. If $0 \in \frac{\partial f}{\partial x}([B])$, then one cannot deduce that x is monotonic. However, it is possible that there exists a subgroup of the occurrences of x, which are replaced by a new auxiliary variable x_a, such that $\frac{\partial f}{\partial x_a}([B]) \geq 0$. Also, another subgroup of occurrences, which are replaced by a new auxiliary variable x_b, may verify $\frac{\partial f}{\partial x_b}([B]) \leq 0$. In this case, such an occurrence grouping rewrites the expression f into a new form f^{og} such that the image $[f^{og}]_M([B])$ computed by the monotonicity-based interval extension is sharper than, or at worst equal to, the image $[f]_M([B]) = [f]([B])$.

This problem can be formalized by a linear program and solved by a specific algorithm `OccurrenceGrouping` in time $O(k\, log_2(k))$ for each variable x occurring k times in a function f. Details can be found in the companion paper [1].

Consider the function $f_1(x) = -x^3 + 2x^2 + 6x$, with $[x] = [-1.2, 1]$. f_1 is not detected monotonic since the image of $[-1.2, 1]$ by natural evaluation of the derivative $f_1'(x) = -3x^2 + 4x + 6$ contains 0. `OccurrenceGrouping` produces: $f_1(x) = f_1^{og}(x_a, x) = -x_a^3 + 2(0.35x_a + 0.65x)^2 + 6\,x_a$. We can observe that: $[f_1^{og}]_M([x]) = [-5.472, 7] \subseteq [f_1]_M([x]) = [-8.2, 10.608]$.

At the end, `OccurrenceGrouping` returns the new expression f^{og} and a new set W that includes the new variables x_a and x_b, if any. (`ExtractMonotonicVars` tranfers x_a and x_b into the set X of monotonic variables.)

4.3 Avoiding calls to the dichotomic process

This section gathers several features that have been introduced in `Monotonic-BoxNarrow` to avoid calls to `LeftNarrowFmax` (and symmetric procedures).

First condition

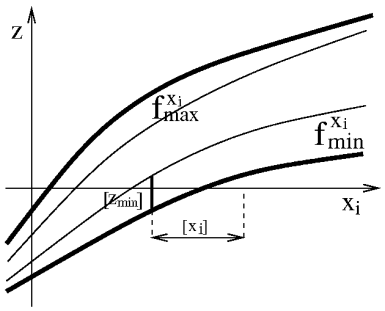

Fig. 3. Monotonic functions $f_{max}^{x_i}$ and $f_{min}^{x_i}$

We first depict in Figure 3 the increasing functions $f_{max}^{x_i}$ and $f_{min}^{x_i}$ enclosing f^{og}. The function between the two curves in bold is sharper than the natural enclosure used in the standard `BoxNarrow` because the monotonic variables in $X \setminus \{x_i\}$ are replaced by points. However, it is still an overestimation of the optimal function because of the multiple occurrences of variables in W.

$f_{max}^{x_i}$ is above $f_{min}^{x_i}$. Note that both functions can overlap in the general case although the upper function $\overline{f_{max}^{x_i}}$ (in bold) is always above $f_{min}^{x_i}$.

We have represented on the figure an initial value for $[x_i]$, before a call to `LeftNarrowFmax`. The figure justifies the condition $\overline{[f_{min}]([B])} \geq 0$ in Algorithm 3 that avoids the call to `LeftNarrowFmax`. Let $[z_{min}]$ be $[f_{min}]([B])$ (vertical segment in bold on the figure). It immediately shows that since $\overline{z_{min}} \geq 0$, $\underline{x_i}$ is solution, so that the lower bound of $[x_i]$ cannot be improved.

Arrays `applyFmin` and `applyFmax`

For every monotonic variable $x_i \in X$, the booleans `applyFmin[i]` and `applyFmax[i]` are initialized to *true*. They are updated in the four procedures `LeftNarrowFmax`, `RightNarrowFmax`, `LeftNarrowFmin` and `RightNarrowFmin`. With no loss of generality, let us explain the principle assuming that f is increasing w.r.t. x_1 ($x_1 \in X^+$; X^+ denotes the set of increasing variables; X^- denotes the set of decreasing variables; $\overline{X^+}$ denotes the vector in which every $x_i \in X^+$ is replaced with the upper bound $\overline{x_i}$.) Since x_1 is increasing, `LeftNarrowFmax` is called to narrow the left bound of $[x_1]$. Three cases may occur according to $\underline{x_1}$, as illustrated in Fig. 4.

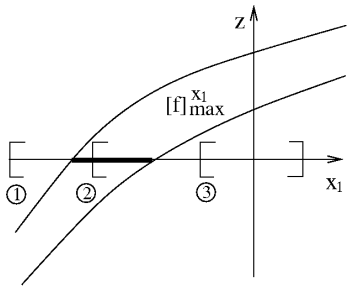

Fig. 4. Three possible cases for the left bound of $[x_1]$.

In the cases 1 and 2, and *not* in the case 3, there is a point $v_1 \in [x_1]$ which is a zero (i.e., which belongs to the segment in bold). That is: $\exists v_1 \in [x_1]$ *s.t.* $0 \in [z_1]$, with: $[z_1] = f^{x_1}_{max}(v_1) = [f](v_1, \overline{X^+ \setminus \{x_1\}}, \underline{X^-}, [Y], [W])$.

Cases 1 and 2 are easily detected at the first line of `LeftNarrowFmax`, by slightly complexifying the existence test. The key point is that the relation $0 \in [z_1]$ implies:

$$\forall i \neq 1, \forall v_i \in [x_i] : \underline{z_i} \leq \underline{z_1} \leq 0$$

With[3]:

$$[z_i] = [f](v_i, \underline{X^+ \setminus \{x_i\}}, \overline{X^-}, [Y], [W]) = f^{x_i}_{min}(v_i)$$

The rightmost inequality comes directly from $0 \in [z_1]$. The leftmost inequality comes from the study of monotonicity of f: $v_i \leq \overline{x_i}$ and the bounds of X selected in $[z_1]$ maximize f while the bounds of X selected in $[z_i]$ minimize f.

Thus, in cases 1 and 2, we know that $\underline{f^{x_i}_{min}(v_i)} \leq 0$. This means that for every $i \neq 1$, $f^{x_i}_{min}$ cannot bring any additional narrowing to $[x_i]$, the relation used

[3] If x_i is decreasing, then $[z_i] = [f](v_i, \underline{X^+}, \overline{X^- \setminus \{x_i\}}, [Y], [W])$.

to shave the interval being always true, even for $v_i = \overline{x_i}$ (if x_i is increasing). In other terms, if x_i is increasing (resp. decreasing), then it is useless to call `RightNarrowFmin` (resp. `LeftNarrowFmin`) to contract $[x_i]$. That is why, in the cases 1 or 2, for all $i \neq 1$, `applyFmin[i]` is set to *false*.

This advanced feature is a slight generalization of the feature proposed by Jaulin and Chabert in [11]. In their paper, $f_{min}^{x_i}$ and $f_{max}^{x_i}$ are punctual functions because there are no sets Y and W.

4.4 Lazy evaluations of f_{min} and f_{max}

The implemented `MohcRevise` procedure is in fact optimized in the aim of reusing as much as possible the different evaluations of f_{min} and f_{max}. In Algorithm 3, intervals $[z_{min}] = [f_{min}]([B])$ and $[z_{max}] = [f_{max}]([B])$ do not need be recomputed at each iteration. Furthermore, these intervals have been previously computed in the bottom-up evaluation phases of `MinMaxRevise` and can be transmitted from a procedure to the other.

The value $\overline{z_{max}}$ can also be used to add a first and cheap call to a Newton iteration at the very beginning of `LeftNarrowFmax`: $[x] \leftarrow [x] \cap \overline{x} - \frac{\overline{z_{max}}}{[g]}$.

Finally, the existence test of `LeftNarrowFmax` makes it possible to reuse the value $[z_l] = [f_{max}^x](\underline{x})$. This allows us to add, just after the existence test, a second cheap Newton iteration : $[x] \leftarrow [x] \cap \underline{x} - \frac{\overline{z_l}}{[g]}$.

4.5 The `LazyMohc` variant

`LazyMohc` is a simplified version of `Mohc` in which the `MonotonicBoxNarrow` procedure only calls the first and cheap Newton iteration described above for every bound of x_i in X. In other words, the `LazyMohc` variant runs `MinMaxRevise`, two cheap Newton iterations per monotonic variable, but no dichotomic process.

5 Properties

A very interesting property concerning `Mohc` is that the `Mohc-Revise` procedure can compute an optimal box w.r.t. an individual constraint if certain conditions are fulfilled. These conditions thus identify a polynomial subclass (Propositions 1 and 2) of the hull-consistency problem (i.e., searching for an optimal box) which is difficult when there exist multiple occurrences of variables. The corresponding propositions appear below and the proofs can be found in appendix.

Proposition 1 *Let $c : f(X, Y, W) = 0$ be a constraint such that f is continuous and differentiable w.r.t. every variable in the box $[B]$. Variables in X and W appear several times in f while variables $y_i \in Y$ appear once. For every $x_i \in X$, f is monotonic w.r.t. x_i.*

If W is empty and if for all $y_i \in Y$, $0 \notin \frac{\partial f}{\partial y_i}([B])$ (implying that y_i is monotonic), then:
One call to `Mohc-Revise` *computes the hull-consistency of the constraint c in $[B]$ (with a precision ϵ).*

The proof is based on the lemma 2 related to the intervals $[Y]$ contracted by `MinMaxRevise`, and on the lemma 3 related to the intervals $[X]$ contracted by `MonotonicBoxNarrow`.

Lemma 2. *With the same hypotheses as in Proposition 1, if W is empty and if for all $y_i \in Y$, $0 \notin \frac{\partial f}{\partial y_i}([B])$ (implying that y_i is monotonic), then:*

One call to `MinMaxRevise` *contracts optimally every $[y_i] \in [Y]$.*

Lemma 3. *With the same hypotheses as in Proposition 1:*

One call to `MonotonicBoxNarrow` *(following a call to* `MinMaxRevise`*) contracts optimally every $[x_i] \in [X]$.*

Proposition 2 is in a sense stronger than Proposition 1 because *no monotonicity hypothesis is required for the variables y_i occurring once* in the expression. However, a stronger and more expensive procedure is used instead of `HC4-Revise`. Replacing `HC4-Revise` with a so-called `TAC-revise` is a way to make a system hull-consistent when all the constraints contain only variables with single occurrence. The fact that a given function f, having only variables with single occurrence, is continuous ensures that the image produced by the natural extension $[f]$ is optimal. But the top-down narrowing phase manages in a sense inverse functions of f that are not necessarily continuous. `HC4-Revise` returns the hull of the different continuous subparts provided by the piecewise analysis performed at each node for the inverse functions. Instead, `TAC-revise` combinatorially combines the continuous subparts of different nodes for narrowing optimally the variables, thus achieving hull-consistency. Details about `TAC-revise` can be found in [6].

Proposition 2 *Let us call* `Mohc-Revise'` *a variant of* `Mohc-Revise` *in which* `HC4-Revise` *is replaced by* `TAC-revise`*. Let us consider the same hypotheses as in Proposition 1. Then:*

If W is empty, one call to `Mohc-Revise'` *computes the hull-consistency of the constraint c in $[B]$ (with a precision ϵ).*

This proposition is significant because in practice, after only a few bisections in the search tree, `HC4-Revise` generally computes a box as sharp as `TAC-revise` does, except in pathological cases. Thus, `Mohc-Revise` (calling the standard `HC4-Revise` procedure) *often* computes hull-consistency when all the variables appear once or are monotonic.

Finally, Proposition 3 details the time complexity of `Mohc-Revise`.

Proposition 3 *Let c be a constraint. Let n be its number of variables, e be the number of nodes in its expression tree (i.e., twice its number of unary and binary operators), and k be the maximum number of occurrences of a variable. Let ϵ be the precision expressed as a ratio of interval diameter.*

`Mohc-Revise` *is time $O(n\,(e\,log_2(\frac{1}{\epsilon}) + k\,log_2(k)))$.* `LazyMohc-Revise` *is time $O(e + n + n\,k\,log_2(k))$.*

6 Experiments

We have implemented `Mohc` and `LazyMohc` with the interval-based `C++` library `Ibex` [5, 4]. `Mohc` has been tested on 17 benchmarks with a finite number of zero-dimensional solutions issued from COPRIN's web page [18]. In the presented experiments, all the `Mohc`-based solving strategies embed `Mohc` as a sub-contractor of `3BCID` [21].

6.1 Tuning the user-defined parameters

The first experiments allow us to get an idea of relevant values for τ_{mohc} and ϵ. Fig. 5-left is interesting because it shows that finely tuning ϵ has no significant impact on performance. For most of the NCSPs, the best value falls between $\frac{1}{32}$ and $\frac{1}{8}$, and the curves are rather flat. We have thus fixed ϵ to 10% (at least when `Mohc` is a sub-contractor of `3BCID`).

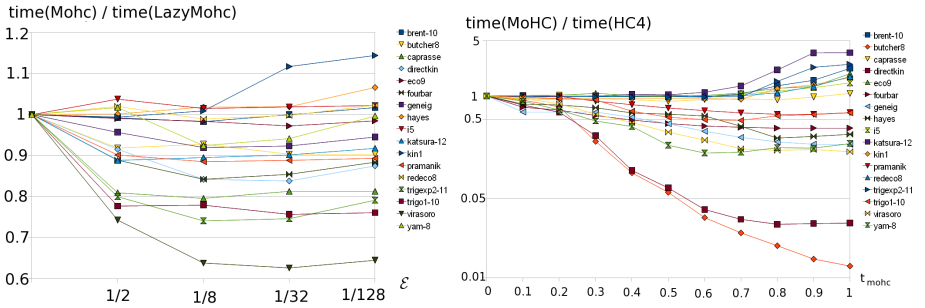

Fig. 5. Tuning the user-defined parameters. **Left:** Tuning ϵ. For every benchmark, the curves shows how a ratio $\frac{Time(Mohc)}{Time(LazyMohc)}$ evolves when ϵ decreases (i.e., the reached precision in `MonotonicBoxNarrow` increases). **Right:** Tuning τ_{mohc}. For every benchmark, the curve shows how a ratio $\frac{Time(Mohc)}{Time(HC4)}$ evolves when τ_{mohc} increases.

Fig. 5-right shows that, for all the NCSPs, the best value falls between 0.6 and 0.99. [4] More precisely, even on NCSPs that do not benefit from our monotonicity-based procedures, setting τ_{mohc} to 0.6 only slightly decreases the performance (only 11% in the worst case, i.e., `Katsura`). That is why the experiments that follow perform two trials with $\tau_{mohc} = 0.7$ and $\tau_{mohc} = 0.99$ (for the most favorable NCSPs).

6.2 Experimental protocol

We have selected all the NCSPs with multiple occurrences of variables found in the first two sections (polynomial and non polynomial systems) of the Web page. We have added `Brent`, `Butcher`, `Direct Kinematics` and `Virasaro` from the section describing the *difficult problems*. All the competitors are also available in `Ibex`, thus making the comparison fair.

Our `Mohc`-based solving strategy uses a round-robin variable selection. Between two branching points, three procedures are called in sequence. First, a monotonicity test just checks whether every monotonicity-based function evaluation contains zero. Its specificity is that the test does not apply to the initial functions f in the NCSP, but to the functions f^{og} produced by `OccurrenceGrouping`. Second, the contractor `3BCID(Mohc)` is called. Third, an interval Newton is run if the current box has a diameter 10 or less. All the parameters in `3BCID`, `HC4`, `Box` and `Mohc` (except τ_{mohc}) have been fixed to default values. The precision

[4] Although not shown on curves, the value 0.99 seems always better than the value 1.0...

ratio in 3B and Box is 10% ; the number of additional slices handled by the CID part is 1; a constraint is pushed into the propagation queue if the interval of one of its variables is reduced more than $\tau_{propag} = 10\%$.

All the experiments have been performed on an Intel 6600 2.4 GHz, and a timeout of at least one hour has been chosen for each benchmark.

6.3 Results

Table 1 compares the CPU time and number of choice points obtained by Mohc with those obtained by competitors: HC4 and Box.

The table reports very good results obtained by Mohc, both in terms of filtering power (low number of choice points) and CPU time. As expected, the bad results obtained by Box highlight that Box is not relevant when several variables in a same contraint have multiple occurrences. For the NCSPs Yamamura1, Brent and Kin1, Box shows a slightly better contraction power than Mohc, but it does not pay off in terms of performance. This underlines that it is better to perform a box narrowing effort less often, when monotonicity has been detected for a given variable.

The comparison with HC4 is more interesting. Mohc and HC4 obtain similar results on 8 of the 17 benchmarks. With $\tau_{mohc} = 0.7$, note that the loss in performance w.r.t. HC4 is negligible. It is inferior to 5%, except for Redeco8 (9%) and Katsura (25%). On 6 NCSPs, Mohc shows a gain comprised between 2.7 and 7.9. On Butcher, Direct Kinematics and Fourbar, a very good gain in CPU time of resp. 153, 48 and 37 is observed.

Although the difference is not so significant, Mohc generally provides better results than LazyMohc, in particular when $\tau_{mohc} = 0.99$.

7 Related Work

The first constraint propagation algorithm exploiting monotonicity appears in the interval-based solver ALIAS [16] developed by Merlet. It is a 2B-consistency algorihm that is improved when the "*GradientSolve*" option is selected to compute and exploit the gradient of functions appearing in constraints. Every projection function f^o_{proj} used to narrow a given occurrence o is first explicitly and symbolically generated (and simplified) before calling on it a procedure close to ExtractMonotonicVars (contrarily to HC4-Revise, no expression tree is used). The monotonicity-based evaluation $[f^o_{proj}]_M$ then brings a better contraction on $[o]$ than $[f^o_{proj}]$ would do.

Monotonicity of functions has also be used in quantified NCSPs to easily contract a universally quantified variable that is monotonic [10].

Jaulin and Chabert propose in [11] a constraint propagation algorithm called Octum. Note that Octum and Mohc have been initiated independently in the first semester of 2009. The most common part is MonotonicBoxNarrow that is nearly the same in both algorithms. We have already mentioned in Section 4.3 that our arrays applyFmin and applyFmax are inspired by Octum and have been adapted to the general case (functions having as well non monotonic variables appearing once (Y) or several times (W)). Compared to Jaulin and Chabert's algorithm, Mohc presents additional features:

Table 1. Results obtained by `Mohc`. The first column includes the name of the benchmark; the bottom of the cell contains the corresponding number of equations and the number of solutions. The other columns report the results obtained by different algorithms. Every cell shows the CPU time in second (above) and the number of choice points (below). The contraction algorithms are `3BCID(HC4)` (column `HC4`), `3BCID(Box)` (column `Box`), `MonoTest` followed by `3BCID(HC4)` (column `MonoTest`). `MonoTest` is also used before the contractor shown in the four last columns: `3BCID(LazyMohc)` with $\rho_{mohc} = 0.7$ (column Lazy(0.7)), the same with $\rho_{mohc} = 0.99$ (column Lazy(0.99)), `3BCID(Mohc)` with $\rho_{mohc} = 0.7$ and $\epsilon = 10\%$ (column Mohc(0.7)), the same with $\rho_{mohc} = 0.99$ (column Mohc(0.99)). All the algorithms are followed by a call to interval Newton before the next bisection. The last column yields the gain obtained by `Mohc`, i.e., $\frac{Time(3BCID(HC4)\ based\ strategy)}{Time(3BCID(Mohc)\ based\ strategy)}$ (the denominator is given by the best of the previous two columns).

NCSP	`HC4`	`Box`	MonoTest	Lazy(0.7)	Lazy(0.99)	Mohc(0.7)	Mohc(0.99)	Gain
Butcher	282528	25867	281664	5221	1986	5026	**1842**	153
8 3	1.8e+8	1.7e+6	1.8e+8	2.2e+6	346063	2.2e+6	**324669**	554
Direct kin.	17515	>28800	17507	481	431	458	**363**	48
11 2	1.4e+6		1.4e+6	11931	8811	9541	**5609**	250
Fourbar	13121	11011	1069	429	420	366	**353**	37
4 3	8.5e+6	732429	965343	79697	67397	58571	**45695**	186
Virasoro	7158	>28800	7173	1538	1413	1241	**902**	7.9
8 224	2.6e+6		2.6e+6	135833	102997	79211	**38739**	67
Geneig	598	>7200	390	117	95.2	116	**87.6**	6.8
6 10	205859		161211	17059	9083	15341	**6975**	30
Yamam.1	11.8	15.3	11.7	2.26	2.91	**2.02**	2.69	5.8
8 7	3017	**183**	3017	357	345	303	297	10
Pramanik	95.9	278	35.9	21.9	22.1	19.6	**19.6**	4.9
3 2	124661	23017	69259	13817	9649	12691	**8435**	15
Hayes	41.7	282	41.6	17.2	**13.6**	17.3	13.9	3.0
8 1	17763	7247	17763	4447	**1707**	4437	1717	10
Trigo1	150.7	773	151	74.0	92.3	**55.8**	71.9	2.7
10 9	2565	1005	2565	641	603	461	455	5.6
Caprasse	2.77	32.2	2.73	2.51	2.94	2.74	**2.34**	1.18
4 18	1309	719	1309	951	499	903	**391**	3.3
Kin1	1.95	68.7	1.96	**1.78**	3.33	1.97	3.36	0.99
6 16	87	**65**	87	83	83	87	81	1.1
Trigexp2	90.1	>3600	90.9	**88.2**	228	91.4	169	0.99
11 0	15187		15187	14303	12567	15099	7717	2.0
I5	**55.7**	>3600	55.9	58.4	81.7	58.5	82.9	0.95
10 30	10621		10621	9809	8849	9811	**8715**	1.2
Eco9	**13.9**	102.0	**13.9**	15.1	26.5	14.6	26.0	0.95
9 16	6193	4991	6193	6047	4707	6037	**4343**	1.4
Brent	19.0	311.0	**18.9**	20.0	42.1	20.2	41.4	0.94
10 1008	3923	**2137**	3923	3807	3341	3815	3189	1.2
Redeco8	**6.23**	69.8	6.28	6.32	11	6.82	10.88	0.91
8 8	2441	1913	2441	2231	1741	2347	**1537**	1.6
Katsura	**77.4**	2265	77.8	104	274	103	245	0.75
12 7	4251	3557	4251	3671	3373	3573	**3151**	1.3

- `Mohc` exploits monotonicity of one function in one variable interval once the box becomes sufficiently small to make appear this property. `Octum` requires a function be monotonic in *all* its variable intervals simultaneously.
- Occurrence grouping quickly rewrites the constraint expressions in order to detect more cases of monotonicity.
- Contrarily to `Octum`, `Mohc` uses the `MinMaxRevise` function to quickly contract the intervals of variables occurring once (Y) and of those which are not monotonic (W). Due to Proposition 2, `Mohc` does not need to call the more costly `MonotonicBoxNarrow` procedure to handle *monotonic variables that appear once* in the expression.

8 Conclusion

This paper has presented a new interval constraint propagation algorithm exploiting the monotonicity of functions. Using ingredients present in the existing `HC4-Revise` and `BoxNarrow`, `Mohc` has the potential to replace advantageously `HC4` and `Box`, as shown by our first experiments. To confirm this claim, we need to also conduct our experiments with strategies using only `Box`, `HC4` or `Mohc`, i.e., without `3B` or `3BCID`.

References

1. I. Araya, B. Neveu, and G. Trombettoni. A New Monotonicity-Based Interval Extension Using Occurrence Grouping. In *Workshop IntCP*, 2009.
2. I. Araya, G. Trombettoni, and B. Neveu. Filtering Numerical CSPs Using Well-Constrained Subsystems. In *Proc. CP'09, LNCS 5732*, 2009.
3. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.
4. G. Chabert. `www.ibex-lib.org`, 2009.
5. G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
6. G. Chabert, G. Trombettoni, and B. Neveu. Box-Set Consistency for Interval-based Constraint Problems. In *SAC - 20th ACM Symposium on Applied Computing*, pages 1439–1443, Santa Fe, USA, 2005.
7. R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.
8. B. Faltings. Arc-consistency for Continuous Variables. *Artif. Intelligence*, 65, 1994.
9. E. Freuder. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1):24–32, 1982.
10. A. Goldsztejn, C. Michel, and M. Rueher. Efficient Handling of Universally Quantified Inequalities. *Constraints*, 14(1):117–135, 2009.
11. L. Jaulin and G. Chabert. Hull Consistency Under Monotonicity. In *To appear in Proc. CP'09, LNCS 5732*, 2009.
12. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.
13. M. Kieffer, L. Jaulin, E. Walter, and D. Meizel. Robust Autonomous Robot Localization Using Interval Analysis. *Reliable Computing*, 3(6):337–361, 2000.
14. Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J.P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.

15. O. Lhomme. Consistency Techniques for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.
16. J.-P. Merlet. ALIAS: An Algorithms Library for Interval Analysis for Equation Systems. Technical Report 621, INRIA Sophia, 2000. `www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html`.
17. J-P. Merlet. Interval Analysis and Robotics. In *Symp. of Robotics Research*, 2007.
18. J-P. Merlet. `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`, 2009.
19. R.E. Moore, B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.
20. A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, 1990.
21. G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.
22. W. Tucker. A Rigorous ODE Solver and Smale's 14th Problem. *Found. Comput. Math.*, 2:53–117, 2002.
23. P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica : A Modeling Language for Global Optimization*. MIT Press, 1997.

A Proofs

A.1 Proof of Lemma 2

First recall that in every node of the expression tree T_{fmax} (resp. T_{fmin}) representing f_{max} (resp. f_{min}), there is no overestimation due to the variables in X because they are replaced by points. The proof of Lemma 2 requires us prove that two traversals of T_{fmax} (with `HC4-Revise`) is sufficient to reach a fixpoint in contraction on variables $y_j \in Y$ occurring once in T_{fmax}. A second proof must ensure that a second call to `MinMaxRevise` following the call to `MonotonicBoxNarrow` would not bring additional contraction to $[y_j]$.

Only two traversals of T_{fmax}

Since f is a continuous function and, for every $y_j \in Y$, $0 \notin \frac{\partial f}{\partial y_j}([B])$, then the (bottom-up) evaluation or (top-down) narrowing functions g called in *every* node during `HC4-Revise` of T_{fmin} (or T_{fmax}) are continuous and monotonic in every of their arguments intervals. This comes from the rule of composition of functions stating $\frac{\partial f}{\partial y_j}([B]) = \frac{\partial f}{\partial g}(g([B])) \times \frac{\partial g}{\partial y_j}([B])$. Since the multiplication preserves the 0 in the resulting interval, $0 \notin \frac{\partial f}{\partial y_j}([B])$ implies that $0 \notin \frac{\partial f}{\partial g}(g([B]))$ and $0 \notin \frac{\partial g}{\partial y_j}([B])$. The same rule applies to two nodes g_1 and g_2 for which g_2 is an argument of g_1: $\frac{\partial f}{\partial g_2}([B]) = \frac{\partial f}{\partial g_1}(g_1([B])) \times \frac{\partial g_1}{\partial g_2}([B])$. The same reasoning yields that $0 \notin \frac{\partial g_1}{\partial g_2}([B])$. Thus, every bottom-up evaluation function g is monotonic in every of their arguments intervals. The same reasoning applies to the top-down narrowing functions g since an "inverse" function of a monotonic (continuous) function is also monotonic.

Because every node function g in T_{fmin} (or T_{fmax}) is monotonic, g computes an arc-consistent output interval $[z_g]$, i.e., every real number $z \in [z_g]$ has a support in the input intervals. Since T_{fmin} (or T_{fmax}) is a tree, the two traversals of T_{fmin} performed by `HC4-Revise` then optimally narrow every $y_j \in Y$. This is an application [8] of a result by Freuder [9] concerning the finite-domain CSPs

stating that two (directed) arc-consistent traversals of an acyclic CSP makes it globally-consistent. That is, no propagation loop is necessary and the two traversals are sufficient to reach the fixpoint in filtering.

No impact of `MonotonicBoxNarrow`

During `MinMaxRevise`, every $x_i \in X$ is replaced by one of its bound $\overline{x_i}$ or $\underline{x_i}$, although these bounds are not optimal before the call to `MonotonicBoxNarrow`. Fortunately, it turns out that the points picked during `MinMaxRevise` inside every $[x_i]$ have no impact on the contraction brought to any $[y_j]$. Let us detail this point on T_{fmax} (the same reasoning holds for T_{fmin}) and with a variable x_i being increasing. Before the call to `MinMaxRevise`, since x_i is increasing, it is replaced by $\overline{x_i}$ in T_{fmax}. The question becomes: if `MonotonicBoxNarrow` reduces $\overline{x_i}$ to a better value $\overline{x_i'}$, cannot a second call to `MinMaxRevise` with $\overline{x_i'}$ bring a new contraction on $[y_j]$?

During `MaxRevise`, at the end of the bottom-up evaluation phase of T_{fmax}, the root interval is $[z] = [z_l, z_u]$. Three cases may occur according to the signs of z_l and z_u. Fig. 6 illustrates the two interesting cases.

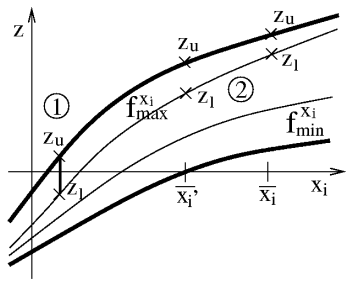

Fig. 6. Two main cases in `Mohc-Revise` when `MaxRevise` is applied to f_{max}.

The case 1 (left side of Fig. 6) occurs when $z_l < 0 < z_u$, resulting in $[z'] = [0, z_u]$ after intersection of $[z]$ with $[0, +\infty]$ in the top of T_{fmax} (because $[f_{max}]([B]) \geq 0$), and thus in a potential contraction of $[y_j]$. However, `MonotonicBoxNarrow` cannot contract $[x_i]$ with $f^{x_i}_{min}$ (i.e., $\overline{x_i'} = \overline{x_i}$) since $\overline{x_i}$ is already a zero. This explains the condition $(\underline{z_l} =)\underline{[f_{max}]([B])} > 0$ in Algorithm 3.

The case 2 (right side of Fig. 6) occurs when $0 < z_l < z_u$, resulting in $[z'] = [z_l, z_u] = [z]$ after intersection of $[z]$ with $[0, +\infty]$. Since $[z]$ is not narrowed, no $[y_j]$ is contracted in the top-down narrowing phase of T_{fmax}. Next, `MonotonicBoxNarrow` reduces $\overline{x_i}$ to $\overline{x_i'}$, but $0 < z_l' < z_u'$ remains true (see figure). Thus, a second call to `MinMaxRevise` could not contract further $[y_j]$.

These cases highlight the duality of the contraction process.

A last trivial case occurs when $z_l < z_u < 0$. `MinMaxRevise` detects that there is no solution due to an empty intersection with $[0, +\infty]$. (Any other smaller values for z_l and z_u, due to $\overline{x_i} \rightarrow \overline{x_i'}$, would imply the same failure.) □

A.2 Proof of Lemma 3

First recall that `MonotonicBoxNarrow` cannot result in an empty box (no solution) because `MinmaxRevise` has been called before with success. This precon-

dition implies that for any bound of each $x_i \in X$, only the three cases depicted in Fig. 4 may occur. When the case 1 occurs, the dichotomic narrowing process performed by `LeftNarrowFmax` (or symmetric procedures) converges onto a final interval $[l]$ including surely a zero of $f_{max}^{x_i}$. Contrarily to the classical `LeftNarrow` procedure used by `Box` [23, 3], $[l]$ surely contains the zero because the $f_{max}^{x_i}$ evaluations lead to no overestimation (see proof of Lemma 2). When the cases 2 or 3 occur (see Fig. 4), $[x_i]$ cannot be (left) narrowed because the bound is a zero of the function.

After only one loop on each variable $x_i \in X$ $(i \in 1, ..., n)$, each of the $2 \times n$ bounds of $[x_i]$ are optimal either because computed by the dichotomic process (case 1 explained above), or because `applyFmin([i])=true` (or `applyFmax([i])=true`), which ensures that no further reduction is expected in these cases 2 or 3 (see Section 4.3). □

A.3 Proof of Proposition 1

Proposition 1 follows Lemmas 2 and 3. One call to `MinMaxRevise` and one loop on the monotonic variables in X ensure that hull-consistency is achieved. □

A.4 Proof of Proposition 2

The lemma 3 also holds for Proposition 2, but the lemma 4 must replace the lemma 2.

Lemma 4. *With the same hypotheses as in Proposition 2, one call to* `MinMaxRevise'` *(calling* `TAC-revise`*) contracts optimally every* $[y_j] \in [Y]$.

The second part of the proof of the lemma 4 follows that of the lemma 2. The first part of the proof is based on the correctness of `TAC-revise`.

Remark. We assume in Proposition 2 that f is continuous in the current box. This hypothesis can be relaxed provided that the bottom-up expression evaluations are performed by a "`TAC-eval`" procedure that would combinatorially combine the continuous parts extracted in the different nodes of the expression tree.

A.5 Proof of Proposition 3 (time complexity)

Calls to `HC4-Revise` (two traversals of an expression tree), `GradientCalculation` (computing all the partial derivatives also amounts in two tree traversals) and `MinMaxRevise` are $O(e)$. A call to `OccurrenceGrouping` [2] is $O(n\,k\,log_2(k))$. The complexity of `MonotonicBoxNarrow` is time $O(n\,e\,log_2(\frac{1}{\epsilon}))$. One Newton iteration takes $O(e)$ (one function evaluation, plus one gradient calculation). The maximum number of possible slices in one interval $[x_i]$ is $\frac{1}{\epsilon}$. The number of Newton iterations is $O(log_2(\frac{1}{\epsilon}))$ (see Lemma 1). In `LazyMohc-Revise`, `MonotonicBoxNarrow` is time $O(n)$ because are called only two constant-time Newton iterations per interval.

Overall, the time complexity of `Mohc-Revise` is $O(e)$ plus $O(n\,k\,log_2(k))$, plus $O(n\,e\log(\frac{1}{\epsilon}))$, or $O(n)$. □