Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)

Exploiting Monotonicity in Interval Constraint Propagation

Ignacio Araya and **Gilles Trombettoni** and **Bertrand Neveu**
COPRIN INRIA, Université Nice–Sophia, Imagine LIGM Université Paris–Est
INRIA, 2004 route des Lucioles, B.P. 93, 06902 Sophia Antipolis, France
rilianx@gmail.com, {Gilles.Trombettoni, Bertrand.Neveu}@sophia.inria.fr

Abstract

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 w.r.t. an individual constraint, uses monotonic versions of the classical HC4-Revise and BoxNarrow procedures.

Mohc-Revise appears to be the first *adaptive* revise procedure ever proposed in (interval) constraint programming. Also, when a function is monotonic w.r.t. every variable, Mohc-Revise is proven to compute the optimal/sharpest box enclosing all the solutions of the corresponding constraint (hull consistency). Very promising experimental results suggest that Mohc has the potential to become an alternative to the state-of-the-art HC4 and Box algorithms.

Introduction

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 apply to various domains such as robotics design and kinematics (Merlet 2007), or dynamic systems in robust control or autonomous robot localization (Kieffer et al. 2000).

Two main types of *contraction algorithms* allow solvers to filter variable domains. Interval Newton and related algorithms generalize to intervals standard numerical analysis methods (Moore 1966). Contraction/filtering algorithms issued from constraint programming are also in the heart of interval-based solvers. The constraint propagation algorithms HC4 and Box (Benhamou et al. 1999; Van Hentenryck, Michel, and Deville 1997) are very often used in solving strategies. 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 of one constraint c when *no* variable appears twice in c. When *one* 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 where *several* variables have multiple occurrences in c.

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 produces no overestimation induced by 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 at the bottom of the search tree, when smaller boxes are handled.

After the background, we describe the Mohc-Revise algorithm. Conditions are then stated to improve the algorithm. Also, when a function is monotonic w.r.t. every variable, a proposition states that Mohc-Revise computes the optimal/sharpest box enclosing all the solutions of the constraint (hull consistency property). Experiments finally highlight the performance of Mohc.

Intervals and numerical CSPs

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

Definition 1 (Basic definitions, notations)
An **interval** $[v] = [a, b]$ *is the set* $\{x \in \mathbb{R},\ a \leq x \leq b\}$.
$\mathbb{IR}$ *denotes the set of all the intervals.*
$\underline{v} = a$ *(resp.* $\overline{v} = b$*) denotes a floating-point number which is the* **left bound** *(resp. the* **right bound***) of* $[v]$.
$\texttt{Mid}([v])$ *denotes the* **midpoint** *of* $[v]$.
$\texttt{Diam}([v]) := \overline{v} - \underline{v}$ *denotes the* **diameter***, or size, of* $[v]$.
A **box** $[V] = [v_1], ..., [v_n]$ *represents the Cartesian product* $[v_1] \times ... \times [v_n]$.

Interval arithmetic has been defined to extend to $\mathbb{IR}$ elementary functions over $\mathbb{R}$ (Moore 1966). For instance, the interval sum is defined by $[v_1] + [v_2] = [\underline{v_1} + \underline{v_2}, \overline{v_1} + \overline{v_2}]$. When a function f is a composition of elementary functions, an *extension* of f to intervals must be defined to ensure a conservative image computation.

Definition 2 (Extension of a function to $\mathbb{IR}$)
Consider a function $f : \mathbb{R}^n \to \mathbb{R}$.
$[f] : \mathbb{IR}^n \to \mathbb{IR}$ *is an* **extension** *of* f *to intervals if:*

$$\forall [V] \in \mathbb{IR}^n \quad [f]([V]) \supseteq \{f(V),\ V \in [V]\}$$
$$\forall V \ \in \mathbb{R}^n \quad f(V) = [f](V)$$

The **natural extension** $[f]_N$ of a real function f corresponds to the mapping of f to intervals using interval arithmetic. The *monotonicity-based extension* is particularly useful in this paper. A function f is *monotonic* w.r.t. a variable v in a given box $[V]$ if the evaluation of the partial derivative of f w.r.t. v is positive (or negative) in every point of $[V]$. For the sake of conciseness, we sometimes write that v is monotonic.

Definition 3 **(f_{min}, f_{max}, monotonicity-based extension**) *Let f be a function defined on variables V of domains $[V]$. Let $X \subseteq V$ be a subset of monotonic variables.*

Consider the values x_i^+ and x_i^- such that: if $x_i \in X$ is an increasing (resp. decreasing) variable, then $x_i^- = \underline{x_i}$ and $x_i^+ = \overline{x_i}$ (resp. $x_i^- = \overline{x_i}$ and $x_i^+ = \underline{x_i}$).

Consider $W = V \setminus X$ the set of variables not detected monotonic. Then, f_{min} and f_{max} are functions defined by:

$$\begin{aligned} f_{min}(W) &= f(x_1^-, ..., x_n^-, W) \\ f_{max}(W) &= f(x_1^+, ..., x_n^+, W) \end{aligned}$$

Finally, the monotonicity-based extension $[f]_M$ of f in the box $[V]$ produces the following interval image:

$$[f]_M([V]) = \left[\underline{[f_{min}]_N([W])}, \overline{[f_{max}]_N([W])}\right]$$

Monotonicity of functions is generally used as an **existence test** checking that 0 belongs to the interval image of functions. It has also been used in quantified NCSPs to easily contract a universally quantified variable that is monotonic (Goldsztejn, Michel, and Rueher 2009).

Consider for example $f(x_1, x_2, w) = -x_1^2 + x_1x_2 + x_2w - 3w$ in the box $[V] = [6, 8] \times [2, 4] \times [7, 15]$.

$[f]_N([x_1], [x_2], [w]) = -[6, 8]^2 + [6, 8] \times [2, 4] + [2, 4] \times [7, 15] - 3 \times [7, 15] = [-83, 35]$.

$\frac{\partial f}{\partial x_1}(x_1, x_2) = -2x_1 + x_2$, and $[\frac{\partial f}{\partial x_1}]_N([6, 8], [2, 4]) = [-14, -8]$. Since $[-14, -8] < 0$, we deduce that f is decreasing w.r.t. x_1. With the same reasoning, we deduce that x_2 is increasing. Finally, $0 \in [\frac{\partial f}{\partial w}]_N([x_1], [x_2], [w]) = [-1, 1]$, so that w is not deduced monotonic. Following Def. 3, the monotonicity-based evaluation yields:

$$[f]_M([V]) = \left[\underline{[f](\overline{x_1}, \underline{x_2}, [w])}, \overline{[f](\underline{x_1}, \overline{x_2}, [w])}\right]$$
$$= \left[\underline{[f](8, 2, [7, 15])}, \overline{[f](6, 4, [7, 15])}\right] = [-79, 27]$$

The dependency problem (multiple occurrences)

The *dependency problem* is the main issue of interval arithmetic. It is due to *multiple occurrences of a same variable* in an expression that are handled as different variables by interval arithmetic. In our example, it explains why the interval image computed by $[f]_M$ is different from (and sharper than) the one produced by $[f]_N$. Also, if a factorized form, e.g., $-x_1^2 + x_1x_2 + (x_2 - 3)w$, was used, we would then obtained an even better image. The dependency problem renders in fact NP-hard the problem of finding the optimal interval image of a polynomial (Kreinovich et al. 1997). (The corresponding extension is denoted by $[f]_{opt}$.) The fact that the monotonicity-based extension replaces intervals by bounds explains the following proposition.

Proposition 1 *Let f be a function of V that is continuous over $[V]$. Then,*

$$[f]_{opt}([V]) \subseteq [f]_M([V]) \subseteq [f]_N([V])$$

In addition, if f is monotonic in the box $[V]$ w.r.t. all its variables appearing several times *in f, then the monotonicity-based extension computes the optimal image:*

$$[f]_M([V]) = [f]_{opt}([V])$$

Numerical CSPs

The `Mohc` algorithm presented in this paper aims at solving nonlinear systems of constraints or numerical CSPs.

Definition 4 **(NCSP)** *A* **numerical CSP** *$P = (V, C, [V])$ 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 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, following a *Branch & Contract* scheme:

- *Branch:* the current box is **bisected** on one dimension (variable), generating two sub-boxes.
- *Contract:* filtering (also called **contraction**) algorithms reduce the bounds of the box with no loss of solution.

The process terminates with **atomic boxes** of size at most ω on every dimension. Contraction algorithms comprise **interval Newton**-like algorithms issued from the numerical *interval analysis* community (Moore 1966) along with algorithms from constraint programming. The contraction algorithm presented in this paper takes advantage of the monotonicity of functions, adapting the classical `HC4-Revise` and `BoxNarrow` procedures. 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, `BoxNarrow` first replaces the other a variables in f by their interval $[y_1], ..., [y_a]$. Then, the procedure reduces the bounds of $[x]$ such that the new left (resp. right) bound is the leftmost (resp. rightmost) solution of the equation $f(x, [y_1], ..., [y_a]) = 0$. Existing revise procedures use a *shaving* principle where slices $[s_i]$ in the bounds of $[x]$ that do not satisfy the constraint are eliminated from $[x]$.

Contracting optimally a box w.r.t. an individual constraint is referred to as the **hull-consistency** problem. Similarly to the optimal interval image computation, due to the dependency problem, hull-consistency is not tractable in general. `HC4-Revise` is known to achieve the hull-consistency of constraints having *no* variable with multiple occurrences, provided that the function and projection functions are continuous. The *Box-consistency* achieved by `BoxNarrow` is stronger (Collavizza, Delobel, and Rueher 1999) and enforces the hull-consistency when the constraint contains only *one* variable with multiple occurrences. Indeed, the shaving process performed by `BoxNarrow` on a variable x suppresses the overestimation effect on x. However, it is *not optimal in case the other variables y_i also have multiple occurrences*.

These algorithms are sometimes used in our experiments as a sub-contractor of a `3BCID` (Trombettoni and Chabert 2007), a variant of `3B` (Lhomme 1993). `3B` uses a shaving refutation principle that 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

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 `HC4` and `Box`. Its novelty lies in the `Mohc-Revise` procedure handling one constraint[1] $f(V) = 0$ individually and described in Algorithm 1.

Algorithm 1 `Mohc-Revise` (**in-out** $[V]$; **in** $f, V, \rho_{mohc}, \tau_{mohc}, \epsilon$)

`HC4-Revise` $(f(V) = 0, [V])$
if `MultipleOccurrences`(V) **and** $\rho_{mohc}[f] < \tau_{mohc}$ **then**
  $(X, Y, W, f_{max}, f_{min}, [G]) \leftarrow$ `PreProcessing`$(f, V, [V])$
  `MinMaxRevise` $([V], f_{max}, f_{min}, Y, W)$
  `MonotonicBoxNarrow` $([V], f_{max}, f_{min}, X, [G], \epsilon)$
end if

`Mohc-Revise` starts by calling the well-known and cheap `HC4-Revise` procedure. The monotonicity-based contraction procedures (i.e., `MinMaxRevise` and `MonotonicBoxNarrow`) are then called only if V contains at least one variable that appears several times (function `MultipleOccurrences`). The other condition makes `Mohc-Revise` adaptive. This condition depends on a user-defined parameter τ_{mohc} detailed in the next section. The second parameter ϵ of `Mohc-Revise` is a precision ratio used by `MonotonicBoxNarrow`.

The procedure `PreProcessing` computes the gradient of f. The gradient is stored in the vector $[G]$ and used to partition the variables in V into three subsets X, Y and W:

- variables in X are monotonic and occur several times in f,
- variables in Y occur once in f (they may be monotonic),
- variables $w \in W$ appear several times in f and are *not* detected monotonic, i.e., $0 \in [\frac{\partial f}{\partial w}]_N([V])$.

The procedure `PreProcessing` also determines the two functions f_{min} and f_{max}, introduced in Definition 3, that approximate f by using its monotonicity.

The next two routines are in the heart of `Mohc-Revise` and are detailed below. Using the monotonicity of f_{min} and f_{max}, `MinMaxRevise` contracts $[Y]$ and $[W]$ while `MonotonicBoxNarrow` contracts $[X]$.

`HC4-Revise`, `MinMaxRevise` and `Monotonic-BoxNarrow` sometimes compute an empty box $[V]$, proving the absence of solution. An exception terminating the procedure is then raised.

At the end, if `Mohc-Revise` has contracted one interval in $[W]$ (more than a user-defined ratio τ_{propag}), then the constraint is pushed into 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 (see Lemmas 2 and 4).

The MinMaxRevise procedure

We know that:

$$(\exists X \in [X])(\exists Y \in [Y])(\exists W \in [W]) : \; f(X \cup Y \cup W) = 0 \implies f_{min}(Y \cup W) \leq 0 \;\; and \;\; 0 \leq f_{max}(Y \cup W)$$

The contraction brought by `MinMaxRevise` is thus simply obtained by calling `HC4-Revise` on the constraints $f_{min}(Y \cup W) \leq 0$ and $0 \leq f_{max}(Y \cup W)$ to narrow intervals of variables in Y and W (see Algorithm 2).

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

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

`HC4-Revise`$(f_{min}(Y \cup W) \leq 0, [V])$ /* `MinRevise` */
`HC4-Revise`$(f_{max}(Y \cup W) \geq 0, [V])$ /* `MaxRevise` */

Fig. 1 illustrates how `MinMaxRevise` contracts the box $[x] \times [y] = [4, 10] \times [-80, 14]$ w.r.t. the constraint:

$$f(x, y) = x^2 - 3\,x + y = 0$$

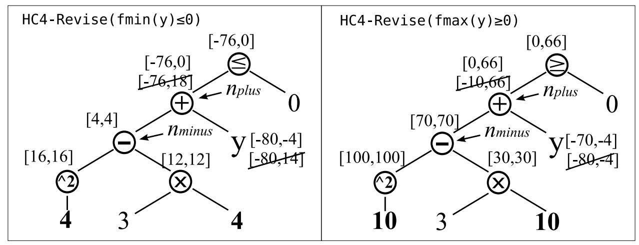

Figure 1: `MinRevise` (left) and `MaxRevise` (right) applied to $x^2 - 3\,x + y = 0$.

Fig. 1-left shows the first step of `MinMaxRevise`. The tree represents the inequality $f(4, y) = f_{min}(y) \leq 0$. `HC4-Revise` works in two phases. The evaluation phase evaluates every node bottom-up (with interval arithmetic) and attaches the result to the node. The second phase, due to the inequality node, starts by intersecting the top interval $[-76, 18]$ with $[-\infty, 0]$ and, if the result is not empty, proceeds top-down by applying *projection ("inverse") functions*. For instance, since $n_{plus} = n_{minus} + y$, the inverse function of this sum yields the difference $[y] \leftarrow [y] \cap ([n_{plus}] - [n_{minus}]) = [-80, 14] \cap ([-76, 0] - [4, 4]) = [-80, -4]$. Following the same principle, `MaxRevise` applies `HC4-Revise` to $f(10, y) = f_{max}(y) \geq 0$ and narrows $[y]$ to $[-70, -4]$ (see Fig. 1-right).

Note that a standard `HC4-Revise` called directly on the constraint $x^2 - 3\,x + y = 0$ (hence not using the monotonicity of f) would have brought no contraction to $[x]$ or $[y]$.

The MonotonicBoxNarrow procedure

This procedure performs a loop on every monotonic variable x_i in X for narrowing $[x_i]$. At each iteration, it works with two *interval* functions, in which all the variables in X, excepting x_i, have been replaced by one bound of the corresponding interval:

$$[f_{min}^{x_i}](x_i) = [f]_N(x_1^-, ..., x_{i-i}^-, x_i, x_{i+1}^-, ..., x_n^-, [Y], [W])$$

$$[f_{max}^{x_i}](x_i) = [f]_N(x_1^+, ..., x_{i-i}^+, x_i, x_{i+1}^+, ..., x_n^+, [Y], [W])$$

Because Y and W have been replaced by their domains, $[f_{max}^{x_i}]$ and $[f_{min}^{x_i}]$ are univariate interval functions depending on x_i (see Fig. 2).

`MonotonicBoxNarrow` calls two subprocedures:

- If x_i is increasing, then it calls:
  - `LeftNarrowFmax` on $[f_{max}^{x_i}]$ to improve $\underline{x_i}$,
  - `RightNarrowFmin` on $[f_{min}^{x_i}]$ to improve $\overline{x_i}$.
- If x_i is decreasing, then it calls:
  - `LeftNarrowFmin` on $[f_{min}^{x_i}]$ to improve $\underline{x_i}$, and
  - `RightNarrowFmax` $[f_{max}^{x_i}]$ to improve $\overline{x_i}$.

We detail in Algorithm 3 how the left bound of $[x]$ is improved by the `LeftNarrowFmax` procedure using $[f^x_{max}]$.

Algorithm 3 `LeftNarrowFmax` (**in-out** $[x]$; **in** $[f^x_{max}]$, $[g]$, ϵ)

if $\overline{[f^x_{max}]_N(\underline{x})} < 0$ /* test of existence */ **then**
    $size \leftarrow \epsilon \times \mathtt{Diam}([x])$
    $[l] \leftarrow [x]$
    while $\mathtt{Diam}([l]) > size$ **do**
        $x_m \leftarrow \mathtt{Mid}([l])$; $z_m \leftarrow \overline{[f^x_{max}](x_m)}$
        /* $z_m \leftarrow \underline{[f^x_{min}](x_m)}$ 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

The process is illustrated by the function depicted in Fig. 2. The goal is to contract $[l]$ (initialized to $[x]$) for providing a sharp enclosure of the point L. The user specifies the precision parameter ϵ (as a ratio of interval diameter) to determine the quality of the approximation. `LeftNarrowFmax` keeps only $\underline{l}$ at the end, as shown in the last line of Algorithm 3 and in step 4 on Fig. 2.

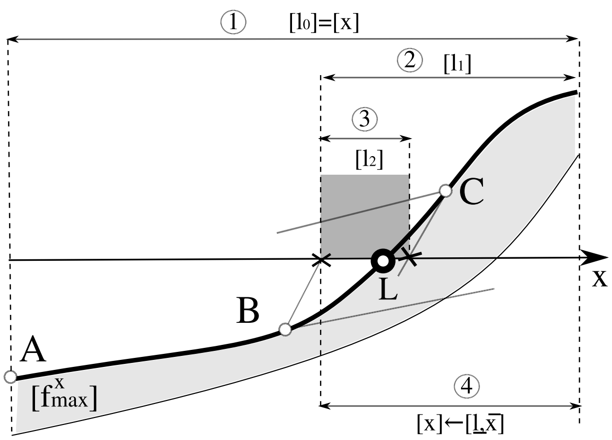

Figure 2: Interval Newton iterations for narrowing $\underline{x}$.

A preliminary existence test checks that $\overline{[f^x_{max}]_N(\underline{x})} < 0$, i.e., the point A in Fig. 2 is below zero. Otherwise, $[f^x_{max}]_N \geq 0$ is satisfied in $\underline{x}$ so that $[x]$ cannot be narrowed, leading to an early termination of the procedure. We then run a dichotomic process until $\mathtt{Diam}([l]) \leq size$. A classical univariate interval Newton iteration is iteratively launched from the midpoint x_m of $[l]$, e.g., in Fig. 2:

1. from the point B (middle of $[l_0]$, i.e., $[l]$ at step 0), and
2. from the point C (middle of $[l_1]$).

Graphically, an iteration of the univariate interval Newton intersects $[l]$ with the projection on the x axis of a cone (e.g., two lines emerging from B and C). The slopes of the lines bounding the cone are equal to the bounds of the partial derivative $[g] = [\frac{\partial f^x_{max}}{\partial x}]_N([x])$. Note that the cone forms an angle of at most 90 degrees because the function is monotonic and $[g]$ is positive. This explains why `Diam([l])` is divided by at least 2 at each iteration.

Lemma 1 *Let ϵ be a precision expressed as a ratio of interval diameter. Then,* `LeftNarrowFmax` *and symmetric procedures terminate and run in time $O(log(\frac{1}{\epsilon}))$.*

Observe that Newton iterations called inside `LeftNarrowFmax` and `RightNarrowFmax` work with $z_m = \overline{[f^x_{max}](x_m)}$, that is, a *degenerate* curve (in bold in the figure), and not with the interval function $[f^x_{max}](x_m)$.

Advanced features of Mohc-Revise

How to make Mohc-Revise adaptive

The user-defined parameter $\tau_{mohc} \in [0, 1]$ allows the monotonicity-based procedures to be called more or less often during the search (see Algorithm 1). For every constraint, the procedures exploiting monotonicity of f are called only if $\rho_{mohc}[f] < \tau_{mohc}$. The ratio ρ_{mohc} indicates whether the monotonicity-based image of a function is sufficiently sharper than the natural one:

$$\rho_{mohc}[f] = \frac{Diam([f]_M([V]))}{Diam([f]_N([V]))}$$

As confirmed by our experiments, 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.

ρ_{mohc} is computed in a preprocessing procedure called after every bisection/branching. Since more cases of monotonicity occur as long as one goes down to the bottom of the search tree (handling smaller boxes), `Mohc-Revise` is able to activate in an adaptive way the machinery related to monotonicity. `Mohc-Revise` thus appears to be the first adaptive revise procedure ever proposed in (interval) constraint programming.

Occurrence Grouping for enhancing monotonicity

A new procedure called `OccurrenceGrouping` has been in fact added in `Mohc-Revise` just after the preprocessing. When f is *not* monotonic w.r.t. a variable x, it is however possible that f be monotonic w.r.t. a *subgroup of occurrences* of x. Thus, this procedure uses a Taylor-based approximation of f and solves on the fly a linear program to perform a good occurrence grouping that enhances the monotonicity-based evaluation of f. Details and experimental evaluation appear in (Araya 2010).

Properties

Proposition 2 (Time complexity)
Let c be a constraint. Let n be its number of variables, e be its number of unary and binary operators ($n \leq e$). Let ϵ be the precision expressed as a ratio of interval diameter. Then,
Mohc-Revise is time $O(n\, e\, log(\frac{1}{\epsilon})) = O(e^2\, log(\frac{1}{\epsilon}))$.

The time complexity is dominated by `MonotonicBoxNarrow` (see Lemma 1). A call to `HC4-Revise` and a gradient calculation are both $O(e)$ (Benhamou et al. 1999).

Proposition 3 *Let $c : f(X) = 0$ be a constraint such that f is continuous, differentiable and monotonic w.r.t. every variable in the box $[X]$. Then, with a precision ϵ,* `MonotonicBoxNarrow` *computes the hull-consistency of c.*

Proofs can be found in (Araya 2010) and (Chabert and Jaulin 2009). However, the new Proposition 4 below is stronger in that the variables appearing once (Y) are handled by `MinMaxRevise` and not by `MonotonicBoxNarrow`.

Proposition 4 *Let $c : f(X, Y) = 0$ be a constraint, in which variables in Y appear once in f. If f is continuous, differentiable and monotonic w.r.t. every variable in the box $[X \cup Y]$, then, with a precision ϵ,* `Mohc-Revise` *computes the hull-consistency of c.*

A complete proof can be found in (Araya 2010). It is also proven that no monotonicity hypothesis is even required for the variables in Y provided that `Mohc-Revise` uses a combinatorial variant of `HC4-Revise`.

Lemmas 2, 3 and 4 below mainly show Propositions 3 and 4. They also prove the correction of `Mohc-Revise`.

Lemma 2 *When* `MonotonicBoxNarrow` *reduces the interval of a variable $x_i \in X$ using $[f_{max}^{x_i}]$ (resp. $[f_{min}^{x_i}]$), then, for all $j \neq i$, $[f_{min}^{x_j}]$ (resp. $[f_{max}^{x_j}]$) cannot bring any additional narrowing to the interval $[x_j]$.*

Lemma 2 is a generalization of Proposition 1 in (Chabert and Jaulin 2009) to interval functions ($[f_{max}^{x_i}]$ and $[f_{min}^{x_i}]$).

Lemma 3 *If $0 \in [z] = [f_{max}]([Y \cup W])$ (resp. $0 \in [z] = [f_{min}]([Y \cup W])$), then* `MonotonicBoxNarrow` *cannot contract an interval $[x_i]$ ($x_i \in X$) using $[f_{min}^{x_i}]$ (resp. $[f_{max}^{x_i}]$).*

Lemma 4 *If* `MonotonicBoxNarrow` *(following a call to* `MinMaxRevise`*) contracts $[x_i]$ (with $x_i \in X$), then a second call to* `MinMaxRevise` *could not contract $[Y \cup W]$ further.*

Lemmas 2 and 4 justify why no loop is required in `MohcRevise` for reaching a fixpoint in terms of filtering.

Proofs of Lemmas 3 and 4

Fig. 3 helps us to understand the proofs in the case f is increasing. We distinguish two cases according to the initial right bound of the interval $[x_i]$.

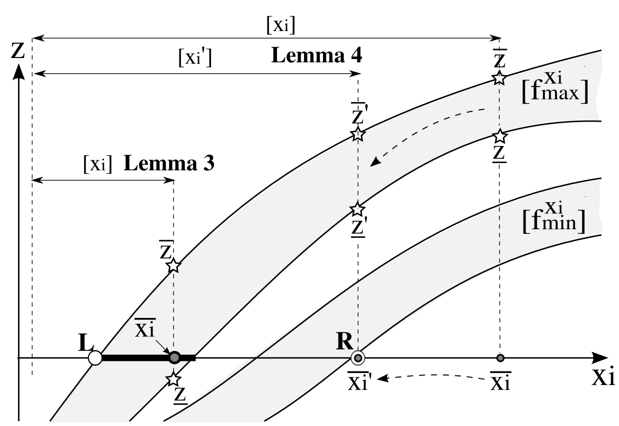

Figure 3: Proofs of Lemmas 3 and 4 stressing the duality of `MinMaxRevise` and `MonotonicBoxNarrow`. $[z]$ is the image of $[f_{max}]$ obtained by the evaluation phase of `MaxRevise`.

In Lemma 3, we have $0 \in [z] = [f_{max}]([Y \cup W])$, i.e., $\underline{z} \leq 0 \leq \overline{z}$. This condition is in particular true when `MaxRevise` brings a contraction. We can verify that $\overline{x_i}$ cannot be improved by `RightNarrowFmin`: since $\overline{x_i}$ is a solution of $[f_{max}^{x_i}](x_i) = 0$ (the dark segment in Fig. 3), $\overline{x_i}$ also satisfies the constraint $[f_{min}^{x_i}](x_i) \leq 0$ that is used by `RightNarrowFmin`.

In Lemma 4, we have $0 < [z] = [f_{max}]([Y \cup W])$ (`MaxRevise` does not bring any contraction). After the contraction performed by `RightNarrowFmin`, the right bound of the interval becomes $\overline{x_i'}$. A new evaluation of $[f_{max}]([Y \cup W])$ yields $[z']$ that is still above 0, so that a second call to `MaxRevise` would not bring any additional contraction. □

Improvement of MonotonicBoxNarrow

Finally, Lemmas 2 and 3 provide simple conditions to save calls to `LeftNarrowFmax` (and symmetric procedures) inside `MonotonicBoxNarrow`.

Due to these added conditions, as confirmed by profiling tests appearing in (Araya 2010), 35% of the CPU time of `Mohc-Revise` is spent in `MinMaxRevise` whereas only 9% is spent in the more costly `MonotonicBoxNarrow` procedure (between 1% and 18% according to the instance).

Experiments

We have implemented `Mohc` with the interval-based `C++` library `Ibex` (Chabert 2010). All the competitors are also available in `Ibex`, thus making the comparison fair: `HC4`, `Box`, `Octum` (Chabert and Jaulin 2009), `3BCID(HC4)`, `3BCID(Box)`, `3BCID(Octum)`.

`Mohc` and competitors have been tested on the same `Intel 6600 2.4 GHz` over 17 NCSPs with a finite number of zero-dimensional solutions issued from COPRIN's web page[2]. 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 Kin.` and `Virasaro` from the section called *difficult problems*.

All the solving strategies use a round-robin variable selection. Between two branching points, three procedures are called in sequence. First, a monotonicity-based existence test, improved by Occurrence Grouping, checks whether the image computed by every function contains zero[3]. Second, the evaluated contractor is called: `Mohc`, `3BCID(Mohc)`, or one of the competitors listed above. Third, an interval Newton is run if the current box has a diameter 10 or less. All the parameters have been fixed to default values. The shaving precision ratio in `3B` and `Box` is 10% ; a constraint is pushed into the propagation queue if the interval of one of its variables is reduced more than $\tau_{propag} = 1\%$ with all the contractors except `3BCID(HC4)` and `3BCID(Mohc)` (10%). For `Mohc`, the parameter τ_{mohc} has been fixed to 70% or 99%. ϵ is 3% in `Mohc` and 10% in `3BCID(Mohc)`.

Results

Table 1 compares the CPU time and number of choice points obtained by `Mohc` and `3BCID(Mohc)` with those obtained by competitors. The last column yields the gain obtained by `Mohc`, i.e.: $Gain = \frac{CPU\ time\,(best\ competitor)}{CPU\ time\,(best\ Mohc\ based\ strategy)}$

The table reports very good results obtained by `Mohc`, both in terms of filtering power (low number of choice points) and CPU time. Results obtained by `3BCID(Box)`,

[2] See www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html

[3] The time required by this existence test is small compared to the total time (generally less than 10%; 1% for `3BCID`) while it sometimes greatly improves the performance of *competitors*.

Table 1: Experimental results. The first column includes the name of the system, its number of equations and the number of solutions. The other columns report the CPU time in second (above) and the number of choice points (below) for all the competitors.

NCSP	HC4	Box	3B(HC4)	Mohc70	Mohc99	3B(Mo70)	3B(Mo99)	Gain
Butcher 8 3	>4e+5	>4e+5	282528 1.8e+8	>4e+5	>4e+5	5431 2.2e+6	**1722** 288773	163 623
Direct kin. 11 2	>2e+4	>2e+4	17507 1.4e+6	2560 777281	2480 730995	428 8859	**356** 5503	49.1 253
Virasoro 8 224	>2e+4	>2e+4	7173 2.5e+6	1180 805047	1089 715407	1051 71253	**897** 38389	8.00 66.8
Yamam.1 8 7	32.4 29513	12.6 3925	11.7 3017	19.2 24767	27.0 29973	**2.2** 345	2.87 295	5.30 10.2
Geneig 6 10	1966 4.1e+6	3721 1.3e+6	390 161211	463 799439	435 655611	107 13909	**81.1** 6061	4.81 26.6
Hayes 8 1	163 541817	323 214253	41.6 17763	30.9 73317	27.6 49059	17.0 4375	**13.8** 1679	3.02 10.6
Trigo1 10 9	93 5725	332 6241	151 2565	**30** 1759	30.6 1673	57.7 459	73.2 443	3.10 5.79
Fourbar 4 3	863 1.6e+6	2441 1.1e+6	1069 965343	361 437959	**359** 430847	366 58571	373 45561	2.40 21.2
Pramanik 3 2	26.9 103827	91.9 81865	35.9 69259	30.3 87961	25.0 69637	**20.8** 12691	21.3 8429	1.29 8.22
Caprasse 4 18	2.04 7671	11.5 5957	2.73 1309	**1.87** 4577	2.69 3741	2.64 867	4.35 383	1.09 3.42
Kin1 6 16	6.91 1303	26.9 689	1.96 87	5.68 1055	5.65 931	**1.79** 83	3.43 83	1.09 1.05
Redeco8 8 8	3769 1.0e+7	9906 7.9e+6	6.28 2441	3529 6.8e+6	2936 4.6e+6	**6.10** 2211	10.65 1489	1.03 1.64
Trigexp2 11 0	1610 1.6e+6	>2e+4	**86.9** 14299	1507 1417759	1027 935227	**87** 14299	165 7291	1.00 1.96
Eco9 9 16	39.9 115445	94.1 110423	**13.9** 6193	46.8 97961	44.2 84457	**14.0** 6025	26.6 4309	0.99 1.44
I5 10 30	9310 2.4e+7	>2e+4	**55.9** 10621	7107 1.6e+7	7129 1.5e+7	57.5 9773	84.1 8693	0.97 1.22
Brent 10 1008	497 1.8e+6	151 23855	**18.9** 3923	244 752533	232 645337	19.9 3805	41.4 3189	0.95 1.23
Katsura 12 7	182 271493	2286 251727	**77.8** 4251	106 98779	143 94249	104 3573	251 3471	0.75 1.22

`Octum` and `3BCID(Octum)` are not reported because the methods are not competitive at all with `Mohc`. For instance, `Octum` is one order of magnitude worse than `Mohc`. The superiority of `Mohc` over `Box` highlights that it is better to perform a better box narrowing effort less often, when monotonicity has been detected for a given variable. `Mohc` and `HC4` obtain similar results on 9 of the 17 benchmarks. With $\tau_{mohc} = 70\%$, note that the loss in performance of `Mohc` (resp. `3BCID(Mohc)`) w.r.t. `HC4` (resp. `3BCID(HC4)`) is negligible. It is inferior to 5%, except for `Katsura` (25%).

On 6 NCSPs, `Mohc` shows a gain comprised between 2.4 and 8. On `Butcher` and `Direct kin.`, a very good gain in CPU time of resp. 163 and 49 is observed. Without the monotonicity existence test before competitors, a gain of 37 would be obtained in `Fourbar`. As a conclusion, the combination `3BCID(Mohc)` appears to be a must.

Related Work

A constraint propagation algorithm exploiting monotonicity appears in the interval-based solver ALIAS [4]. The revise procedure does not use a tree for representing an expression f (contrarily to `HC4-Revise`). Instead, every projection function f^o_{proj} is generated to narrow every occurrence o and is evaluated with a monotonicity-based extension $[f^o_{proj}]_M$. This is more expensive than `MinMaxRevise` and is not optimal since no `MonotonicBoxNarrow` procedure is used.

(Chabert and Jaulin 2009) describes a constraint propagation algorithm called `Octum`. `Mohc` and `Octum` have been initiated independently in the first semester of 2009. To sum up, `Octum` calls `MonotonicBoxNarrow` when *all* the variables of the constraint are monotonic.

Compared to `Octum`, (a) `Mohc` does not require a function be monotonic w.r.t. *all* its variables simultaneously; (b) `Mohc` uses `MinMaxRevise` to quickly contract the intervals of variables (in Y) occurring once (see Proposition 4); (c) `Mohc` uses an Occurrence Grouping to detect more cases of monotonicity.

A first experimental analysis (not reported here) shows that the even better performance of `Mohc` is mainly due to the condition stated in Lemma 3 (and tested during `MinMaxRevise`), used to save calls to `LeftNarrowFmax` (and symmetric procedures).

Conclusion

This paper has presented an interval constraint propagation algorithm exploiting monotonicity. Using ingredients present in the existing procedures `HC4-Revise` and `BoxNarrow`, `Mohc` has the potential to replace advantageously `HC4` and `Box`, as shown by our first experiments. `3BCID(Mohc)` seems to be a very promising combination.

The `Mohc-Revise` procedure manages two user-defined parameters, including τ_{mohc} for tuning the sensitivity to monotonicity. A significant future work is to render `Mohc-Revise` auto-adaptive by allowing τ_{mohc} to be automatically tuned during the combinatorial search.

References

Araya, I. 2010. *Exploiting Common Subexpressions and Monotonicity of Functions for Filtering Algorithms over Intervals.* Ph.D. Dissertation, University of Nice–Sophia.

Benhamou, F.; Goualard, F.; Granvilliers, L.; and Puget, J.-F. 1999. Revising Hull and Box Consistency. In *Proc. ICLP*, 230–244.

Chabert, G., and Jaulin, L. 2009. Hull Consistency Under Monotonicity. In *Proc. CP, LNCS 5732*, 188–195.

Chabert, G. 2010. `www.ibex-lib.org`.

Collavizza, H.; Delobel, F.; and Rueher, M. 1999. Comparing Partial Consistencies. *Reliable Comp.* 5(3):213–228.

Goldsztejn, A.; Michel, C.; and Rueher, M. 2009. Efficient Handling of Universally Quantified Inequalities. *Constraints* 14(1):117–135.

Kieffer, M.; Jaulin, L.; Walter, E.; and Meizel, D. 2000. Robust Autonomous Robot Localization Using Interval Analysis. *Reliable Computing* 3(6):337–361.

Kreinovich, V.; Lakeyev, A.; Rohn, J.; and Kahl, P. 1997. *Computational Complexity and Feasibility of Data Processing and Interval Computations*. Kluwer.

Lhomme, O. 1993. Consistency Techniques for Numeric CSPs. In *IJCAI*, 232–238.

Merlet, J.-P. 2007. Interval Analysis and Robotics. In *Symp. of Robotics Research.*

Moore, R. E. 1966. *Interval Analysis.* Prentice-Hall.

Trombettoni, G., and Chabert, G. 2007. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, 635–650.

Van Hentenryck, P.; Michel, L.; and Deville, Y. 1997. *Numerica : A Modeling Language for Global Optimization.* MIT Press.

[4]See www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html