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 when no variable appears twice in it. When one variable appears several times in it, 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.

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 $\{x \in \mathbb{R}, a \leq x \leq b\}$. $\mathbb{R}^n$ denotes the set of all the intervals. $\mathbb{R}$ is the set of all the intervals $v = a$ (resp. $\mathbb{R} = b$) denotes a floating-point number which is the left bound (resp. the right bound) of $[v]$. $\text{Mid}(v)$ denotes the midpoint of $[v]$. $\text{Diam}(v) := v - \text{Mid}(v)$ 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 $\mathbb{R}$ elementary functions over $\mathbb{R}$ (Moore 1966). For instance, the interval sum is defined by $[v_1] + [v_2] = [v_1 + v_2, \text{Mid}(v_1) + v_2, v_1 + \text{Mid}(v_2), v_1 + v_2]$. 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 $\mathbb{R}$)

Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. $[f] : \mathbb{R}^n \rightarrow \mathbb{R}$ is an extension of $f$ to intervals if:

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

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_{\text{min}}, f_{\text{max}}, \text{monotonicity-based extension})**

Let \( f \) be a function defined on variables \( V \) of domains \( [V] \). Let \( X \subset V \) 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 \( x_i^- = x_i \) and \( x_i^+ = x_i \) (resp. \( x_i^- = x_i \) and \( x_i^+ = x_i \)).

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

\[
\begin{align*}
    f_{\text{min}}(W) & = f(x_1^-, ..., x_n^-, W) \\
f_{\text{max}}(W) & = f(x_1^+, ..., x_n^+, W)
\end{align*}
\]

Finally, the monotonicity-based extension \( f_M \) of \( f \) in the box \( [V] \) produces the following interval image:

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

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

\[
\begin{align*}
    [f]_N([x_1], [x_2], [w]) & = [-6, 8, 6, 8] \times [2, 4] \times [2, 4] \times [7, 15] - 3 \times [7, 15] = [-83, 35] \\
    \frac{\partial f}{\partial x_1}(x_1, x_2) & = -2x_1 + x_2, \quad \frac{\partial f}{\partial x_2}(x_1, x_2) = -2x_1 + x_2, \quad \frac{\partial f}{\partial w}(x_1, x_2) = 2x_2 \\
    [f]_M([V]) & = \left([-2x_1 + x_2, 2x_2], [-2x_1 + x_2, 2x_2]\right) \\
    & = \left([-14, -8], [-14, -8]\right) = [-14, -8]
\end{align*}
\]

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., \( -x_1^2 + x_1 x_2 + (x_2 - 3)w \), 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 \( [f]_{\text{opt}} \).) 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,

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

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:

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

**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 \( [V] \in \mathbb{R}^n \).

A solution \( S \subset [V] \) 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 \( \omega \) 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 \([x_i]\) 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 **M**Onotonic **H**ull-**C**onsistency 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 $f(V) = 0$ 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 $\rho_{mohc}[f] < \tau_{mohc}$

(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 $\tau_{mohc}$ detailed in the next section. The second parameter $\epsilon$ 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., $0 \in [\frac{\partial f}{\partial x}]_N([V])$.

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 MonotonicBoxNarrow 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 $\tau_{propag}$), then the constraint is pushed into the propagation queue in order to be handled again in a subsequent calls 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:

$$\exists X \subseteq [X](\exists Y \subseteq [Y] \exists W \subseteq [W]): f(X \cup Y \cup W) = 0 \Rightarrow f_{\min}(Y \cup W) \leq 0 \text{ and } 0 \leq f_{\max}(Y \cup W)$$

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).

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:

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

Fig. 1-left shows the first step of MinMaxRevise. The tree represents the inequality $f(x, y) = f_{\min}(y) \leq 0$. 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 $[y] \leftarrow [y] \cap ([n_{plus}] - [n_{minus}] = [-80, 14] \cap ([70, 0] - [4, 4]) = [-80, -4]$. 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).

The saturated HC4-Revise then raises the constraint $x^2 - 3x + y = 0$ (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, excluding $x_i$, have been replaced by one of the corresponding intervals:

$$[f_{\min}(x_i)](x_i) = [f] \cap [x_i \leftarrow x_i^-, ..., x_i \leftarrow x_i^-; [Y], [W])$$

$$[f_{\max}(x_i)](x_i) = [f] \cap [x_i \leftarrow x_i^+, ..., x_i \leftarrow x_i^+; [Y], [W])$$

Because Y and W have been replaced by their domains, $[f_{\max}]$ and $[f_{\min}]$ 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 $[f_{\max}]$ to improve $x_i$,
  - RightNarrowFmin on $[f_{\min}]$ to improve $x_i$.
- If $x_i$ is decreasing, then it calls:
  - LeftNarrowFmin on $[f_{\min}]$ to improve $x_i$ and
  - RightNarrowFmax on $[f_{\max}]$ to improve $x_i$.

1The procedure can be straightforwardly extended to handle an inequality.
We detail in Algorithm 3 how the left bound of $[x]$ is improved by the LeftNarrowFmax procedure using $\{f_{\text{max}}^x\}$.

**Algorithm 3 LeftNarrowFmax** (in-out $[x]$; in $\{f_{\text{max}}^x\}$; $[g]; \epsilon$

if $\{f_{\text{max}}^x\}N(x) < 0$ /* test of existence */ then
  size $\leftarrow \epsilon \times \text{Diam}(x)$
  $[l] \leftarrow [x]$
  while $\text{Diam}([l]) > size$ do
    $x_m \leftarrow \text{Mid}([l])$; $z_m \leftarrow \{f_{\text{max}}^x\}[x_m]$
    /* $z_m \leftarrow \{f_{\text{max}}^x\}[x_m]$ in $\{\text{Left|Right}\text{NarrowFmax}\}$ */
    $[l] \leftarrow [l] \cap [x_m - \frac{z_m}{|g|}]$ /* Newton iteration */
  end while
  $[x] \leftarrow [l, x]$
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 precision enclosure of the point $L$. The user specifies the precision parameter $\epsilon$ (as a ratio of interval diameter) to determine the quality of the approximation. LeftNarrowFmax keeps only $[l]$ at the end, as shown in the last line of Algorithm 3 and in step 4 on Fig. 2.

A preliminary existence test checks that $\{f_{\text{max}}^x\}N(x) < 0$, i.e., the point $A$ in Fig. 2 is below zero. Otherwise, $\{f_{\text{max}}^x\}N \geq 0$ is satisfied in $x$ so that $[x]$ cannot be narrowed, leading to an early termination of the procedure. We then run a dichotomic process until $\text{Diam}([l]) \leq size$. 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 iteration 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 $|g| = |f_{\text{max}}^x|N(x)$. Note that the cone forms an angle of at most 90 degrees because the function is monotonic and $|g|$ is positive. This explains why $\text{Diam}([l])$ is divided by at least 2 at each iteration.

**Lemma 1** Let $\epsilon$ 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 $\frac{z_m}{|g|} = \frac{\text{Diam}(f_{\text{max}}^x)(x_m)}{\text{Diam}(f_{\text{max}}^x)(x)}$, that is, a degenerate curve (in bold in the figure), and not with the interval function $\{f_{\text{max}}^x\}N(x_m)$.

**Advanced features of Mohc-Revise**

**How to make Mohc-Revise adaptive**
The user-defined parameter $\tau_{\text{mohc}} \in [0, 1]$ 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 $\rho_{\text{mohc}}[f] < \tau_{\text{mohc}}$. The ratio $\rho_{\text{mohc}}$ indicates whether the monotonicity-based image of a function is sufficiently sharper than the natural one:

$$\rho_{\text{mohc}}[f] = \frac{\text{Diam}(f_M([V]))}{\text{Diam}(f_N([V]))}$$

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. $\rho_{\text{mohc}}$ 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, $\epsilon$ be its number of unary and binary operators ($n \leq \epsilon$). Let $\epsilon$ be the precision expressed as a ratio of interval diameter. Then,

Mohc-Revise is time $O(n \epsilon \log(\frac{1}{\epsilon})) = O(\epsilon^2 \log(\frac{1}{\epsilon}))$.

The time complexity is dominated by MonotonicBoxNarrow (see Lemma 1). A call to HC4-Revise and a gradient calculation are both $O(\epsilon)$ (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 $\epsilon$, 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 \( \epsilon \),

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 \( j \neq i \), \([f_{\min}^{x_i}] \) (resp. \([f_{\max}^{x_i}] \)) 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 \( 0 \in [z] = [f_{\max}^{x_i}](Y \cup W) \) (resp. \( 0 \in [z] = [f_{\min}^{x_i}](Y \cup W) \)), 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 toMinMaxRevise) 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 \([x_i]\).

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

In Lemma 4, we have \( 0 < [z] = [f_{\max}^{x_i}](Y \cup W) \) (MaxRevise does not bring any contraction). After the contraction performed by RightNarrowFmin, the right bound of the interval becomes \( z' \). A new evaluation of \([f_{\max}^{x_i}](Y \cup W) \) 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 CORIN’s web page. 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 Wirasarno 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. 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 \( r_{propag} = 1% \) with all the contractors except 3BCID(HC4) and 3BCID(Mohc) (10%). For Mohc, the parameter \( r_{mohc} \) has been fixed to 70% or 99%. \( \epsilon \) 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).

\[ \text{See } \text{www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html} \]

\[ \text{The time required by this existence test is small compared to the total time (generally less than 10%); 1% for 3BCID) while it sometimes greatly improves the performance of competitors.} \]
Table 1: Experimental results. The first column includes the name of the system, the 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.  

<table>
<thead>
<tr>
<th>NCS</th>
<th>HC4</th>
<th>Box</th>
<th>Box(HC4)</th>
<th>Mohc70</th>
<th>Mohc99</th>
<th>Min(Mohc70)</th>
<th>Mohc99(Gain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butcher 8</td>
<td>&gt;4e+5</td>
<td>&gt;4e+5</td>
<td>225258</td>
<td>1.8e+3</td>
<td>&gt;4e+5</td>
<td>&gt;4e+5</td>
<td>5431</td>
</tr>
<tr>
<td>Direct kin. 11/2</td>
<td>&gt;2e+4</td>
<td>&gt;2e+4</td>
<td>1507</td>
<td>1.4e+6</td>
<td>&gt;2e+4</td>
<td>&gt;2e+4</td>
<td>423</td>
</tr>
<tr>
<td>Viasco 8</td>
<td>&gt;2e+4</td>
<td>&gt;2e+4</td>
<td>7173</td>
<td>11.8e3</td>
<td>&gt;2e+4</td>
<td>&gt;2e+4</td>
<td>10899</td>
</tr>
<tr>
<td>Yamam.1 8/9</td>
<td>32.4</td>
<td>12.6</td>
<td>11.7</td>
<td>19.2</td>
<td>27.0</td>
<td>2.2</td>
<td>2.8</td>
</tr>
<tr>
<td>Genmod 6/10</td>
<td>1966</td>
<td>3721</td>
<td>390</td>
<td>463</td>
<td>435</td>
<td>107</td>
<td>81.1</td>
</tr>
<tr>
<td>Hayes 8/1</td>
<td>454817</td>
<td>214253</td>
<td>17763</td>
<td>70317</td>
<td>49059</td>
<td>4275</td>
<td>1679</td>
</tr>
<tr>
<td>Treigol 10/9</td>
<td>93</td>
<td>332</td>
<td>131</td>
<td>30</td>
<td>30.6</td>
<td>577</td>
<td>73.2</td>
</tr>
<tr>
<td>Fourbar 4/3</td>
<td>865</td>
<td>2443</td>
<td>1069</td>
<td>361</td>
<td>359</td>
<td>366</td>
<td>373</td>
</tr>
<tr>
<td>Pramanik 3/2</td>
<td>103827</td>
<td>81865</td>
<td>69259</td>
<td>8796</td>
<td>69967</td>
<td>12697</td>
<td>8429</td>
</tr>
<tr>
<td>Kapsame 11/8</td>
<td>284</td>
<td>11.5</td>
<td>2.7</td>
<td>1.87</td>
<td>2.56</td>
<td>4.39</td>
<td>1.09</td>
</tr>
<tr>
<td>Kin1 6/16</td>
<td>6.91</td>
<td>1303</td>
<td>193</td>
<td>58.6</td>
<td>56.5</td>
<td>1.79</td>
<td>3.43</td>
</tr>
<tr>
<td>Rescon 8/8</td>
<td>1.0e+7</td>
<td>7.9e+6</td>
<td>2441</td>
<td>6.3e+6</td>
<td>4.0e+6</td>
<td>4.3e+6</td>
<td>2211</td>
</tr>
<tr>
<td>Trigexp21 16/11</td>
<td>&gt;2e+4</td>
<td>&gt;2e+4</td>
<td>86.9</td>
<td>1507</td>
<td>1077</td>
<td>87</td>
<td>165</td>
</tr>
<tr>
<td>Ecor 9/16</td>
<td>115445</td>
<td>114759</td>
<td>117522</td>
<td>729</td>
<td>729</td>
<td>1.09</td>
<td></td>
</tr>
<tr>
<td>Brent 10/100</td>
<td>397</td>
<td>151</td>
<td>18.9</td>
<td>244</td>
<td>232</td>
<td>19.9</td>
<td>41.4</td>
</tr>
<tr>
<td>Katsuura 12/7</td>
<td>182</td>
<td>2266</td>
<td>77.8</td>
<td>106</td>
<td>143</td>
<td>106</td>
<td>351</td>
</tr>
</tbody>
</table>

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 Katsuura (25\%).

On 6 NCSs, 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 ALIAS4. The revise procedure does not use a tree for representing an expression $f$ (contrarily to HC4-Revise). Instead, every projection function $f_{\text{proj}}$ is generated to narrow every occurrence $o$ and is evaluated with a monotonicity-based extension $f_{\text{proj}}|_{M}$. 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 to 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 $\tau_{mohc}$, for tuning the sensitivity to monotonicity. A significant future work is to render Mohc-Revise auto-adaptive by allowing $\tau_{mohc}$ to be automatically tuned during the combinational search.

References


---

4See www-sop.inria.fr/coprin/logicels/ALIAS/ALIAS.html