A New Monotonicity-Based Interval Extension
Using Occurrence Grouping

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni
INRIA, University of Nice-Sophia, ENPC
Firstname.Name@sophia.inria.fr

Abstract. When a function $f$ is monotonic w.r.t. a variable in a given domain, it is well-known that the monotonicity-based interval extension of $f$ computes a sharper image than the natural interval extension does. This paper presents a so-called “occurrence grouping” interval extension $[f]_{og}$ of a function $f$. When $f$ is not monotonic w.r.t. a variable $x$ in the given domain $[B]$, we try to transform $f$ into a new function $f^{og}$ that is monotonic in two subsets $x_a$ and $x_b$ of the occurrences of $x$. $f^{og}$ is increasing w.r.t. $x_a$ and decreasing w.r.t. $x_b$. $[f]_{og}$ is the interval extension by monotonicity of $f^{og}$ and produces a sharper interval image than the natural extension does.

For finding a good occurrence grouping, we propose an algorithm that minimizes a Taylor-based overestimation of the image diameter of $[f]_{og}$.

Finally, experiments show the benefits of this new interval extension for solving systems of equations.

1 Introduction

The computation of sharp interval image enclosures is in the heart of interval arithmetics. It allows a computer to evaluate a mathematical formula while taking into account in a reliable way round-off errors due to floating point arithmetics. Sharp enclosures also allow interval methods to quickly converge towards the solutions of a system of constraints over the reals. At every node of the search tree, a test of existence checks that, for every equation $f(X) = 0$, the interval extension of $f$ returns an interval including 0 (otherwise the branch is cut). Also, constraint propagation algorithms can be improved when they use better interval extensions. For instance, the Box algorithm uses a test of existence inside its iterative splitting process [2].

This paper proposes a new interval extension and we first recall basic material about interval arithmetics [10, 11, 8] to introduce the interval extensions useful in our work.

An interval $[x] = [a, b]$ is the set of real numbers between $a$ and $b$. $\overline{x}$ denotes the minimum of $[x]$ and $\overline{x}$ denotes the maximum of $[x]$. The diameter of an interval is: $diam([x]) = \overline{x} - \underline{x}$, and the absolute value of an interval is: $||x|| = max(|\overline{x}|, |\underline{x}|)$. A Cartesian product of intervals is named a box, and is denoted by $[B]$ or by a vector $\{[x_1], [x_2], ..., [x_n]\}$. 
An interval function $[f]$ is a function from $\mathbb{IR}$ to $\mathbb{IR}$, $\mathbb{IR}$ being the set of all the intervals over $\mathbb{R}$. $[f]$ is an interval extension of a function $f$ if the following condition is verified:

- The image $[f](\{x\})$ must be a conservative interval containing the set $\mathcal{I} f(\{x\}) = \{y \in \mathbb{R}, \exists x \in [x] \mapsto y = f(x)\}$. The computation of the image is called evaluation of $f$ in this article.

We can extend this definition to functions with several variables, as follows:

- Let $f(x_1, \ldots, x_n)$ be a function from $\mathbb{R}^n$ to $\mathbb{R}$ and let box $[B]$ be the vector of intervals $\{[x_1], [x_2], \ldots, [x_n]\}$. The image of $[B]$ by $[f]$ must be an interval containing the set $\mathcal{I} f([B]) = \{y \in \mathbb{R}, \exists \{x_1, x_2, \ldots, x_n\} \in [B], y = f(x_1, x_2, \ldots, x_n)\}$.

The optimal image $[f]_{opt}(\{B\})$ is the sharpest interval containing $\mathcal{I} f([B])$. There exist many possible interval extensions for a function, the difficulty being to define an extension that computes the optimal image, or a sharp approximation of it.

The first idea is to use interval arithmetics. Interval arithmetics extends to intervals arithmetic operators $+, -, \times, /$ and elementary functions ($power, \ exp, \ log, \ sin, \ cos, \ldots$). For instance, $[a, b] + [c, d] = [a + c, b + d]$. The natural interval extension $[f]_n$ of a function $f$ evaluates with interval arithmetics all the arithmetic operators and elementary functions in $f$.

When $f$ is continuous inside a box $[B]$, the natural evaluation of $f$ (i.e., the computation of $[f]_n([B])$) yields the optimal image when each variable occurs only once in $f$. When a variable appears several times, the evaluation by interval arithmetics generally produces an overestimation of $[f]_{opt}(\{B\})$, because the correlation between the occurrences of a same variable is lost. Two occurrences of a variable are handled as independent variables. For example $[x] - [x]$, with $[x] \in [0, 1]$ gives the result $[-1, 1]$, instead of $[0, 0]$, as does $[x] - [y]$, with $[x] \in [0, 1]$ and $[y] \in [0, 1]$.

This main drawback of interval arithmetics causes a real difficulty for implementing efficient interval-based solvers, since the natural evaluation is a basic tool for these solvers.

One way to overcome this difficulty is to use monotonicity [5]. In fact, when a function is monotonic w.r.t. each of its variables, this problem disappears and the evaluation (using a monotonicity extension) becomes optimal. For example, if $f(x_1, x_2)$ is increasing w.r.t. $x_1$, and decreasing w.r.t. $x_2$, then the extension by monotonicity $[f]_m$ of $f$ is defined by:

$$[f]_m([B]) = \{f([x_1], [x_2]), f([x_1], [x_2]) = [[f]_n([x_1], [x_2]), [f]_n([x_1], [x_2])]$$

It appears that $[f]_m([B]) = [f]_{opt}(\{B\})$. This property can also be used when $f$ is monotonic w.r.t. a subset of variables, replacing in the natural evaluations the intervals of monotonic variables by intervals reduced to their maximal or minimal values [6]. The obtained image is not optimal, but is sharper than, or equal to,
the image obtained by natural evaluation. For example, if \( f \) is increasing w.r.t. \( x_1 \), decreasing w.r.t. \( x_2 \), and not monotonic w.r.t. \( x_3 \):

\[
[f]_{\text{opt}}([B]) \subseteq [f]_{\text{m}}([B]) = ([f]_{\text{n}}([x_1, x_2], [x_3]), [f]_{\text{n}}([x_1], [x_2], [x_3]) \subseteq [f]_{\text{n}}([B])
\]

This paper explains how to use monotonicity when a function is not monotonic w.r.t. a variable \( x \), but is monotonic w.r.t. subgroups of occurrences of \( x \). We present the idea of grouping the occurrences into 3 sets (increasing, decreasing and non monotonic auxiliary variables) in the next section. Linear programs for obtaining “interesting” occurrence groupings are described in Sections 3 and 4. In Section 5 we propose an algorithm to solve the linear programming problem presented in Section 4. Finally, in Section 6, some experiments show the benefits of this occurrence grouping for solving systems of equations, in particular when we use a filtering algorithm like Mohc [1] exploiting monotonicity.

2 Evaluation by monotonicity with occurrence grouping

In this section, we study the case of a function which is not monotonic w.r.t. a variable with multiple occurrences. We can, without loss of generality, limit the study to a function of one variable: the generalization to a function of several variables is straightforward, the evaluations by monotonicity being independent.

Example 1. Consider \( f_1(x) = -x^3 + 2x^2 + 6x \). We want to calculate a sharp evaluation of this function when \( x \) falls in \([-1.2, 1]\). The derivative of \( f_1 \) is \( f_1'(x) = -3x^2 + 4x + 6 \) and contains a positive term (6), a negative term \((-3x^2)\) and a term containing zero \((4x)\).

\([f_1]_{\text{opt}}([B]) = [-3.05786, 7]\), but we cannot obtain it directly by a simple interval function evaluation (one needs to solve \( f_1'(x) = 0 \), which is in the general case a problem in itself).

In the interval \([-1.2, 1]\), the function \( f_1 \) is not monotonic. The natural interval evaluation yields \([-8.2, 10.608]\), the Horner evaluation \([-11.04, 9.2]\) (see [7]).

When a function is not monotonic w.r.t. a variable \( x \), it sometimes appears that it is monotonic w.r.t. some occurrences. A first naive idea for using the monotonicity of these occurrences is the following. We replace the function \( f \) by a function \( f^{\text{noq}} \), regrouping all increasing occurrences into one variable \( x_a \), all decreasing occurrences into one variable \( x_b \), and the non monotonic occurrences into \( x_c \). The domain of the new auxiliary variables is the same: \([x_a] = [x_b] = [x_c] = [x]\).

For \( f_1 \), this grouping results in \( f_1^{\text{noq}}(x_a, x_b, x_c) = -x_b^3 + 2x_c^2 + 6x_a \). The evaluation by monotonicity of \( f_1^{\text{noq}} \) computes the lower (resp. upper) bound replacing the increasing (resp. decreasing) instances by the minimum (resp. maximum) and the decreasing (resp. increasing) instances by the maximum (resp. minimum), i.e.,

\[
[f_1^{\text{noq}}]_{\text{m}}([-1.2, 1]) = [f_1^{\text{noq}}]_{\text{n}}([-1.2, 1], [-1.2, 1]) = -1^3 + 2[-1.2, 1]^2 - 7.2 = -8.2
\]
(resp. \([f_{1}^{\text{noog}}]_{m}([-1.2, 1]) = 10.608\). Finally, the evaluation by monotonicity is \([f_{1}^{\text{noog}}]_{m}([-1.2, 1]) = [-8.2, 10.608]\).

It appears that the evaluation by monotonicity of the new function \(f^{\text{noog}}\) always provides the same result as the natural evaluation. Indeed, when a node in the evaluation tree corresponds to an increasing function w.r.t. a variable occurrence, the natural evaluation automatically selects the right bound (among both) of the occurrence domain during the evaluation process.

The main idea is then to change this grouping in order to reduce the dependency problem and obtain sharper evaluations. We can in fact group some occurrences (increasing, decreasing, or non monotonic) into an increasing variable \(x_{a}\) as long as the function remains increasing w.r.t. this variable \(x_{a}\).

For example, if one can move a non monotonic occurrence into a monotonic group, the evaluation will be the same or sharper. Also, if it is possible to transfer all decreasing occurrences into the increasing part, the dependency problem will now occur only on the occurrences in the increasing and non monotonic parts.

For \(f_{1}\), if we group together the positive derivative term with the derivative term containing zero we obtain the new function: \(f_{1}^{\text{noog}}(x_{a}, x_{b}) = -x_{b}^3 + 2x_{a}^2 + 6x_{a}\), where \(f_{1}^{\text{noog}}\) is increasing w.r.t. \(x_{a}\) and decreasing w.r.t. \(x_{b}\). We can then use the evaluation by monotonicity obtaining the interval \([-5.32, 9.728]\). We can in the same manner obtain \(f_{1}^{\text{noog}}(x_{a}, x_{c}) = -x_{a}^3 + 2x_{c}^2 + 6x_{a}\), the evaluation by monotonicity yields then \([-5.472, 7.88\]. We remark that we find sharper images than the natural evaluation of \(f_{1}\) does.

In Section 3, we present a linear program to perform occurrence grouping automatically.

**Interval extension by occurrence grouping**

Consider the function \(f(x)\) with multiple occurrences of \(x\). We obtain a new function \(f^{\text{og}}(x_{a}, x_{b}, x_{c})\) by replacing in \(f\) every occurrence of \(x\) by one of the three variables \(x_{a}, x_{b}, x_{c}\), such that \(f^{\text{og}}\) is increasing w.r.t. \(x_{a}\) in \([x]\), and \(f^{\text{og}}\) is decreasing w.r.t. \(x_{b}\) in \([x]\).

Then, we define the interval extension by occurrence grouping of \(f\) by:

\[
[f]^{\text{og}}([B]) := [f^{\text{og}}]_{m}([B])
\]

Unlike the natural interval extension and the interval extension by monotonicity, the interval extension by occurrence grouping is not unique for a function \(f\) since it depends on the occurrence grouping (og) that transforms \(f\) into \(f^{\text{og}}\).

### 3 A 0,1 linear program to perform occurrence grouping

In this section, we propose a method for automatizing occurrence grouping. First, we calculate a Taylor-based overestimation of the diameter of the image computed by \([f]^{\text{og}}\). Then, we propose a linear program performing a grouping that minimizes this overestimation.
3.1 Taylor-Based overestimation

On one hand, as \( f^{og} \) could be not monotonic w.r.t. \( x_c \), the evaluation by monotonicity considers the occurrences of \( x_c \) as different variables such as the natural evaluation would. On the other hand, as \( f^{og} \) is monotonic w.r.t. \( x_a \) and \( x_b \), the evaluation by monotonicity of these variables is optimal. The following two propositions are well-known.

**Proposition 1** Let \( f(x) \) be a continuous function in a box \( [B] \) with a set of occurrences of \( x \): \( \{x_1, x_2, ..., x_k\} \). \( f^c(x_1, ..., x_k) \) is a function obtained from \( f \) considering all the occurrences of \( x \) as different variables. \( f_n([B]) \) computes \( f^{og}([B]) \).

**Proposition 2** Let \( f(x_1, x_2, ..., x_n) \) be a monotonic function w.r.t. each of its variables in a box \( [B] = \{[x_1], [x_2], ..., [x_n]\} \). Then, the evaluation by monotonicity is optimal in \( [B] \), i.e., it computes \( f^{og}([B]) \).

Using these propositions, we observe that \( f^{og}([B]) = f^{og}([x_a], [x_b], [x_c], ..., [x_{c_k}]) \), considering each occurrence of \( x_c \) in \( f^{og} \) as an independent variable \( x_c \) in \( f^c \). Using Taylor evaluation, an upper bound of \( diam([f]_{opt}([B])) \) is given by the right side of (1) in Proposition 3.

**Proposition 3** Let \( f(x_1, ..., x_n) \) be a function with domains \( [B] = \{[x_1], ..., [x_n]\} \). Then,

\[
diam([f]_{opt}([B])) \leq \sum_{i=1}^{n} (diam([x_i]) × |g_i|([B]))
\]

where \( |g_i| \) is an interval extension of \( g_i = \frac{\partial f}{\partial x_i} \).

Using Proposition 3, we can calculate an upper bound of the diameter of \( [f]_{og}([B]) = f^{og}([B]) = f^{og}([f]_{opt}([B])]: \)

\[
diam([f]_{og}([B])) \leq diam([x])([g_a]([B]) + [g_b]([B]) + \sum_{i=1}^{k} [g_{c_i}]([B]))
\]

Where \( [g_a] \), \( [g_b] \) and \( [g_{c_i}] \) are the interval extensions of \( g_a = \frac{\partial f^{og}}{\partial x_a} \), \( g_b = \frac{\partial f^{og}}{\partial x_b} \) and \( g_{c_i} = \frac{\partial f^{og}}{\partial x_{c_i}} \) respectively. \( diam([x]) \) is factorized because \( [x] = [x_a] = [x_b] = [x_{c_1}] = ... = [x_{c_k}] \).

In order to respect the monotonicity conditions required by \( f^{og} \): \( \frac{\partial f^{og}}{\partial x_a} \geq 0 \), \( \frac{\partial f^{og}}{\partial x_b} \leq 0 \), we have the sufficient conditions \( [g_a]([B]) \geq 0 \) and \( [g_b]([B]) \leq 0 \), implying \( [g_a]([B]) = [g_a]([B]) \) and \( [g_b]([B]) = -[g_b]([B]) \). Finally:

\[
diam([f]_{og}([B])) \leq diam([x])([g_a]([B]) - [g_b]([B]) + \sum_{i=1}^{k} [g_{c_i}]([B]))
\]
3.2 A linear program

We want to transform $f$ into a new function $f^{og}$ that minimizes the right side of the relation (2). The problem can be easily transformed into the following integer linear program:

Find the values $r_{a_i}$, $r_{b_i}$ and $r_{c_i}$ for each occurrence $x_i$ that minimize

$$G = [g_a([B])] - [g_b([B])] + \sum_{i=1}^{k} (|g_i([B])|) r_{c_i}$$

subject to:

$$[g_a([B])] \geq 0$$

$$[g_b([B])] \leq 0$$

$$r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1, \ldots, k$$

$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0, 1\} \quad \text{for } i = 1, \ldots, k,$$

where a value $r_{a_i}$, $r_{b_i}$ or $r_{c_i}$ equal to 1 indicates that the occurrence $x_i$ in $f$ will be replaced, respectively, by $x_a$, $x_b$ or $x_c$ in $f^{og}$. $k$ is the number of occurrences of $x$, $[g_a([B])] = \sum_{i=1}^{k} [g_i([B])] r_{a_i}$, $[g_b([B])] = \sum_{i=1}^{k} [g_i([B])] r_{b_i}$, and $[g_i([B])]$, ..., $[g_k([B])]$ are the derivatives w.r.t. each occurrence.

We can remark that all the gradients (e.g., $[g_a([B])]$, $[g_b([B])]$) are calculated using only the derivatives of $f$ w.r.t. each occurrence of $x$ (i.e., $[g_i([B])])$.

Linear program corresponding to Example 1

We have $f_1(x) = -x^3 + 2x^2 + 6x$, $f_1'(x) = -3x^2 + 4x + 6$ for $x \in [-1.2, 1]$. The gradient values for each occurrence are: $[g_1([-1.2, 1])] = [-4.32, 0]$, $[g_2([-1.2, 1])] = [-4.8, 4]$, and $[g_3([-1.2, 1])] = [6, 6]$. Then, the linear program is:

Find the values $r_{a_1}$, $r_{b_1}$ and $r_{c_1}$ that minimize

$$G = \sum_{i=1}^{3} [g_i([B])] r_{a_i} - \sum_{i=1}^{3} [g_i([B])] r_{b_i} + \sum_{i=1}^{3} (|g_i([B])|) r_{c_i})$$

$$= (4r_{a_1} + 6r_{a_2}) + (4.32r_{b_1} + 4.8r_{b_2} - 6r_{b_3}) + (4.32r_{c_1} + 4.8r_{c_2} + 6r_{c_3})$$

subject to:

$$\sum_{i=1}^{3} [g_i([B])] r_{a_i} = -4.32r_{a_1} - 4.8r_{a_2} + 6r_{a_3} \geq 0$$

$$\sum_{i=1}^{3} [g_i([B])] r_{b_i} = 4r_{b_1} + 6r_{b_2} \leq 0$$

$$r_{a_1} + r_{b_1} + r_{c_1} = 1 \quad \text{for } i = 1, \ldots, 3$$

$$r_{a_1}, r_{b_1}, r_{c_1} \in \{0, 1\} \quad \text{for } i = 1, \ldots, 3$$
We obtain the minimum 10.8, and the solution \( r_{a_1} = 1, r_{b_1} = 0, r_{c_1} = 0, r_{a_2} = 0, r_{b_2} = 0, r_{c_2} = 1, r_{a_3} = 1, r_{b_3} = 0, r_{c_3} = 0 \), which is the last solution presented in Section 2. We can remark that the value of the overestimation of \( \text{diam}([f]_{\log}(B)) \) is equal to 23.76 (10.8 \times \text{diam}([-1.2, 1])) whereas \( \text{diam}([f]_{\log}(B)) = 13.352 \). Although the overestimation is quite rough, the heuristic works well on this example. Indeed, \( \text{diam}([f]_{\log}(B)) = 18.808 \), and \( \text{diam}([f]_{\text{opt}}(B)) = 10.06 \).

### 4 A tractable linear programming problem

The linear program above is a 0,1 linear program and is known to be NP-hard in general. We can render it continuous and tractable by allowing \( r_a, r_b, \) and \( r_c \) to get real values. In other words, we allow each occurrence of \( x \) in \( f \) to be replaced by a convex linear combination of auxiliary variables, \( x_a, x_b \) and \( x_c \).

\( f_{\text{opt}} \) being increasing w.r.t. \( x_a \), and decreasing w.r.t. \( x_b \). Each occurrence \( x_i \) is replaced in \( f_{\text{opt}} \) by \( r_a x_a + r_b x_b + r_c x_c \), with \( r_a + r_b + r_c = 1, \frac{\partial f_{\text{opt}}}{\partial x_a} \geq 0 \) and \( \frac{\partial f_{\text{opt}}}{\partial x_b} \leq 0 \). We can then remark that \( f \) and \( f_{\text{opt}} \) have the same natural evaluation.

In Example 1, we can replace \( f_1 \) by \( f_{11}^{\text{opt}} \) or \( f_{12}^{\text{opt}} \) in a way respecting the monotonicity constraints of \( x_a \) and \( x_b \). Considering the interval \( [x] = [-1.2, 1] \):

1. \( f_{11}^{\text{opt}}(x_a, x_b) = -(\frac{5}{15} x_a + \frac{13}{15} x_b)^3 + 2 x_a^3 + 6 x_a : [f_{11}^{\text{opt}}]_{\text{m}}([x]) = [-4.38, 8.205] \)
2. \( f_{12}^{\text{opt}}(x_a, x_b, x_c) = -x_a + 2(0.35 x_a + 0.65 x_c)^2 + 6 x_a : [f_{12}^{\text{opt}}]_{\text{m}}([x]) = [-5.472, 7] \)

**Example 2.** Consider the function \( f_2(x) = x^3 - x \) and the interval \( [x] = [0.5, 2] \). \( f_2 \) is not monotonic and the optimal image \( [f_2]_{\text{opt}}([x]) \) is \([0.385, 6] \). The natural evaluation yields \([-1.975, 7.5] \), the Horner evaluation \([-1.5, 6] \). We can replace \( f_2 \) by one of the following functions.

1. \( f_{21}^{\text{opt}}(x_a, x_b) = x_a - \frac{7}{12} x_a + \frac{1}{12} x_b : [f_{21}^{\text{opt}}]_{\text{m}}([x]) = [-0.75, 6.375] \)
2. \( f_{22}^{\text{opt}}(x_a, x_b) = (\frac{1}{12} x_a + \frac{1}{12} x_b)^3 - x_b : [f_{22}^{\text{opt}}]_{\text{m}}([x]) = [-1.756, 6.09] \)

Taking into account the convex linear combination for realizing the occurrence grouping, the new linear program is:

Find the values \( r_a, r_b, \) and \( r_c \) for each occurrence \( x_i \) that minimize (3) subject to (4), (5), (6) and

\[
\left( r_a, r_b, r_c \right)_i \in [0, 1] \quad \text{for } i = 1, \ldots, k. \quad (7)
\]

**Linear program corresponding to Example 1**

In this example we obtain the minimum 10.58 and the new function \( f_{11}^{\text{opt}}(x_a, x_b, x_c) = -x_a^3 + 2(0.35 x_a + 0.65 x_c)^2 + 6 x_a : [f_{11}^{\text{opt}}]_{\text{m}}([x]) = [-5.472, 7] \). The minimum 10.58 is less than 10.8 (obtained by the 0,1 linear program).

The evaluation by occurrence grouping of \( f_1 \) yields \([-5.472, 7] \), which is sharper than the image \([-5.472, 7.88] \) obtained by the 0,1 linear program presented in Section 3.
Linear program corresponding to Example 2

In this example we obtain the minimum 11.25 and the new function \( f^{og}(x_a, x_b) = \left( \frac{44}{15} x_a + \frac{1}{15} x_b \right)^3 - \left( \frac{11}{15} x_a + \frac{4}{15} x_b \right) \). The image \([-0.75, 6.01]\) obtained by occurrence grouping is sharper than the interval computed by natural and Horner evaluations. Note that in this case the 0,1 linear program of Section 3 yields the naive grouping due to the constraints.

Note that the continuous linear program not only makes the problem tractable but also improves the minimum of the objective function.

5 An efficient Occurrence Grouping algorithm

Algorithm 1 finds \( r_a, r_b, r_c \) (r-values) that minimize \( G \) subject to the constraints. The algorithm also generates the new function \( f^{og} \) that replaces each occurrence \( x_i \) in \( f \) by \( [r_a_i] x_a + [r_b_i] x_b + [r_c_i] x_c \). Note that the r-values are represented by thin intervals, of a few u.l.p. large, for taking into account the floating point rounding errors appearing in the computations.

Algorithm 1 uses a vector \([g_\star]\) of size \( k \) containing interval derivatives of \( f \) w.r.t. each occurrence \( x_i \) of \( x \). For the sake of conciseness, each component of \([g_\star]\) is denoted by \([g_i]\) hereafter, instead of \([g_i](\mathbb{B})\), i.e., \([g_i]\) is the interval \( \frac{\partial f}{\partial x_i}(\mathbb{B}) \).

Algorithm 1

\begin{algorithm}
\caption{Occurrence\_Grouping(in: \( f, [g_\star] \) out: \( f^{og} \))}
\begin{algorithmic}[1]
\State \([G_0] \leftarrow \sum_{i=1}^{k} [g_i] \)
\State \([G_m] \leftarrow \sum_{g_i} [g_i] \)
\If {\( 0 \notin [G_0] \)}
\State OĠ\_case1\(([g_\star], [r_a], [r_b], [r_c])\)
\ElsIf {\( 0 \in [G_m] \)}
\State OĠ\_case2\(([g_\star], [r_a], [r_b], [r_c])\)
\Else
\State /* \( 0 \notin [G_m] \) and \( 0 \in [G_0] \) */
\If {\( [G_m] \geq 0 \)}
\State OĠ\_case3\(([g_\star], [r_a], [r_b], [r_c])\)
\Else
\State OĠ\_case3\(([g_\star], [r_a], [r_b], [r_c])\)
\EndIf
\EndIf
\State \( f^{og} \leftarrow \text{Generate\_New\_Function}(f, [r_a], [r_b], [r_c]) \)
\end{algorithmic}
\end{algorithm}

An asterisk (*) in the index of a symbol represents a vector (e.g., \([g_\star], [r_a] \)).

We illustrate the algorithm using the two univariate functions of our examples: \( f_1(x) = -x^3 + 2x^2 + 6x \) and \( f_2(x) = x^3 - x \) for domains of \( x \): \([-1.2, 1]\) and \([0.5, 2]\) respectively.
The interval derivatives of \( f \) w.r.t. each occurrence of \( x \) have been previously calculated. For the examples, the interval derivatives of \( f_2 \) w.r.t. \( x \) occurrences are \( [g_1] = [0.75, 12] \) and \( [g_2] = [-1, -1] \); the interval derivatives of \( f_1 \) w.r.t. \( x \) occurrences are \( [g_1] = [-4.32, 0], [g_2] = [-4.8, 4] \) and \( [g_3] = [6, 6] \).

In line 1, the partial derivative \( [G_0] \) of \( f \) w.r.t. \( x \) is calculated using the sum of the partial derivatives of \( f \) w.r.t. each occurrence of \( x \). In line 2, \( [G_m] \) gets the value of the partial derivative of \( f \) w.r.t. the monotonic occurrences of \( x \). In the examples, for \( f_1: [G_0] = [g_1] + [g_2] + [g_3] = [-3.12, 10] \) and \( [G_m] = [g_1] + [g_3] = [1.68, 6] \), and for \( f_2: [G_0] = [G_m] = [g_1] + [g_2] = [-0.25, 11] \).

According to the values of \( [G_0] \) and \( [G_m] \), we can distinguish 3 cases. The first case is well-known (\( 0 \not\in [G_0] \) in line 3) and occurs when \( x \) is a monotonic variable. The procedure \texttt{OG\_case1} does not achieve any occurrence grouping: \textit{all the occurrences of} \( x \) \textit{are replaced by} \( x_a \) (if \( [G_0] \geq 0 \)) or by \( x_b \) (if \( [G_0] \leq 0 \)). The evaluation by monotonicity of \( f^{\text{mon}} \) is equivalent to the evaluation by monotonicity of \( f \).

In the second case, when \( 0 \in [G_m] \) (line 5), the procedure \texttt{OG\_case2} (Algorithm 2) performs a grouping of the occurrences of \( x \). Increasing occurrences are replaced by \( (1 - \alpha_1)x_a + \alpha_1x_b \), decreasing occurrences by \( \alpha_2x_a + (1 - \alpha_2)x_b \) and non monotonic occurrences by \( x_c \) (lines 7 to 13 of Algorithm 2). \( f_2 \) falls in this case: \( \alpha_1 = \frac{1}{15} \) and \( \alpha_2 = \frac{11}{15} \) are calculated in lines 3 and 4 of Algorithm 2 using \( [G^+] = [g_1] = [0.75, 12] \) and \( [G^-] = [g_2] = [-1, -1] \). The new function is: \( f_2^{\text{og}}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b) \).

\begin{algorithm}
\caption{\texttt{OG\_case2}(in: \([g_*]\) out: \([r_{a_*}], [r_{b_*}], [r_{c_*}]\))}
\begin{algorithmic}[1]
\STATE \( [G^+] \leftarrow \sum_{|g_i| > 0} [g_i] \)
\STATE \( [G^-] \leftarrow \sum_{|g_i| \leq 0} [g_i] \)
\STATE \( \alpha_1 \leftarrow \frac{[G^+] [G^-] + [G^-] [G^+]}{[G^+] [G^-] - [G^-] [G^+]}} \)
\STATE \( \alpha_2 \leftarrow \frac{[G^+] [G^+] + [G^-] [G^+]}{[G^+] [G^-] - [G^-] [G^+]}} \)
\FORALL{\( g_i \in [g_*] \)}
\STATE \textbf{if} \( |g_i| \geq 0 \) \textbf{then}
\STATE \( (r_{a_i}, [r_{b_i}], [r_{c_i}]) \leftarrow (1 - \alpha_1, [\alpha_1], 0) \)
\STATE \textbf{else if} \( |g_i| \leq 0 \) \textbf{then}
\STATE \( (r_{a_i}, [r_{b_i}], [r_{c_i}]) \leftarrow ([\alpha_2], 1 - \alpha_2, 0) \)
\STATE \textbf{else}
\STATE \( (r_{a_i}, [r_{b_i}], [r_{c_i}]) \leftarrow (0, 0, 1) \)
\STATE \textbf{end if}
\STATE \textbf{end for}
\end{algorithmic}
\end{algorithm}
The third case occurs when \( 0 \notin [G_m] \) and \( 0 \in [G_0] \). W.l.o.g., if \([G_m] \geq 0\), the procedure \( \text{OG\_case3}^+ \) (Algorithm 3) first groups all the decreasing occurrences with the increasing group, i.e., it replaces every monotonic occurrence \( x_i \) by \( x_a \) (lines 2–5). The non monotonic occurrences are then replaced by \( x_a \) in a determined order stored by an array \( \text{index} \) (line 7) as long as the constraint
\[
\sum_{i=1}^{k} r_{a_i} [g_i] \geq 0
\]
is satisfied (lines 9-13). The first non monotonic occurrence \( x_y \) that cannot be replaced because it would make the constraint unsatisfiable is replaced by \( \alpha x_a + (1-\alpha) x_c \), with \( \alpha \) such that the constraint is satisfied and equal to 0, i.e.,
\[
\left( \sum_{i=1, i \neq i'}^{k} r_{a_i} [g_i] \right) + \alpha [g_i] = 0
\]
(lines 15–17). The rest of the non monotonic occurrences are replaced by \( x_c \) (lines 20–22). \( f_1 \) falls in this case. The first and third occurrences of \( x \) are monotonic and are then replaced by \( x_a \). Only the second occurrence of \( x \) is not monotonic, and it cannot be replaced by \( x_a \) because it would make the constraint unsatisfiable. It is then replaced by \( \alpha x_a + (1-\alpha) x_c \), where \( \alpha = 0.35 \) is obtained forcing the constraint (4) to be 0:
\[
[g_1] + [g_3] + \alpha [g_2] = 0.
\]
The new function is:
\[
f^{OG}_1 = -x_a^3 + 2(0.35 x_a + 0.65 x_c)^2 + 6 x_a.
\]

\begin{algorithm}
\caption{\text{OG\_case3}^+(in: \([g_*]\) out: \([r_{a_*}], [r_{b_*}], [r_{c_*}]\))}
\begin{algorithmic}[1]
\STATE \( [g_*] \leftarrow [0, 0] \)
\FORALL \( [g_i] \in [g_*], 0 \notin [g_i] \)
\STATE \([g_a] \leftarrow [g_i] + [g_i] \) /* All positive and negative derivatives are absorbed by \([g_a] */
\STATE \(([r_{a_*}], [r_{b_*}], [r_{c_*}]) \leftarrow (1, 0, 0)\)
\ENDFOR
\STATE \( \text{index} \leftarrow \text{ascending\_sort}([\{[g_i] \in [g_*], 0 \in [g_i]\}, \text{criterion} \leftarrow [\overline{g_i}]/[g_i] ]) \)
\STATE \( j \leftarrow 1 \) ; \( i \leftarrow \text{index}[1] \)
\WHILE \( [g_a] + [g_i] \geq 0 \)
\STATE \(([r_{a_*}], [r_{b_*}], [r_{c_*}]) \leftarrow (1, 0, 0)\)
\STATE \([g_a] \leftarrow [g_i] + [g_i] \)
\STATE \( j \leftarrow j + 1 \) ; \( i \leftarrow \text{index}[j] \)
\ENDWHILE
\STATE \( [\alpha] \leftarrow \frac{[g_a]}{[g_i]} \)
\STATE \(([r_{a_*}], [r_{b_*}], [r_{c_*}]) \leftarrow ([\alpha], 0, 1 - [\alpha])\)
\STATE /* \([g_a] \leftarrow [g_a] + [\alpha] [g_i] */
\STATE \( j \leftarrow j + 1 \) ; \( i \leftarrow \text{index}[j] \)
\ENDWHILE
\STATE \( [\alpha] \leftarrow \frac{[g_a]}{[g_i]} \)
\STATE \(([r_{a_*}], [r_{b_*}], [r_{c_*}]) \leftarrow (0, 0, 1)\)
\STATE \( j \leftarrow j + 1 \) ; \( i \leftarrow \text{index}[j] \)
\ENDWHILE
\end{algorithmic}
\end{algorithm}

1 An occurrence \( x_{i_1} \) is handled before \( x_{i_2} \) if \( [\overline{g_{i_1}}]/[g_{i_1}] \leq [\overline{g_{i_2}}]/[g_{i_2}] \). \text{index}[j] yields the occurrence index \( i \) such that \([g_i]\) is the \( j^{th} \) interval in the sorting order.
Finally, the procedure `Generate_New_Function` (line 15 of Algorithm 1) creates the new function $f^{og}$ symbolically.

**Observations**

Algorithm 1 respects the four constraints (4)–(7). We are currently proving that the minimum of the objective function in (3) is also reached.

Instead of Algorithm 1, we may use a standard Simplex algorithm, providing that the used Simplex implementation is adapted to take into account rounding errors due to floating point arithmetics. In a future work, we will compare the performances of Algorithm 1 and Simplex.

**Time complexity**

The time complexity of `Occurrence_Grouping` for a variable with $k$ occurrences is $O(k \log_2(k))$. It is dominated by the complexity of `ascending_sort` in the `OG_case3` procedure. As shown in the experiments of the next section, the time required in practice by `Occurrence_Grouping` is negligible when it is used for solving systems of equations.

6 Experiments

`Occurrence_Grouping` has been implemented in the Ibex [4, 3] open source interval-based solver in C++. The goal of these experiments is to show the improvements in CPU time brought by `Occurrence_Grouping` when solving systems of equations. Sixteen benchmarks are issued from the COPRIN website [9]. They correspond to square systems with a finite number of zero-dimensional solutions of at least two constraints involving multiple occurrences of variables and requiring more than 1 second to be solved (considering the times appearing in the website). Two instances (`<name>-bis`) have been simplified due to the long time required for their resolution: the input domains of variables have been arbitrarily reduced.

6.1 Occurrence grouping for improving a monotonicity-based existence test

First, `Occurrence_Grouping` has been implemented to be used in a monotonicity-based existence test (`OG` in Table 1), i.e., an occurrence grouping transforming $f$ into $f^{og}$ is applied after a bisection and before a contraction. Then, the monotonicity-based existence test is applied to $f^{og}$: if the evaluation by monotonicity of $f^{og}$ does not contain 0, the current box is eliminated.

The competitor (¬`OG`) applies directly the monotonicity-based existence test to $f$ without occurrence grouping.

The contractors used in both cases are the same: 3BCID [12] and Interval Newton.
Problem & 3BCID & ¬OG & OG & Problem & 3BCID & ¬OG & OG \\
--- & --- & --- & --- & --- & --- & --- & --- \\
brent-10 & 18.9 & 19.5 & 19.1 & butcher-bis & 351 & 360 & 340 \\
 & 3941 & 3941 & 3941 & & 228305 & 228303 & 228245 \\
caprasse & 2.51 & 2.56 & 2.56 & fourbar & 13576 & 6742 & 1091 \\
 & 1305 & 1301 & 1301 & & 8685907 & 4278767 & 963113 \\
hayes & 39.5 & 41.1 & 40.7 & geneig & 593 & 511 & 374 \\
 & 17701 & 17701 & 17701 & & 205087 & 191715 & 158927 \\
i5 & 55.0 & 56.3 & 56.7 & pramanik & 100 & 66.6 & 37.2 \\
 & 10645 & 10645 & 10645 & & 124661 & 98971 & 69271 \\
katsura-12 & 74.1 & 74.5 & 75.0 & trigexp2-11 & 82.5 & 87.0 & 86.7 \\
 & 4317 & 4317 & 4317 & & 14287 & 14287 & 14287 \\
kin1 & 1.72 & 1.77 & 1.77 & trigol-10 & 152 & 155 & 156 \\
 & 85 & 85 & 85 & & 2691 & 2691 & 2691 \\
eco9 & 12.7 & 13.5 & 13.2 & virasoro-bis & 21.1 & 21.5 & 19.8 \\
 & 6203 & 6203 & 6203 & & 2781 & 2781 & 2623 \\
redeco8 & 5.61 & 5.71 & 5.66 & yamamura-8 & 9.67 & 10.04 & 9.86 \\
 & 2295 & 2295 & 2295 & & 2883 & 2883 & 2883 \\

Table 1. Experimental results using the monotonicity-based existence test. The first and fifth columns indicate the name of each instance, the second and sixth columns yield the CPU time (above) and the number of nodes (below) obtained on an Intel 6600 2.4 GHz by a strategy based on 3BCID. The third and seventh columns report the results obtained by the strategy using a (standard) monotonicity-based existence test and 3BCID. Finally, the fourth and eighth columns report the results of our strategy using an existence test based on occurrence grouping and 3BCID.

From these first results we can observe that only in three benchmarks OG is clearly better than ¬OG (fourbar, geneig and pramanik). In the other ones, the evaluation by occurrence grouping seems to be useless. Indeed, in most of the benchmarks, the existence test based on occurrence grouping does not cut branches in the search tree. However, note that it does not require additional time w.r.t. ¬OG. This clearly shows that the time required by occurrence grouping is negligible.

### 6.2 Occurrence Grouping inside a monotonicity-based contractor

Mohc [1] is a new constraint propagation contractor (like HC4 or Box) that uses the monotonicity of a function to improve the contraction/filtering of the related variables. Called inside a propagation algorithm, the Mohc-revise(f) procedure improves the filtering obtained by HC4-revise(f) by mainly achieving two additional calls to HC4-revise(fmin ≤ 0) and HC4-revise(fmax ≥ 0), where fmin and fmax correspond to the functions used when the evaluation by monotonicity calculates the lower and upper bounds of f. It also performs a monotonic version of the BoxNarrow procedure used by Box [2].
Table 2 shows the results of Mohc without the OG algorithm (¬OG), and with Occurrence Grouping (OG), i.e., when the function \( f \) is transformed into \( f^{og} \) before applying Mohc-revise(\( f^{og} \)).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mohc</th>
<th></th>
<th></th>
<th></th>
<th>Problem</th>
<th>Mohc</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>¬OG</td>
<td>OG</td>
<td>#OG calls</td>
<td>butcher-bis</td>
<td></td>
<td>¬OG</td>
<td>OG</td>
<td>#OG calls</td>
<td></td>
</tr>
<tr>
<td>brent-10</td>
<td>20</td>
<td>20.3</td>
<td>30867</td>
<td>220.64</td>
<td>7.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>caprasse</td>
<td>2.57</td>
<td>2.71</td>
<td>60073</td>
<td>4277.95</td>
<td>385.62</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hayes</td>
<td>17.62</td>
<td>17.45</td>
<td>5316</td>
<td>328.34</td>
<td>111.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i5</td>
<td>57.25</td>
<td>58.12</td>
<td>835130</td>
<td>67.98</td>
<td>21.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>katsura-12</td>
<td>100</td>
<td>103</td>
<td>39659</td>
<td>90.57</td>
<td>88.24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kin1</td>
<td>1.82</td>
<td>1.79</td>
<td>316</td>
<td>137.27</td>
<td>57.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eco9</td>
<td>13.31</td>
<td>13.96</td>
<td>70499</td>
<td>18.95</td>
<td>3.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redeco8</td>
<td>5.98</td>
<td>6.12</td>
<td>56312</td>
<td>11.59</td>
<td>2.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mohc</th>
<th></th>
<th></th>
<th></th>
<th>Problem</th>
<th>Mohc</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3811</td>
<td>3805</td>
<td></td>
<td>3805</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>caprasse</td>
<td>1251</td>
<td>867</td>
<td></td>
<td>60073</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hayes</td>
<td>4599</td>
<td>4415</td>
<td></td>
<td>5316</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i5</td>
<td>10399</td>
<td>9757</td>
<td></td>
<td>835130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kin1</td>
<td>3711</td>
<td>3625</td>
<td></td>
<td>39659</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eco9</td>
<td>6161</td>
<td>6025</td>
<td></td>
<td>70499</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redeco8</td>
<td>2285</td>
<td>2209</td>
<td></td>
<td>56312</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Experimental results using Mohc. The first and fifth columns indicate the name of each instance, the second and sixth columns report the results obtained by the strategy using 3BCID(Mohc) without OG. The third and seventh columns report the results of our strategy using 3BCID(OG+Mohc). The fourth and eighth columns indicate the number of calls to Occurrence Grouping.

We observe that, for 7 of the 16 benchmarks, Occurrence Grouping is able to improve the results of Mohc; in butcher-bis, fourbar, virasoro-bis and yamamura-8 the gains in CPU time (¬OG) obtained are 30, 11, 5.6 and 5.4 respectively.

7 Conclusion

We have proposed a new method to improve the monotonicity-based evaluation of a function \( f \). This Occurrence Grouping method creates for each variable three auxiliary, respectively increasing, decreasing and non monotonic variables in \( f \). It then transforms \( f \) into a function \( f^{og} \) that groups the occurrences of a variable into these auxiliary variables. As a result, the evaluation by occurrence grouping of \( f \), i.e., the evaluation by monotonicity of \( f^{og} \), is better than the evaluation by monotonicity of \( f \).
Occurrence grouping shows good performances when it is used to improve the monotonicity-based existence test, and when it is embedded in a contractor algorithm, called Mohc, that exploits monotonicity of functions.

References