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

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*, ...). 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:

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

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:

$$[f]_{opt}([B]) \subseteq [f]_m([B]) = [\underline{[f]_n(\underline{[x_1]}, \overline{[x_2]}, [x_3])}, \overline{[f]_n(\overline{[x_1]}, \underline{[x_2]}, [x_3])}] \subseteq [f]_n([B])$$

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 ($4x$).

$[f_1]_{opt}([B])$ is $[-3.05786, 7]$, 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^{nog}, 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 $f_1^{nog}(x_a, x_b, x_c) = -x_b^3 + 2x_c^2 + 6x_a$. The evaluation by monotonicity of f_1^{nog} 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.,
$\underline{[f_1^{nog}]_m([-1.2, 1])} = \underline{[f_1^{nog}]_n(-1.2, 1, [-1.2, 1])} = \underline{-1^3 + 2[-1.2, 1]^2 - 7.2} = -8.2$

(resp. $\overline{[f_1^{nog}]_m([-1.2,1])} = 10.608$). Finally, the evaluation by monotonicity is $[f_1^{nog}]_m([-1.2,1]) = [-8.2, 10.608]$.

It appears that the evaluation by monotonicity of the new function f^{nog} 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: $f_1^{og}(x_a, x_b) = -x_b^3 + 2x_a^2 + 6x_a$, where f_1^{og} 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 $f_1^{og}(x_a, x_c) = -x_a^3 + 2x_c^2 + 6x_a$, 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 $f^{og}(x_a, x_b, x_c)$ by replacing in f every occurrence of x by one of the three variables x_a, x_b, x_c, such that f^{og} is increasing w.r.t. x_a in $[x]$, and f^{og} is decreasing w.r.t. x_b in $[x]$.

Then, we define the *interval extension by occurrence grouping* of f by:

$$[f]_{og}([B]) := [f^{og}]_m([B])$$

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^{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 $[f]_{og}$. 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: $\{x_1, x_2, ..., x_k\}$. $f^\circ(x_1, .., x_k)$ is a function obtained from f considering all the occurrences of x as different variables. $[f]_n([B])$ computes $[f^\circ]_{opt}([B])$.*

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 $[f]_{opt}([B])$.*

Using these propositions, we observe that $[f^{og}]_m([x_a], [x_b], [x_c])$ is equivalent to $[f^\circ]_{opt}([x_a], [x_b], [x_{c_1}], ..., [x_{c_{ck}}])$, considering each occurrence of x_c in f^{og} as an independent variable x_{c_j} in f°. 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,*

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

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 $[f]_{og}([B]) = [f^{og}]_m([B]) = [f^\circ]_{opt}([B])$:

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

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 $[x] = [x_a] = [x_b] = [x_{c_1}] = ... = [x_{c_{ck}}]$.

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 $\underline{[g_a]([B])} \geq 0$ and $\overline{[g_b]([B])} \leq 0$, implying $|[g_a]([B])| = \overline{[g_a]([B])}$ and $|[g_b]([B])| = -\underline{[g_b]([B])}$. Finally:

$$diam([f]_{og}([B])) \leq diam([x]) \Big(\overline{[g_a]([B])} - \underline{[g_b]([B])} + \sum_{i=1}^{ck} |[g_{c_i}]([B])| \Big) \tag{2}$$

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

$$G = \overline{[g_a]([B])} - \underline{[g_b]([B])} + \sum_{i=1}^{k} \big(|[g_i]([B])| r_{c_i} \big) \tag{3}$$

subject to:

$$\underline{[g_a]([B])} \geq 0 \tag{4}$$

$$\overline{[g_b]([B])} \leq 0 \tag{5}$$

$$r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1, ..., k \tag{6}$$

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

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_i}, r_{b_i} and r_{c_i} that minimize

$$\begin{aligned} G &= \sum_{i=1}^{3} \overline{[g_i]([B])} r_{a_i} - \sum_{i=1}^{3} \underline{[g_i]([B])} r_{b_i} + \sum_{i=1}^{3} \big(|[g_i]([B])| r_{c_i} \big) \\ &= (4 r_{a_2} + 6 r_{a_3}) + (4.32 r_{b_1} + 4.8 r_{b_2} - 6 r_{b_3}) + (4.32 r_{c_1} + 4.8 r_{c_2} + 6 r_{c_3}) \end{aligned}$$

subject to:

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

$$\sum_{i=1}^{3} \overline{[g_i]([B])} r_{b_i} = 4 r_{b_2} + 6 r_{b_3} \leq 0$$

$$r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1, ..., 3$$

$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0, 1\} \quad \text{for } i = 1, ..., 3$$

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 $diam([f]_{og}([B]))$ is equal to 23.76 ($10.8 \times diam[-1.2, 1]$)) whereas $diam([f]_{og}([B])) = 13.352$. Although the overestimation is quite rough, the heuristic works well on this example. Indeed, $diam([f]_n([B])) = 18.808$, and $diam([f]_{opt}([B])) = 10.06$.

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_i}, r_{b_i} and r_{c_i} 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^{og} being increasing w.r.t. x_a, and decreasing w.r.t. x_b. Each occurrence x_i is replaced in f^{og} by $r_{a_i}x_a + r_{b_i}x_b + r_{c_i}x_c$, with $r_{a_i} + r_{b_i} + r_{c_i} = 1$, $\frac{\partial f^{og}}{\partial x_a} \geq 0$ and $\frac{\partial f^{og}}{\partial x_b} \leq 0$. We can then remark that f and f^{og} have the same natural evaluation.

In Example 1, we can replace f_1 by f^{og_1} or f^{og_2} in a way respecting the monotonicity constraints of x_a and x_b. Considering the interval $[x] = [-1.2, 1]$:

1. $f_1^{og_1}(x_a, x_b) = -(\frac{5}{18}x_a + \frac{13}{18}x_b)^3 + 2x_a^2 + 6x_a$: $[f_1^{og_1}]_m([x]) = [-4.38, 8.205]$
2. $f_1^{og_2}(x_a, x_b, x_c) = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$: $[f_1^{og_2}]_m([x]) = [-5.472, 7]$

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 $[f_2]_{opt}([x])$ 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. $f_2^{og_1}(x_a, x_b) = x_a^3 - (\frac{1}{4}x_a + \frac{3}{4}x_b)$: $[f_2^{og_1}]_m([x]) = [-0.75, 6.375]$
2. $f_2^{og_2}(x_a, x_b) = (\frac{11}{12}x_a + \frac{1}{12}x_b)^3 - x_b$: $[f_2^{og_2}]_m([x]) = [-1.756, 6.09]$

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

Find the values r_{a_i}, r_{b_i} and r_{c_i} for each occurrence x_i that minimize (3) subject to (4), (5), (6) and

$$r_{a_i}, r_{b_i}, r_{c_i} \in [0, 1] \quad \text{for } i = 1, ..., k. \tag{7}$$

Linear program corresponding to Example 1

In this example we obtain the minimum 10.58 and the new function $f_1^{og}(x_a, x_b, x_c) = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$: $[f_1^{og}]_m([x]) = [-5.472, 7]$. 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 $f_2^{og}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$. 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_i}, r_{b_i}, r_{c_i} (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 $[g_*]$ 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 $[g_*]$ is denoted by $[g_i]$ hereafter, instead of $[g_i]([B])$, i.e., $[g_i]$ is the interval $\frac{\partial f}{\partial x_i}([B])$.

Algorithm 1 `Occurrence_Grouping`(**in:** f, $[g_*]$ **out:** f^{og})

```
1: $[G_0] \leftarrow \sum_{i=1}^{k} [g_i]$
2: $[G_m] \leftarrow \sum_{0 \notin [g_i]} [g_i]$
3: if $0 \notin [G_0]$ then
4:     OG_case1($[g_*], [r_{a_*}], [r_{b_*}], [r_{c_*}]$)
5: else if $0 \in [G_m]$ then
6:     OG_case2($[g_*], [r_{a_*}], [r_{b_*}], [r_{c_*}]$)
7: else
8:     /* $0 \notin [G_m]$ and $0 \in [G_0]$ */
9:     if $[G_m] \geq 0$ then
10:        OG_case3$^+$($[g_*], [r_{a_*}], [r_{b_*}], [r_{c_*}]$)
11:    else
12:        OG_case3$^-$($[g_*], [r_{a_*}], [r_{b_*}], [r_{c_*}]$)
13:    end if
14: end if
15: $f^{og} \leftarrow$ Generate_New_Function($f, [r_{a_*}], [r_{b_*}], [r_{c_*}]$)
```

An asterisk (*) in the index of a symbol represents a vector (e.g., $[g_*]$, $[r_{a_*}]$).

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 ($0 \notin [G_0]$ in line 3) and occurs when x is a monotonic variable. The procedure `OG_case1` does not achieve any occurrence grouping: *all the occurrences* of x 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^{og} is equivalent to the evaluation by monotonicity of f.

In the second case, when $0 \in [G_m]$ (line 5), the procedure `OG_case2` (Algorithm 2) performs a grouping of the occurrences of x. Increasing occurrences are replaced by $(1 - \alpha_1)x_a + \alpha_1 x_b$, decreasing occurrences by $\alpha_2 x_a + (1 - \alpha_2)x_b$ and non monotonic occurrences by x_c (lines 7 to 13 of Algorithm 2). f_2 falls in this case: $\alpha_1 = \frac{1}{45}$ 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: $f_2^{og}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$.

Algorithm 2 `OG_case2`(**in:** $[g_*]$ **out:** $[r_{a_*}], [r_{b_*}], [r_{c_*}]$)

```
 1: $[G^+] \leftarrow \sum_{[g_i] \geq 0} [g_i]$
 2: $[G^-] \leftarrow \sum_{[g_i] \leq 0} [g_i]$
 3: $[\alpha_1] \leftarrow \frac{\underline{[G^+]}\,\overline{[G^-]} + \overline{[G^-]}\,\underline{[G^-]}}{\underline{[G^+]}\,\overline{[G^-]} - \underline{[G^-]}\,\overline{[G^+]}}$
 4: $[\alpha_2] \leftarrow \frac{\underline{[G^+]}\,\overline{[G^+]} + \overline{[G^-]}\,\underline{[G^+]}}{\underline{[G^+]}\,\overline{[G^-]} - \underline{[G^-]}\,\overline{[G^+]}}$
 5:
 6: for all $[g_i] \in [g_*]$ do
 7:   if $\underline{[g_i]} \geq 0$ then
 8:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow (1 - [\alpha_1], [\alpha_1], 0)$
 9:   else if $\overline{[g_i]} \leq 0$ then
10:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow ([\alpha_2], 1 - [\alpha_2], 0)$
11:   else
12:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow (0, 0, 1)$
13:   end if
14: end for
```

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 `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 *index*[1] (line 7) as long as the constraint $\sum_{i=1}^{k} r_{a_i}\underline{[g_i]} \geq 0$ is satisfied (lines 9-13). The first non monotonic occurrence $x_{i'}$ that cannot be replaced because it would make the constraint unsatisfiable is replaced by $\alpha x_a + (1-\alpha)x_c$, with α such that the constraint is satisfied and equal to 0, i.e., $(\sum_{i=1, i\neq i'}^{k} r_{a_i}\underline{[g_i]}\ ) + \alpha\underline{[g_{i'}]} = 0$ (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: $\underline{[g_1]}+\underline{[g_3]}+\alpha\underline{[g_2]} = 0$. The new function is: $f_1^{og} = -x_a^3+2(0.35x_a+0.65x_c)^2+6x_a$.

Algorithm 3 `OG_case3`$^+$**(in:** $[g_*]$ **out:** $[r_{a_*}], [r_{b_*}], [r_{c_*}]$)

```
1: $[g_a] \leftarrow [0,0]$
2: **for all** $[g_i] \in [g_*]$, $0 \notin [g_i]$ **do**
3:     $[g_a] \leftarrow [g_a] + [g_i]$ /*All positive and negative derivatives are absorbed by $[g_a]$ */
4:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow (1,0,0)$
5: **end for**
6:
7: $index \leftarrow$ ascending_sort($\{[g_i] \in [g_*], 0 \in [g_i]\}$, criterion $\rightarrow |\overline{[g_i]}/\underline{[g_i]}|$)
8: $j \leftarrow 1$ ; $i \leftarrow index[1]$
9: **while** $\underline{[g_a]} + \underline{[g_i]} \geq 0$ **do**
10:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow (1,0,0)$
11:     $[g_a] \leftarrow [g_a] + [g_i]$
12:     $j \leftarrow j+1$ ; $i \leftarrow index[j]$
13: **end while**
14:
15: $[\alpha] \leftarrow -\frac{\underline{[g_a]}}{\underline{[g_i]}}$
16: $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow ([\alpha], 0, 1-[\alpha])$
17: /* $[g_a] \leftarrow [g_a] + [\alpha][g_i]$ */
18: $j \leftarrow j+1$ ; $i \leftarrow index[j]$
19:
20: **while** $j \leq$ length($index$) **do**
21:     $([r_{a_i}], [r_{b_i}], [r_{c_i}]) \leftarrow (0,0,1)$
22:     $j \leftarrow j+1$ ; $i \leftarrow index[j]$
23: **end while**
```

[1] An occurrence x_{i_1} is handled before x_{i_2} if $|\overline{[g_{i_1}]}/\underline{[g_{i_1}]}| \leq |\overline{[g_{i_2}]}/\underline{[g_{i_2}]}|$. *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 ($\neg$`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	trigo1-10	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	yamamura1-8	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`($f_{min} \leq 0$) and `HC4-revise`($f_{max} \geq 0$), where f_{min} and f_{max} 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}).

Problem	Mohc			**Problem**	Mohc		
	¬OG	OG	#OG calls		¬OG	OG	#OG calls
brent-10	20	20.3		butcher-bis	220.64	**7.33**	
	3811	3805	30867		99033	2667	111045
caprasse	2.57	2.71		fourbar	4277.95	**385.62**	
	1251	867	60073		1069963	57377	8265730
hayes	17.62	17.45		geneig	328.34	**111.43**	
	4599	4415	5316		76465	13705	2982275
i5	57.25	58.12		pramanik	67.98	**21.23**	
	10399	9757	835130		51877	12651	395083
katsura-12	100	103		trigexp2-11	90.57	88.24	
	3711	3625	39659		14299	14301	338489
kin1	1.82	1.79		trigo1-10	137.27	**57.09**	
	85	83	316		1513	443	75237
eco9	13.31	13.96		virasoro-bis	18.95	**3.34**	
	6161	6025	70499		2029	187	241656
redeco8	5.98	6.12		yamamura1-8	11.59	**2.15**	
	2285	2209	56312		2663	343	43589

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 butcher8-bis, fourbar, virasoro-bis and yamamura-8 the gains in CPU time ($\frac{\neg\texttt{OG}}{\texttt{OG}}$) 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

1. Ignacio Araya, Bertrand Neveu, and Gilles Trombettoni. An Interval Constraint Propagation Algorithm Exploiting Monotonicity. In *Workshop INTCP*, 2009.
2. Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.
3. Gilles Chabert. `www.ibex-lib.org`, 2009.
4. Gilles Chabert and Luc Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
5. Gilles Chabert and Luc Jaulin. Hull Consistency Under Monotonicity. In *Proc. Constraint Programming CP, LNCS 5732*, 2009.
6. Eldon Hansen and G. William Walster. *Global Optimization using Interval Analysis.* CRC Press, 2^{nd} edition, 2003.
7. William G. Horner. A new Method of Solving Numerical Equations of all Orders, by Continuous Approximation. *Philos. Trans. Roy. Soc. London*, 109:308–335, 1819.
8. Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis.* Springer, 2001.
9. Jean-Pierre Merlet. `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`, 2009.
10. Ramon Moore. *Interval Analysis.* Prentice Hall, 1966.
11. Arnold Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, 1990.
12. Gilles Trombettoni and Gilles Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.