Computing manuscript No.
(will be inserted by the editor)

An Interval Extension Based on Occurrence Grouping

Ignacio Araya · **Bertrand Neveu** ·
Gilles Trombettoni

Accepted: 12 October 2011

Abstract In interval arithmetics, special care has been brought to the definition of interval extension functions that compute narrow interval images. In particular, 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 a given domain, we try to transform f into a new function f^{og} that is monotonic w.r.t. 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 a linear program and an algorithm that minimize a Taylor-based over-estimate of the image diameter of $[f]_{og}$. Experiments show the benefits of this new interval extension for solving systems of nonlinear equations.

Keywords Intervals · Interval extension · Monotonicity · Occurrence grouping

1 Introduction

The computation of sharp interval image enclosures is in the heart of interval arithmetics [14]. 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 combinatorial 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, used at every node of the search tree to reduce the search space, can be improved when they use better interval extensions. For instance, the `Box` constraint propagation algorithm [3] uses a *test of existence* inside its iterative splitting process.

This paper proposes a new interval extension and we first recall basic material about interval arithmetics [14, 15, 10] to introduce the interval extensions useful in our work.

An interval $[x] = [a,b]$ is the set of real numbers between a and b. $\underline{x} = a$ denotes the minimum of $[x]$ and $\overline{x} = b$ denotes the maximum of $[x]$. The diameter/width of an interval is:

I. Araya, UTFSM, Valparaiso, Chile. E-mail: iaraya@inf.utfsm.cl
B. Neveu, Imagine LIGM Univ. Paris-Est, Champs-sur-Marne, France. E-mail: Bertrand.neveu@enpc.fr
G. Trombettoni, INRIA, I3S, Univ. Nice-Sophia, Sophia-Antipolis, France.
E-mail: Gilles.Trombettoni@inria.fr

$diam([x]) = \overline{x} - \underline{x}$, 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 a vector. $[V] = \{[x_1], [x_2], ..., [x_n]\}$. (Vectorial variables appear in upper case in this article.)

An *interval function* $[f]$ is a function from $\mathbb{IR}$ to $\mathbb{IR}^n$, $\mathbb{IR}$ being the set of all the intervals over $\mathbb{R}$. 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 1 (Extension of a function to $\mathbb{IR}$; also called *inclusion function*)
Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
$[f] : \mathbb{IR}^n \rightarrow \mathbb{IR}$ is an **extension** *of f to intervals iff:*

$$\begin{array}{ll} \forall [V] \in \mathbb{IR}^n & [f]([V]) \supseteq \Im f([V]) \equiv \{f(V),\ V \in [V]\} \\ \forall V \ \ \in \mathbb{R}^n & f(V) = [f](V) \end{array}$$

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 real function f simply replaces arithmetic over the reals by interval arithmetic.

The *optimal* image $[f]_{opt}([V])$ is the sharpest interval containing $\Im f([V])$. If f is continuous inside a box $[V]$, the natural evaluation of f (i.e., the computation of $[f]_n([V])$) yields the optimal image when each variable occurs only once in f. When a variable appears several times in f, the evaluation by interval arithmetics generally produces an over-estimate of $[f]_{opt}([V])$, 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 $[x] = [0,1]$ gives the result $[-1,1]$, instead of $[0,0]$, as does $[x]-[y]$, with $[x] = [0,1]$ and $[y] = [0,1]$. Thus, multiple occurrences of variables render NP-hard the computation of the *optimal* image of a polynomial [11]. This main drawback of interval arithmetics causes a real difficulty for implementing efficient interval-based solvers. There exist many possible interval extensions for a function, the difficulty being to define an extension that computes a sharp approximation of the optimal image at a low cost.

The first-order *Taylor extension* $[f]_t$ of f, also called centered form [14], uses the Taylor form of f:

$$[f]_t([V]) = f(V_m) + \sum_{i=1}^{n} \left(\left[\frac{\partial f}{\partial x_i} \right]([V]) \ . \ ([x_i] - x_i^m) \right)$$

where n is the number of variables in f, x_i^m is the value in the middle of the interval $[x_i]$, V_m is the n-dimensional point in the middle of $[V]$ (i.e., $V_m = (x_1^m, ..., x_k^m)$) and $\left[\frac{\partial f}{\partial x_i}\right]$ is an interval extension of $\frac{\partial f}{\partial x_i}$. The Taylor extension generally calculates sharp evaluations when the diameters of the partial derivatives are close to 0. In other cases it can be even worse than the natural extension.

A well-known variant of the Taylor extension, called here *Hansen extension* $[f]_h$, computes a sharper image at a higher cost [8]: $[f]_h([V]) \subseteq [f]_t([V])$.

Another extension to intervals uses the monotonicity of a function in a given domain. When f is monotonic w.r.t. a subset of variables, one can replace, in the natural evaluations, the intervals of these monotonic variables[1] by degenerated intervals reduced to their maximal or minimal values [14, 8].

[1] For the sake of conciseness, we sometimes write that a "variable x is monotonic" instead of writing that f is monotonic w.r.t. x.

Definition 2 *(* f_{min}, f_{max}, **monotonicity-based extension***)*
Let f be a function defined on variables V of domains $[V]$. Let $X \subseteq V$ be a subset of monotonic variables. Consider the values x_i^+ and x_i^- such that: if $x_i \in X$ is an increasing (resp. decreasing) variable, then $x_i^- = \underline{x_i}$ and $x_i^+ = \overline{x_i}$ (resp. $x_i^- = \overline{x_i}$ and $x_i^+ = \underline{x_i}$).

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

$$f_{min}(W) = f(x_1^-,...,x_n^-,W)$$
$$f_{max}(W) = f(x_1^+,...,x_n^+,W)$$

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

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

The image $[f]_m([V])$ obtained by the monotonicity-based extension is sharper than, or equal to, the image $[f]_n([V])$ obtained by natural evaluation. That is:

$$[f]_{opt}([V]) \subseteq [f]_m([V]) \subseteq [f]_n([V])$$

In addition, when a function is monotonic w.r.t. each of its variables, i.e., when W is empty in Def. 2, the problem of multiple occurrences disappears and the evaluation (using a monotonicity-based extension) becomes optimal: $[f]_m([V]) = [f]_{opt}([V])$.

Note that the bounds of the evaluation by monotonicity can be computed using any interval extension, not necessarily the natural extension. When the bounds are computed with the Taylor (resp. Hansen) extension, we denote this variant by $[f]_{m+t}$ (resp. $[f]_{m+h}$).

When the evaluation by monotonicity uses, recursively, the same evaluation by monotonicity for computing the bounds of the image, we call it *recursive evaluation by monotonicity* and denote it by $[f]_{mr}$. This computes images sharper than or equal to the evaluation by monotonicity does (i.e., $[f]_{mr}([V]) \subseteq [f]_m([V])$) at a cost between 2 and $2n$ higher. The extended paper illustrates this interval extension [1].

Contribution

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. a *subgroup of occurrences* of x. We present in the next section the idea of grouping the occurrences into three sets, increasing, decreasing and non monotonic auxiliary variables. 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 7, experiments show that this new occurrence grouping interval extension function compares favorably with existing ones and show its benefits for solving systems of equations, in particular when we use a filtering algorithm like `Mohc` [2,6] 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 $f_1(x) = -x^3 + 2x^2 + 14x$. We want to calculate a sharp evaluation of this function when x falls in $[-2,1]$. The derivative of f_1 is $f_1'(x) = -3x^2 + 4x + 14$ and contains a positive term (14), a negative term $(-3x^2)$ and the term $4x$ that is negative when $x \in [-2,0]$ and positive when $x \in [0,1]$. $[f_1]_{opt}([V])$ is $[-13.18, 15]$, but we cannot obtain it directly by a simple interval function evaluation (one needs to solve $f_1'(x) = 0$, which is difficult in the general case). In the interval $[-2,1]$, f_1 is not monotonic. The natural interval evaluation yields $[-29,30]$, the Horner evaluation $[-34,17]$ (see [9]).

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 leads to replace the function f by a function f^{nog}, grouping 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: $[x_a] = [x_b] = [x_c] = [x]$. However, the evaluation by monotonicity of the new function f^{nog} always provides the same result as the natural evaluation. The main idea is then to change this grouping in order to reduce the dependency problem and obtain sharper evaluations. *We can indeed group some occurrences (increasing, decreasing, or non monotonic) into an increasing variable x_a (resp. a decreasing variable x_b) as long as the function remains increasing (resp. decreasing) w.r.t. this variable x_a (resp. x_b). If one can move a non monotonic occurrence into a monotonic group, the evaluation will be better (or remain the same).* 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: $f_1^{og}(x_a,x_b) = -x_b^3 + 2x_a^2 + 14x_a$. As the interval derivative of the grouping of the first two occurrences (the variable x_a) is positive: $4[x] + 14 = [6,18]$, f_1^{og} is increasing w.r.t. x_a. We can then achieve the evaluation by monotonicity and obtain the interval $[-21,24]$. We can in the same manner obtain $f_1^{og}(x_a,x_c) = -x_a^3 + 2x_c^2 + 14x_a$, the evaluation by monotonicity yields then $[-20,21]$. We remark that we find sharper images than the natural evaluation of f_1 does.

Interval extension by occurrence grouping

Consider the function $f(x)$ with multiple occurrences of x. We obtain a 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: $[f]_{og}([V]) := [f^{og}]_m([V])$

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. Using the Taylor extension, we first compute an over-estimate of the diameter of the image computed by $[f]_{og}$. Then, we propose a linear program performing a grouping that minimizes this over-estimate.

3.1 Taylor-based over-estimate

First, 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 do.

Proposition 1 *([14]) Let $f(x)$ be a continuous function in a box $[V]$ 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 by considering all the occurrences of x as different variables. Then, $[f]_n([V]) = [f^\circ]_{opt}([V])$.*

Second, as f^{og} is monotonic w.r.t. x_a and x_b, the evaluation by monotonicity of these variables is optimal.

Proposition 2 *([14]) Let $f(x_1,...,x_n)$ be a monotonic function w.r.t. each of its variables in a box $[V] = \{[x_1],...,[x_n]\}$. Then, the evaluation by monotonicity computes $[f]_{opt}([V])$.*

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°, ck being the number of occurrences of x_c in f^{og}. Using the Taylor evaluation, an upper bound of $diam([f]_{opt}([V]))$ is given by the right side of (1) in Proposition 3.

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

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

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

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

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}: $\frac{\partial f^{og}}{\partial x_a}([V]) \geq 0$, $\frac{\partial f^{og}}{\partial x_b}([V]) \leq 0$, we have the sufficient conditions $\underline{[g_a]([V])} \geq 0$ and $\overline{[g_b]([V])} \leq 0$, implying $|[g_a]([V])| = \overline{[g_a]([V])}$ and $|[g_b]([V])| = -\underline{[g_b]([V])}$. Finally:

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

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

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

subject to:

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

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

$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0,1\};\quad r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1,...,k, \tag{6}$$

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, $[g_a]([V]) = \sum_{i=1}^{k} [g_i]([V]) r_{a_i}$, $[g_b]([V]) = \sum_{i=1}^{k} [g_i]([V]) r_{b_i}$, and $[g_1]([V]),...,[g_k]([V])$ are the derivatives w.r.t. each occurrence. We can remark that $[g_a]([V])$ and $[g_b]([V])$ are calculated using only the derivatives of f w.r.t. each occurrence of x (i.e., $[g_i]([V])$).

Linear program corresponding to Example 1

We have $f_1(x) = -x^3 + 2x^2 + 14x$, $f_1'(x) = -3x^2 + 4x + 14$ with $x \in [-2, 1]$. The gradient is: $[g_1]([-2,1]) = [-12, 0]$, $[g_2]([-2,1]) = [-8, 4]$ and $[g_3]([-2,1]) = [14, 14]$. Then, the linear program is:

Find the values r_{a_i}, r_{b_i} and r_{c_i} that minimize

$$\begin{aligned} G &= \sum_{i=1}^{3} \overline{[g_i]([V])} r_{a_i} - \sum_{i=1}^{3} \underline{[g_i]([V])} r_{b_i} + \sum_{i=1}^{3} \left(|[g_i]([V])| r_{c_i} \right) \\ &= (4r_{a_2} + 14r_{a_3}) + (12r_{b_1} + 8r_{b_2} - 14r_{b_3}) + (12r_{c_1} + 8r_{c_2} + 14r_{c_3}) \end{aligned}$$

subject to:

$$\sum_{i=1}^{3} \underline{[g_i]([V])} r_{a_i} = -12r_{a_1} - 8r_{a_2} + 14r_{a_3} \geq 0; \;\; \sum_{i=1}^{3} \overline{[g_i]([V])} r_{b_i} = 4r_{b_2} + 14r_{b_3} \leq 0$$
$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0, 1\}; \quad r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1, ..., 3$$

We obtain the minimum 22, 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. Note that the value of the over-estimate of $diam([f]_{og}([V]))$ is equal to 66 ($22 * diam[-2, 1]$) whereas $diam([f]_{og}([V])) = 41$. Although the over-estimate is rough, the heuristic works rather well on this example. Indeed, $diam([f]_n([V])) = 59$ and $diam([f]_{opt}([V])) = 28.18$.

4 A linear programming problem achieving a better occurrence grouping

The linear program above is a 0,1 linear program and is known to be NP-hard in general. We can render it tractable while, more important in practice, improving the minimum G 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 such that f^{og} is increasing w.r.t. x_a and decreasing w.r.t. x_b in $[x]$.

Definition 3 *(***Interval extension by occurrence grouping***)*
Let $f(x)$ be a function with multiple occurrences of the variable x. $f^{og}(x_a, x_b, x_c)$ is the function obtained by replacing in f every occurrence of x by $r_{a_i}x_a + r_{b_i}x_b + r_{c_i}x_c$, such that:

- $r_{a_i}, r_{b_i}, r_{c_i} \in [0,1]^3$ *and* $r_{a_i} + r_{b_i} + r_{c_i} = 1$,
- $\frac{\partial f^{og}}{\partial x_a}([x],[x],[x]) \geq 0$ *and* $\frac{\partial f^{og}}{\partial x_b}([x],[x],[x]) \leq 0$.

The interval extension by occurrence grouping of f is defined by $[f]_{og}([x]) := [f^{og}]_m([x],[x],[x])$

Note 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:

1. $f_1^{og_1}(x_a, x_b) = -(\frac{1}{2}x_a + \frac{1}{2}x_b)3 + 2x_a2 + 14x_a \quad \rightarrow \quad [f_1^{og_1}]_m([-2,1]) = [-19.875, 16.125]$
2. $f_1^{og_2}(x_a, x_b, x_c) = -x_a^3 + 2(0.25x_a + 0.75x_c)^2 + 14x_a \quad \rightarrow \quad [f_1^{og_2}]_m([-2,1]) = [-20, 16.125]$

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 (among others).

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

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** $\underline{G_m} \geq 0$ **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`$(f,[r_{a_*}],[r_{b_*}],[r_{c_*}])$

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

$[G^+] \leftarrow \sum_{[g_i] \geq 0} [g_i]$
$[G^-] \leftarrow \sum_{[g_i] \leq 0} [g_i]$
$[\alpha_1] \leftarrow \frac{\underline{\underline{G^+}\overline{G^-} + \overline{G^-}\underline{G^-}}}{\overline{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}}}$
$[\alpha_2] \leftarrow \frac{\underline{\underline{G^+}\overline{G^+} + \overline{G^-}\underline{G^+}}}{\overline{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}}}$

for all $[g_i] \in [g_*]$ **do**
    if $\underline{g_i} \geq 0$ **then**
        $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (1-[\alpha_1],[\alpha_1],0)$
    else if $\overline{g_i} \leq 0$ **then**
        $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow ([\alpha_2],1-[\alpha_2],0)$
    else
        $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (0,0,1)$
    end if
end for

Thus, the new linear program that computes convex linear combinations for achieving occurrence grouping becomes:

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

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

Note that this continuous linear program improves the minimum of the objective function because the integrity conditions are relaxed.

Examples

In Example 1, we obtain the minimum 21 and the new function $f_1^{og}(x_a,x_b,x_c) = -x_a^3 + 2(0.25x_a + 0.75x_c)^2 + 14x_a$: $[f_1^{og}]_m([x]) = [-20, 16.25]$. The minimum 21 is inferior to 22 (obtained by the 0,1 linear program). The evaluation by occurrence grouping of f_1 yields $[-20, 16.25]$, which is sharper than the image $[-20, 21]$ obtained by the 0,1 linear program presented in Section 3.

In Example 2, 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. In this case, the 0,1 linear program of Section 3 yields the naive grouping.

5 An efficient Occurrence Grouping algorithm

Algorithm 1 finds r_{a_i}, r_{b_i}, r_{c_i} that minimize G subject to the constraints. At line 15, the algorithm generates symbolically 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 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. Each component of $[g_*]$ is denoted by $[g_i]$ and corresponds to the interval $\left[\frac{\partial f}{\partial x_i}\right]([V])$. A symbol indexed by an asterisk refers to a vector (e.g., $[g_*]$, $[r_{a_*}]$).

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

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

At 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. At line 2, $[G_m]$ gets the value of the partial derivative of f w.r.t. the *monotonic* occurrences of x.

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. In the procedure `OG_case1`, *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) achieves 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). The third case occurs when $0 \notin [G_m]$ and $0 \in [G_0]$. W.l.o.g., assume that $\underline{G_m} \geq 0$. The procedure `OG_case3`$^+$ (Algorithm 3) first groups all the positive and negative occurrences in the increasing group x_a (lines 2–5). The non monotonic occurrences are then replaced by x_a in an order determined by an array *index*[2] (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 occurrence $x_{i'}$ that cannot be (entirely) replaced by x_a because it would make the constraint (4) 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 15–16). The rest of the occurrences are replaced by x_c (lines 18–22). The extended paper [1] illustrates how the procedure `OG_case3`$^+$ (resp. `OG_case2`) handles the example f_1 (resp. f_2).

[2] An occurrence x_{i_1} is handled before x_{i_2} if $\frac{\overline{g_{i1}} - |[g_{i1}]|}{\underline{g_{i1}}} \geq \frac{\overline{g_{i2}} - |[g_{i2}]|}{\underline{g_{i2}}}$. $index[j]$ yields the index of the j^{th} occurrence in this order.

6 Properties

Proposition 4 *Algorithm 1 (*`Occurrence_grouping`*) is correct and solves the linear program that minimizes (3), modulo floating-point roundings, subject to the constraints (4), (5), (6) and (7).*

We can check that Algorithm 1 respects the four constraints (4)–(7). We have also proven that the minimum of the objective function (3) is reached. The proof concerning `OG_case3` is sophisticated, due to the sort of indices, and uses known results about the continuous knapsack problem. Special care has been brought to ensure the correctness modulo floating-point roundings. A full proof can be found in [1].

Proposition 5 *The time complexity of* `Occurrence_Grouping` *for one variable with k occurrences is* $O(k\,log_2(k))$*. It is time* $O(n\,k\,log_2(k))$ *when a multi-variate function is iteratively transformed by* `Occurrence_Grouping` *for each of its n variables having at most k occurrences each.*

A preliminary gradient calculation by automatic differentiation is time $O(e)$*, where e is the number of unary and binary operators in the expression.*

Computing the gradient of a function amounts in two traversals of the tree representing the mathematical expression [3]. The time complexity of Algorithm 1 is dominated by that of `descending_sort` in the `OG_case3` procedure.

Instead of Algorithm 1, we may use a standard Simplex algorithm providing that the used Simplex implementation takes into account floating-point rounding errors. A performance comparison between Algorithm 1 and Simplex is shown in Section 7.3. Also, as shown in Section 7.2, the time required in practice by `Occurrence_Grouping` is negligible when it is used for solving systems of equations.

Although `Occurrence_Grouping` can be viewed as a heuristic since it minimizes a Taylor-based over-estimate of the function image diameter, it is important to stress that our new interval extension improves the well-known monotonicity-based interval extension.

Proposition 6 *Consider a function* $f : \mathbb{R}^n \rightarrow \mathbb{R}$*, and the previously defined interval natural (*$[f]_n$*), monotonicity-based (*$[f]_m$*) and occurrence grouping (*$[f]_{og}$*) extensions of f. Let V be the n variables involved in f with domains* $[V]$*. Then,* $[f]_{og}([V]) \subseteq [f]_m([V]) \subseteq [f]_n([V])$.

7 Experiments

`Occurrence_Grouping` has been implemented in the `Ibex` [5,4] open source interval-based solver in `C++`. The main goal of these experiments is to show the improvements in CPU time brought by `Occurrence_Grouping` when solving systems of equations.

We briefly recall the combinatorial process followed by an interval-based solver to find all the solutions of a system of equations. 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, 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 boxes of size smaller than a given positive ω.

Contraction algorithms comprise multidimensional interval Newton algorithms (or variants) issued from the numerical *interval analysis* community [14,15] along with algorithms from *constraint programming*. In all our experiments, at each node of the search tree, i.e., between two bisections, the solving strategy uses two types of *contractors* (all available in `Ibex`) in sequence:

1. (from constraint programming) the shaving/slicing contractor `3BCID` [12,16] and a recent *constraint propagation* algorithm, `Mohc` [2,6], exploiting monotonicity of functions;
2. (from interval analysis) an interval Newton using a Hansen-Sengupta matrix, a left-preconditioning of the matrix, and a Gauss-Seidel method to solve the interval linear system [8].

Sixteen systems of equations have been used in our experiments. They are issued from the COPRIN team website [13], most of them being also known in the COCONUT benchmark suite devoted to interval analysis and global optimization.[3] They correspond to square systems with a finite number of zero-dimensional (isolated) 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 COPRIN website). All experiments have been performed on a same computer (`Intel 6600 2.4 GHz`).

7.1 Comparison between interval extensions

We first report a comparison between the evaluation by occurrence grouping (i.e., the diameter of $[f]_{og}([V])$) and several existing interval evaluations ($[f]_{ext}$), including Taylor, Hansen and monotonicity-based extensions (see Section 1). Since the Taylor and Hansen extensions are not comparable with the natural extension, to obtain more reasonable comparisons, we have redefined $[f]_t([V]) = [f]'_t([V]) \cap [f]([V])$ and $[f]_h([V]) = [f]'_h([V]) \cap [f]_n([V])$, where $[f]'_t$ and $[f]'_h$ are the actual Taylor and Hansen extensions respectively.

Table 1 Different interval extensions compared to $[f]_{og}$

System	$[f]_n$	$[f]_t$	$[f]_h$	$[f]_m$	$[f]_{mr}$	$[f]_{mr+h}$	$[f]_{mr+og}$
`Brent`	0.857	0.985	0.987	0.997	0.998	0.999	1.000
`ButcherA`	0.480	0.742	0.863	0.666	0.786	0.963	1.028
`Caprasse`	0.602	0.883	0.960	0.856	0.953	1.043	1.051
`Direct kin.`	0.437	0.806	0.885	0.875	0.921	0.979	1.017
`Eco9`	0.724	0.785	0.888	0.961	0.980	0.976	1.006
`Fourbar`	0.268	0.718	0.919	0.380	0.427	1.040	1.038
`Geneig`	0.450	0.750	0.847	0.823	0.914	0.971	1.032
`Hayes`	0.432	0.966	0.974	0.993	0.994	0.998	1.001
`I5`	0.775	0.859	0.869	0.925	0.932	0.897	1.005
`Katsura`	0.620	0.853	0.900	0.993	0.999	0.999	1.000
`Kin1`	0.765	0.872	0.880	0.983	0.983	0.995	1.001
`Pramanik`	0.375	0.728	0.837	0.666	0.689	0.929	1.017
`Redeco8`	0.665	0.742	0.881	0.952	0.972	0.997	1.011
`Trigexp2`	0.904	0.904	0.904	0.942	0.945	0.921	1.002
`Trigo1`	0.483	0.766	0.766	0.814	0.814	0.895	1.000
`Virasoro`	0.479	0.738	0.859	0.781	0.795	1.025	1.062
`Yamamura1`	0.272	0.870	0.870	0.758	0.758	0.910	1.000
AVERAGE	0.564	0.822	0.888	0.845	0.874	0.973	1.016

The first column of Table 1 indicates the name of each instance. The other columns refer to existing extensions $[f]_{ext}$ and report an average of ratios $\rho_{ext} = \frac{Diam([f]_{og}([V]))}{Diam([f]_{ext}([V]))}$. These ratios

[3] See `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

are measured while the Branch & Contract solving strategy mentioned above[4] is run to solve the tested system of equations. They are calculated every time a constraint is handled inside the constraint propagation, *at every node of the search tree*, thus avoiding biases.

The table highlights that $[f]_{og}$ computes, in general, sharper interval images than all the competitors. (Only $[f]_{mr+og}$ achieves better evaluations, but it also uses occurrence grouping.) The improvements w.r.t. the two evaluation methods by monotonicity (i.e., $[f]_m$ and $[f]_{mr}$) corroborate the benefits of our approach. For example, in `Fourbar`, $[f]_{og}$ obtains an evaluation diameter which is 42.7% of the evaluation diameter provided by $[f]_{mr}$.

$[f]_{mr+h}$ obtains the sharpest evaluations in three benchmarks (`Caprasse`, `Fourbar` and `Virasoro`). However, $[f]_{mr+h}$ is more expensive than $[f]_{og}$. $[f]_{mr+h}$ requires computing $2n$ interval partial derivatives, thus traversing $4n$ times the expression tree if an automatic differentiation method is used [8].

The extension using occurrence grouping $[f]_{mr+og}$ provides necessarily a better evaluation than, or equal to, $[f]_{og}$. However, the experiments on the tested systems show that the gain in evaluation diameter is only 1.6% on average (between 0% and 6.2%), so that we do not believe it constitutes a promising extension due to its additional cost.

7.2 Occurrence Grouping inside a monotonicity-based contractor

These experiments are significant in that they underline the benefits of occurrence grouping for improving the solving of systems of constraints. `Mohc` [2,6] is a new constraint propagation contractor (like `HC4` or `Box` [3]) that exploits the monotonicity of a function to improve the contraction of the related variable intervals.

Table 2 shows the results of `Mohc` without the `OG` algorithm ($\neg$`OG`), and with `Occurrence-Grouping` (`OG`), i.e., when the function f is transformed into f^{og} before applying the main `MohcRevise(`f^{og}`)` procedure. We observe that, for 7 of the 16 benchmarks, `Occurrence-Grouping` is able to improve significantly the results of `Mohc`; in `Butcher`, `Fourbar`, `Virasoro` and `Yamamura1` the gains in CPU time ($\frac{\neg OG}{OG}$) obtained are greater than 30, 11, 7.5 and 5.4 respectively.

7.3 Practical time complexity

We have first compared the performance of two Occurrence Grouping implementations: using our ad-hoc algorithm (`Occurrence_Grouping`) and using a Simplex method. The basic Simplex implementation [7] we have used is not rigorous, i.e., it does not take into account rounding errors due to floating point arithmetic. Adding this feature should make the algorithm run more slowly. Two important results have been obtained. First, we have checked experimentally that our algorithm is correct, i.e., it obtains the minimum value for the objective function G. Second, just as we expected, the performance of the general-purpose Simplex method is worse than the performance of our algorithm. It runs between 2.32 (`Brent`) and 10 (`Virasoro`) times more slowly.

Two results highlight that the CPU time required by `Occurrence_Grouping` is very interesting in practice. In Table 7.2 indeed, for instances like `Brent`, `Eco9` or `Trigexp2`, `Occurrence_Grouping` is called a large number of times with roughly no effect on solving: similar number of choice points is reported with and without occurrence grouping. However, the overall CPU is also very similar. Another experiment is reported in [1].

[4] Similar results are obtained by other strategies.

Table 2 Experimental results using the monotonicity-based contractor `Mohc` inside a solving strategy. The first column indicates the name of each instance, along with its number of variables/equations (left) and solutions (right). The second column reports the CPU time (first row of a multi-row) and the number of nodes (second row) obtained by a strategy using `3BCID(Mohc)` without `OG`. The third column report the results of our strategy using `3BCID(OG+Mohc)`. The fourth column indicates the (large) number of calls to `Occurrence_Grouping` during the solving process.

System	`Mohc`		
	¬`OG`	`OG`	#`OG` calls
`ButcherA` 8 3	>1 day	**1722** 288,773	 16,772,045
`Brent` 10 1008	20 3811	20.3 3805	 30,867
`Caprasse` 4 18	2.57 1251	2.71 867	 60,073
`Eco9` 9 16	13.31 6161	13.96 6025	 70,499
`Fourbar` 4 3	4277 1,069,963	**385** 57,377	 8,265,730
`Geneig` 6 10	328 76,465	**111** 13,705	 2,982,275
`Hayes` 8 1	17.62 4599	17.45 4415	 5316
`I5` 10 30	57.25 10,399	58.12 9757	 835,130
`Katsura` 12 7	100 3711	103 3625	 39,659
`Kin1` 6 16	1.82 85	1.79 83	 316
`Pramanik` 3 2	67.98 51,877	**21.23** 12,651	 395,083
`Redeco8` 8 8	5.98 2285	6.12 2209	 56,312
`Trigexp2` 11 0	90.5 14,299	88.2 14301	 338,489
`Trigo1` 10 9	137 1513	**57** 443	 75,237
`Virasoro` 8 24	6790 619,471	**901** 38,389	 5,633,140
`Yamamura1` 8 7	11.59 2663	**2.15** 343	 43,589

8 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 of f three auxiliary, respectively increasing, decreasing and non monotonic variables. It then transforms f into a function f^{og} that replaces the occurrences of every variable by a convex linear combination of these auxiliary variables. The evaluation by monotonicity of f^{og} defines the *evaluation by occurrence grouping* of f and is better than the evaluation by monotonicity of f.

The extension by occurrence grouping computes at a low cost sharper interval images than existing interval extensions do. The main benefits of occurrence grouping lie in the improvement of an efficient contraction algorithm, called `Mohc`, that exploits monotonicity of functions. Occurrence grouping transforms, with nearly no overhead in practice, the constraints on the fly, during the constraint propagation and the solving process.

References

1. Araya, I., Neveu, B., Trombettoni, G.: An Interval Extension Based on Occurrence Grouping: Method and Properties. Tech. Rep. 7806, INRIA (2011)
2. Araya, I., Trombettoni, G., Neveu, B.: Exploiting Monotonicity in Interval Constraint Propagation. In: Proc. AAAI, pp. 9–14. AAAI Press (2010)
3. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising Hull and Box Consistency. In: Proc. ICLP, pp. 230–244 (1999)
4. Chabert, G.: Ibex – An Interval Based EXplorer. `www.ibex-lib.org` (2010)
5. Chabert, G., Jaulin, L.: Contractor Programming. Artificial Intelligence **173**, 1079–1100 (2009)
6. Chabert, G., Jaulin, L.: Hull Consistency Under Monotonicity. In: Proc. Constraint Programming CP, LNCS 5732, pp. 188–195 (2009)
7. Flannery, B., Press, W., Teukolsky, S., Vetterling, W.: Numerical Recipes in C. Press Syndicate of the University of Cambridge, New York (1992)
8. Hansen, E.: Global Optimization using Interval Analysis. Marcel Dekker inc. (1992)
9. Horner, W.G.: A new Method of Solving Numerical Equations of all Orders, by Continuous Approximation. Philosophical Transactions of the Royal Society of London **109**, 308–335 (1819)
10. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer (2001)
11. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations. Kluwer (1997)
12. Lhomme, O.: Consistency Techniques for Numeric CSPs. In: Proc. IJCAI, pp. 232–238 (1993)
13. Merlet, J.P.: The COPRIN examples page. `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html` (2010)
14. Moore, R.: Interval Analysis. Prentice Hall (1966)
15. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press (1990)
16. Trombettoni, G., Chabert, G.: Constructive Interval Disjunction. In: Proc. CP, LNCS 4741, pp. 635–650 (2007)