Une nouvelle extension de fonctions aux intervalles basée sur le regroupement d'occurrences

Ignacio Araya[1], **Bertrand Neveu**[2], **Gilles Trombettoni**[1]
[1] COPRIN, INRIA Sophia Antipolis, Université Nice Sophia
[2] Imagine, LIGM, Université Paris-Est, France
rilianx@gmail.com {neveu,trombe}@sophia.inria.fr

Résumé

Quand une fonction f est monotone par rapport à une variable sur un domaine donné, il est bien connu que l'*extension aux intervalles par monotonie* de f calcule une image plus étroite que l'*extension naturelle*.

Cet article présente une nouvelle extension aux intervalles d'une fonction f appelée *regroupement d'occurrences* et notée $[f]_{og}$. Quand f n'est pas monotone par rapport à une variable x sur un domaine donné $[B]$, nous essayons de transformer f en une nouvelle fonction f^{og} qui est monotone sur deux variables x_a et x_b, qui regroupent des occurrences de x de telle sorte que f^{og} soit croissante par rapport à x_a et décroissante par rapport à x_b. $[f]_{og}$ est l'extension aux intervalles par monotonie de f^{og} et produit une image plus étroite que l'extension naturelle.

Pour trouver un bon regroupement d'occurrences, nous proposons un programme linéaire et un algorithme qui minimisent une surestimation du diamètre de l'image de $[f]_{og}$ basée sur une forme de Taylor de f. Finalement, des expérimentations montrent les avantages de cette nouvelle extension lors de la résolution de systèmes d'équations.

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 a linear program and an algorithm that minimize 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

Le calcul de l'image optimale d'une fonction sur des intervalles est un problème difficile qui est au cœur de l'arithmétique d'intervalles. Il permet d'évaluer une formule mathématique en tenant compte de manière fiable des erreurs d'arrondis dus à l'arithmétique en virgule flottante. Des encadrements étroits permettent aussi aux méthodes par intervalles de converger rapidement vers les solutions d'un système de contraintes sur les réels. A chaque nœud de l'arbre de recherche, un *test d'existence* vérifie que, pour chaque équation $f(X) = 0$, l'extension aux intervalles de f retourne un intervalle incluant 0 (sinon la branche est coupée). Les algorithmes de propagation de contraintes peuvent aussi être améliorés s'ils utilisent de meilleures extensions aux intervalles. Par exemple, l'algorithme `Box` utilise un test d'existence à l'intérieur de son processus itératif de bissection/rognage [3].

Cet article propose une nouvelle extension aux intervalles et nous rappelons d'abord quelques notions de base de l'arithmétique d'intervalles [10, 11, 8] pour introduire les extensions aux intervalles utiles à nos travaux.

Un intervalle $[x] = [a, b]$ est l'ensemble des nombres réels compris entre a et b. $\underline{x} = a$ étant le minimum de $[x]$ et $\overline{x} = b$ le maximum. Le *diamètre* ou taille d'un intervalle est : $diam([x]) = \overline{x} - \underline{x}$, et la valeur absolue d'un intervalle est : $|[x]| = max(|\overline{x}|, |\underline{x}|)$. Un produit cartésien d'intervalles, appelé *boîte*, est noté $[B]$ ou par un vecteur d'intervalles $\{[x_1], [x_2], ..., [x_n]\}$.

Une *fonction sur intervalles* $[f]$ est une fonction de $\mathbb{IR}^n$ dans $\mathbb{IR}$, $\mathbb{IR}$ étant l'ensemble de tous les inter-

valles sur $\mathbb{R}$. $[f]$ est une *extension aux intervalles* d'une fonction $f(x_1, ..., x_n)$ de $\mathbb{R}^n$ dans $\mathbb{R}$ si l'image d'une boîte $[B] = \{[x_1], [x_2], ..., [x_n]\}$ par $[f]$ est un intervalle *conservatif*, c.-à-d. contient l'ensemble $\mathcal{I}f([B]) = \{y \in \mathbb{R}, \exists\{x_1, x_2, ..., x_n\} \in [B], y = f(x_1, x_2, ..., x_n)\}$. (Le calcul de l'image est appelé *évaluation* de f.)

L'image *optimale* $[f]_{opt}([B])$ est le plus petit intervalle contenant $\mathcal{I}f([B])$. Il existe de nombreuses extensions aux intervalles pour une fonction, la difficulté étant de définir une extension qui calcule l'image optimale, ou une approximation étroite de cette dernière.

La première idée est d'utiliser l'arithmétique d'intervalles, qui étend aux intervalles les opérateurs arithmétiques $+$, $-$, $\times$, $/$ et les fonctions élémentaires (*power*, *exp*, *log*, *sin*, *cos*, ...). Par exemple, $[a, b] + [c, d] = [a + c, b + d]$. L'*extension naturelle* $[f]_N$ d'une fonction f évalue avec l'arithmétique d'intervalles tous les opérateurs arithmétiques et toutes les fonctions élémentaires de f.

Si f est continue dans une boîte $[B]$, l'*évaluation naturelle* de f (c.-à-d. le calcul de $[f]_N([B])$) produit l'image optimale si chaque variable n'apparaît qu'une fois dans f. Quand une variable apparaît plusieurs fois, l'évaluation par l'arithmétique d'intervalles produit généralement une surestimation de $[f]_{opt}([B])$, car la corrélation entre les occurrences d'une même variable est perdue : deux occurrences d'une variable sont traitées comme des variables indépendantes. Par exemple $[x] - [x]$, avec $[x] \in [0, 1]$ donne le résultat $[-1, 1]$, au lieu de $[0, 0]$, comme le ferait $[x] - [y]$, avec $[x] \in [0, 1]$ et $[y] \in [0, 1]$.

Ce principal inconvénient de l'arithmétique d'intervalles cause une réelle difficulté pour implanter des résolveurs sur intervalles efficaces, puisque l'évaluation naturelle est un outil de base pour ces résolveurs.

Plusieurs extensions aux intervalles plus sophistiquées ont été proposées pour surmonter cette difficulté. L'*extension de Taylor* $[f]_T$ de f, définie dans [10], utilise la forme de Taylor de f :

$$[f]_T([B]) = f(B_m) + \sum_{i=1}^{n} \left(\left[\frac{\partial f}{\partial x_i} \right] ([B]) \times ([x_i] - x_i^m) \right)$$

où n est le nombre de variables de f, x_i^m est la valeur du milieu de l'intervalle $[x_i]$, B_m est le point au milieu de $[B]$ (i.e., $B_m = (x_1^m, ..., x_n^m)$) et $\left[\frac{\partial f}{\partial x_i} \right]$ est une extension aux intervalles de $\frac{\partial f}{\partial x_i}$. L'extension de Taylor calcule généralement des évaluations étroites quand les diamètres des dérivées partielles sont proches de 0. Dans d'autres cas, elle peut être pire que l'extension naturelle.

Une variante bien connue de l'extension de Taylor, appelée *extension de Hansen* $[f]_H$, calcule une image plus étroite mais à un coût plus élevé [6]. Sans entrer dans les détails, l'extension de Hansen, contrairement à celle de Taylor, calcule les dérivées partielles en utilisant la boîte $[B]$ dans laquelle certains intervalles ont été remplacés par leur point milieu. Cela implique que pour tout $[B] \in \mathbb{IR}^n$: $[f]_H([B]) \subseteq [f]_T([B])$. Cependant, quand on utilise la *différenciation automatique* pour calculer les dérivées partielles de f, l'extension de Taylor utilise la même boîte $[B]$ tandis que l'extension de Hansen doit utiliser une boîte différente par variable. Ainsi, le calcul des dérivées partielles est n fois plus cher avec l'extension de Hansen qu'avec celle de Taylor. L'extension naturelle et l'extension de Hansen (comme l'extension de Taylor) sont incomparables.

Une autre extension aux intervalles utilise la monotonie d'une fonction sur un domaine donné. En fait, quand une fonction est monotone par rapport à toutes ses variables, le problème des occurrences multiples disparaît et l'évaluation (utilisant l'extension par monotonie) devient optimale.

Définition 1 (f_{min}, f_{max}, extension par monotonie) *Soit f une fonction définie sur les variables V de domaines $[V]$. Soit $X \subseteq V$ un sous-ensemble de variables monotones.*

Considérons les valeurs x_i^+ et x_i^- telles que : si $x_i \in X$ est une variable croissante (resp. décroissante), alors $x_i^- = \underline{x_i}$ et $x_i^+ = \overline{x_i}$ (resp. $x_i^- = \overline{x_i}$ et $x_i^+ = \underline{x_i}$).

Soit $W = V \setminus X$ l'ensemble des variables non détectées monotones. Alors, f_{min} et f_{max} sont les fonctions définies par :

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

Finalement, l'extension par monotonie $[f]_M$ de f dans la boîte $[V]$ produit l'intervalle suivant :

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

Les bornes de l'évaluation par monotonie peuvent être calculées en utilisant n'importe quelle extension aux intervalles (et non nécessairement l'extension naturelle).

Quand l'évaluation par monotonie utilise, récursivement, la même évaluation par monotonie pour calculer les bornes de l'image, nous désignons cette évaluation par *extension récursive par monotonie.*

Considérons par exemple la fonction $f(x_1, x_2) = -6x_1 + x_1x_2^2 + 3x_2$. Les dérivées partielles par rapport à x_1 et x_2 sont $\frac{\partial f}{\partial x_1}(x_2) = -6 + x_2^2$ et $\frac{\partial f}{\partial x_2}(x_1, x_2) = 2x_1x_2 + 3$. Si les intervalles des variables sont $[x_1] = [-2, -1]$ et $[x_2] = [0, 1]$, alors f est décroissante en x_1 et non monotone en x_2. Ainsi, l'évaluation récursive par monotonie devra calculer :

$$[f]_{MR}([B]) = [\underline{[f]_{MR}(-1, [0, 1])}, \overline{[f]_{MR}(-2, [0, 1])}]$$

Dans la boîte $\{-1\} \times [0,1]$ impliquée dans l'évaluation de la borne gauche, il se trouve que f est maintenant croissante en x_2 ($[\frac{\partial f}{\partial x_2}]_N(-1,[0,1]) = [1,3]$). Nous pouvons donc remplacer l'intervalle $[x_2]$ par sa borne gauche. En revanche, dans la boîte $\{-2\} \times [0,1]$, f n'est toujours pas monotone en x_2. Ainsi, l'évaluation récursive par monotonie calcule finalement :

$$[f]_{MR}([B]) = [\underline{[f](-1,0)}, \overline{[f](-2,[0,1])}] = [6,15]$$

où $[f]$ peut être n'importe quelle extension aux intervalles, par exemple l'extension naturelle.

De manière générale, l'évaluation récursive par monotonie calcule un intervalle image plus étroit (ou égal) que ne le fait l'évaluation par monotonie (c.-à-d., $[f]_{MR}([B]) \subseteq [f]_M([B])$) à un coût entre 2 et $2n$ fois plus grand. Dans l'exemple, on vérifie bien que l'évaluation par monotonie produit une évaluation ($[f]_M([B]) = [5,15]$) légèrement plus mauvaise que la variante récursive.

Cet article explique comment utiliser la monotonie quand une fonction n'est pas monotone par rapport à une variable x, mais est monotone par rapport à un sous-ensemble des occurrences de x.

Nous présentons dans la partie suivante l'idée de regrouper les occurrences en trois groupes, croissantes, décroissantes et non monotones. Des programmes linéaires permettant d'obtenir des regroupements d'occurrences *intéressants* sont décrits dans les parties 3 et 4. Dans la partie 5, nous proposons un algorithme qui résout le programme linéaire présenté à la partie 4. Finalement, dans la partie 6, quelques expérimentations montrent les avantages de ce regroupement d'occurrences pour résoudre des systèmes d'équations, en particulier quand on utilise un algorithme de filtrage comme `Mohc` [2, 1] qui exploite la monotonie.

2 Évaluation par monotonie avec regroupement d'occurrences

Dans cette partie, nous étudions le cas d'une fonction qui n'est pas monotone par rapport à une variable à occurrences multiples. Nous pouvons, sans perte de généralité, limiter l'étude à une fonction d'une variable : la généralisation à une fonction de plusieurs variables est immédiate, les évaluations par monotonie étant indépendantes.

Exemple 1 *Considérons* $f_1(x) = -x^3 + 2x^2 + 6x$. *Nous voulons calculer l'image de l'intervalle* $[-1.2, 1]$ *par cette fonction. La dérivée est* $f_1'(x) = -3x^2 + 4x + 6$ *: elle contient un terme positif* (6)*, un terme négatif* $(-3x^2)$ *et un terme contenant zéro* $(4x)$.

$[f_1]_{opt}([B])$ *vaut* $[-3.05786, 7]$*, mais on ne peut pas l'obtenir directement par une simple évaluation de fonction sur intervalles. (On doit résoudre* $f_1'(x) = 0$*, ce qui est dans le cas général un problème en soi.)*

Dans l'intervalle $[-1.2, 1]$*, la fonction* f_1 *n'est pas monotone. L'évaluation naturelle donne* $[-8.2, 10.608]$*, celle de Horner (utilisant une forme factorisée ; voir [7]) produit* $[-11.04, 9.2]$.

Quand une fonction n'est pas monotone par rapport à une variable x, elle peut parfois être monotone par rapport à certaines occurrences. Une première idée naïve pour utiliser la monotonie en ces occurrences est la suivante. On remplace la fonction f par une fonction f^{nog}, regroupant toutes les occurrences croissantes dans une variable x_a, toutes les occurrences décroissantes dans une variable x_b, et les non monotones dans x_c. Le domaine des nouvelles variables auxiliaires est le même : $[x_a] = [x_b] = [x_c] = [x]$.

Pour f_1, ce regroupement aboutit à $f_1^{nog}(x_a, x_b, x_c) = -x_b^3 + 2x_c^2 + 6x_a$. En suivant la définition 1 :

- $\underline{[f_1^{nog}]_M([-1.2,1])} = \underline{[f_1^{nog}]_N(-1.2, 1, [-1.2,1])} = \underline{-1^3 + 2[-1.2,1]^2 - 7.2} = -8.2$
- $\overline{[f_1^{nog}]_M([-1.2,1])} = 10.608$

Finalement, l'évaluation par monotonie produit : $[f_1^{nog}]_M([-1.2,1]) = [-8.2, 10.608]$.

Il apparaît que l'évaluation par monotonie de la nouvelle fonction f^{nog} produit toujours le même résultat que l'évaluation naturelle. En effet, quand un nœud dans l'arbre d'évaluation correspond à une fonction croissante par rapport à un nœud fils, l'évaluation naturelle choisit automatiquement la borne droite du domaine du nœud fils pour calculer la borne droite du domaine du nœud.

L'idée principale de l'article est donc de changer ce regroupement pour réduire le problème de dépendance et obtenir des évaluations plus étroites. Nous pouvons en effet regrouper des occurrences (croissantes, décroissantes ou non monotones) dans une variable croissante x_a tant que la fonction reste croissante par rapport à cette variable x_a. Par exemple, si on place une occurrence non monotone dans un groupe monotone, l'évaluation peut être meilleure (ou rester la même). De même, si on peut transférer toutes les occurrences décroissantes dans la partie croissante, le problème de dépendance n'apparaît plus qu'entre les occurrences des groupes croissant et non monotone.

Pour f_1, si on regroupe le terme à dérivée positive avec le terme à dérivée contenant 0, on obtient la nouvelle fonction : $f_1^{og}(x_a, x_b) = -x_b^3 + 2x_a^2 + 6x_a$. Comme la dérivée par rapport au regroupement des deux occurrences (la variable x_a) est positive : $4[x_a] + 6 = [1.2, 10]$, f_1^{og} est croissante par rapport à x_a. On peut alors utiliser l'évaluation par monotonie et obtenir l'intervalle $[-5.32, 9.728]$. On peut de la même manière

obtenir $f_1^{og}(x_a, x_c) = -x_a^3 + 2x_c^2 + 6x_a$, l'évaluation par monotonie donnant alors $[-5.472, 7.88]$. On remarque alors qu'on trouve des images plus étroites que l'évaluation naturelle de f_1.

Dans la partie 3, nous présentons un programme linéaire qui réalise automatiquement un *regroupement d'occurrences.*

Extension aux intervalles par regroupement d'occurrences

Considérons la fonction $f(x)$ avec des occurrences multiples de x. On obtient une nouvelle fonction $f^{og}(x_a, x_b, x_c)$ en remplaçant dans f chaque occurrence de x par l'une des trois variables x_a, x_b, x_c, de telle sorte que f^{og} soit croissante par rapport à x_a dans $[x]$ et décroissante par rapport à x_b. On définit alors l'*extension aux intervalles par regroupement d'occurrences* de f par :

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

Contrairement aux extensions naturelle et par monotonie, l'extension aux intervalles par regroupement d'occurrences n'est pas unique pour une fonction f puisqu'elle dépend du regroupement d'occurrences (og) qui transforme f en f^{og}.

3 Un programme linéaire en 0,1 pour réaliser le regroupement d'occurrences

Dans cette partie, nous proposons une méthode pour automatiser le regroupement d'occurrences. En nous basant sur l'extension de Taylor, nous calculons d'abord une surestimation du *diamètre* de l'image calculée par $[f]_{og}$. Ensuite, nous proposons un programme linéaire réalisant un regroupement qui minimise cette surestimation.

3.1 Surestimation basée sur Taylor

D'une part, comme f^{og} peut ne pas être monotone par rapport à x_c, l'évaluation par monotonie considère les occurrences de x_c comme des variables différentes, comme le ferait l'évaluation naturelle. D'autre part, comme f^{og} est monotone par rapport à x_a et x_b, l'évaluation par monotonie de ces variables est optimale. Les deux propositions suivantes sont bien connues.

Proposition 1 *Soit $f(x)$ une fonction continue dans une boîte $[B]$ avec l'ensemble des occurrences de x : $\{x_1, x_2, ..., x_k\}$. $f^\circ(x_1, .., x_k)$ est la fonction obtenue en considérant toutes les occurrences de x comme des variables différentes. Alors, $[f]_N([B])$ calcule $[f^\circ]_{opt}([B])$.*

Proposition 2 *Soit $f(x_1, x_2, ..., x_n)$ une fonction monotone en toutes ses variables sur la boîte $[B] = \{[x_1], [x_2], ..., [x_n]\}$. Alors, l'évaluation par monotonie est optimale sur $[B]$, c.-à-d. calcule $[f]_{opt}([B])$.*

En utilisant ces deux propositions, on observe que $[f^{og}]_M([x_a], [x_b], [x_c])$ est équivalent à $[f^\circ]_{opt}([x_a], [x_b], [x_{c_1}], ..., [x_{c_{ck}}])$, en considérant chaque occurrence de x_c dans f^{og} comme une variable indépendante x_{c_j} dans f°, ck étant le nombre d'occurrences de x_c dans f^{og}. En utilisant l'évaluation de Taylor, une surestimation du diamètre $diam([f]_{opt}([B]))$ est donnée par le côté droit de (1) dans la proposition 3.

Proposition 3 *Soit $f(x_1, ..., x_n)$ une fonction sur les domaines $[B] = \{[x_1], ..., [x_n]\}$. Alors,*

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

où $[g_i]$ est une extension aux intervalles de $g_i = \frac{\partial f}{\partial x_i}$.

En utilisant la proposition 3, on peut calculer un majorant du **diamètre** de $[f]_{og}([B]) = [f^{og}]_M([B]) = [f^\circ]_{opt}([B])$:

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

$[g_a]$, $[g_b]$ et $[g_{c_i}]$ sont les extensions aux intervalles de $g_a = \frac{\partial f^{og}}{\partial x_a}$, $g_b = \frac{\partial f^{og}}{\partial x_b}$ et $g_{c_i} = \frac{\partial f^{og}}{\partial x_{c_i}}$. $diam([x])$ est factorisé car $[x] = [x_a] = [x_b] = [x_{c_1}] = ... = [x_{c_{ck}}]$.

Pour garantir les conditions de monotonie requises par f^{og} : $\frac{\partial f^{og}}{\partial x_a} \geq 0$, $\frac{\partial f^{og}}{\partial x_b} \leq 0$, nous avons les conditions suffisantes $\underline{[g_a]([B])} \geq 0$ et $\overline{[g_b]([B])} \leq 0$, qui impliquent $|[g_a]([B])| = \overline{[g_a]([B])}$ et $|[g_b]([B])| = -\underline{[g_b]([B])}$. Finalement :

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

3.2 Un programme linéaire

Nous voulons transformer f en une nouvelle fonction f^{og} qui minimise le côté droit de la relation (2). Le problème peut être facilement transformé en le programme linéaire en nombres entiers suivant :

Trouver les valeurs r_{a_i}, r_{b_i} et r_{c_i} pour chaque occurrence x_i qui minimisent

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

sous les contraintes :

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

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

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

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

où une valeur r_{a_i}, r_{b_i} ou r_{c_i} égale à 1 indique que l'occurrence x_i de f sera remplacée, respectivement, par x_a, x_b ou x_c dans f^{og}. k est le nombre d'occurrences de x, $[g_a]([B]) = \sum_{i=1}^{k} [g_i]([B])r_{a_i}$, $[g_b]([B]) = \sum_{i=1}^{k} [g_i]([B])r_{b_i}$, et $[g_i]([B]), ..., [g_k]([B])$ sont les dérivées par rapport à chaque occurrence.

On peut remarquer que $[g_a]([B])$ et $[g_b]([B])$) sont calculés en utilisant uniquement les dérivées de f par rapport à chaque occurrence de x ($[g_i]([B])$).

Programme linéaire correspondant à l'exemple 1

Nous avons $f_1(x) = -x^3 + 2x^2 + 6x$ et $f_1'(x) = -3x^2 + 4x + 6$ avec $x \in [-1.2, 1]$. Le gradient vaut : $[g_1]([-1.2, 1]) = [-4.32, 0]$, $[g_2]([-1.2, 1]) = [-4.8, 4]$ et $[g_3]([-1.2, 1]) = [6, 6]$. Alors, le programme linéaire est :

Trouver les valeurs r_{a_i}, r_{b_i} et r_{c_i} qui minimisent

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

sous les contraintes :

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

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

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

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

On obtient le minimum 10.8, et la solution $r_{a_1} = 1, r_{b_1} = 0, r_{c_1} = 0, r_{a_2} = 0, r_{b_2} = 0, r_{c_2} = 1, r_{a_3} = 1, r_{b_3} = 0, r_{c_3} = 0$, qui est la dernière solution présentée à la section 2. On peut remarquer que la valeur de la surestimation de $diam([f]_{og}([B]))$ est égale à 23.76 ($10.8 \times diam([-1.2, 1])$) alors que $diam([f]_{og}([B])) = 13.352$. Bien que la surestimation soit assez grossière, l'heuristique fonctionne bien sur cet exemple. En effet, $diam([f]_N([B])) = 18.808$ et $diam([f]_{opt}([B])) = 10.06$.

4 Un programme linéaire continu

Le programme linéaire précédent est un programme linéaire en 0,1 qui est connu comme étant NP-difficile en général. On peut le rendre continu et polynomial en permettant à r_{a_i}, r_{b_i} et r_{c_i} de prendre des valeurs réelles. En d'autres termes, on permet à chaque occurrence de x dans f d'être remplacée par une combinaison linéaire convexe des variables auxiliaires x_a, x_b et x_c, f^{og} étant croissante par rapport à x_a et décroissante par rapport à x_b. Chaque occurrence x_i est remplacée dans f^{og} par $r_{a_i}x_a + r_{b_i}x_b + r_{c_i}x_c$, avec $r_{a_i} + r_{b_i} + r_{c_i} = 1$, $\frac{\partial f^{og}}{\partial x_a} \geq 0$ et $\frac{\partial f^{og}}{\partial x_b} \leq 0$. On peut remarquer alors que f et f^{og} ont la même évaluation naturelle.

Dans l'exemple 1, on peut remplacer f_1 par f^{og_1} ou f^{og_2} en respectant les contraintes de monotonie sur x_a et x_b. Considérons l'intervalle $[x] = [-1.2, 1]$:

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. $f_1^{og_2}(x_a, x_b, x_c) = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$
   $[f_1^{og_2}]_M([x]) = [-5.472, 7]$

Exemple 2 *Considérons la fonction $f_2(x) = x^3 - x$ et l'intervalle $[x] = [0.5, 2]$. f_2 n'est pas monotone et l'image optimale $[f_2]_{opt}([x])$ vaut $[-0.385, 6]$. L'évaluation naturelle donne $[-1.975, 7.5]$, l'évaluation de Horner $[-1.5, 6]$. On peut remplacer f_2 par l'une des fonctions suivantes :*

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. $f_2^{og_2}(x_a, x_b) = (\frac{11}{12}x_a + \frac{1}{12}x_b)^3 - x_b$
   $[f_2^{og_2}]_M([x]) = [-1.756, 6.09]$

Le nouveau programme linéaire qui prend en compte la combinaison linéaire convexe pour réaliser le regroupement d'occurrences devient :
Trouver les valeurs r_{a_i}, r_{b_i} et r_{c_i} pour chaque occurrence x_i qui minimisent (3) sous les contraintes (4), (5), (6) et

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

Programme linéaire correspondant à l'exemple 1

Dans cet exemple, nous obtenons comme minimum 10.58 et la nouvelle fonction $f_1^{og}(x_a, x_b, x_c) = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$: $[f_1^{og}]_M([x]) = [-5.472, 7]$. Le minimum 10.58 est inférieur à 10.8 (obtenu par le programme linéaire en 0,1). L'évaluation par regroupement d'occurrences de f_1 donne $[-5.472, 7]$, ce qui est plus étroit que l'image $[-5.472, 7.88]$ obtenue par le programme linéaire en 0,1 présenté dans la partie 3.

Programme linéaire correspondant à l'exemple 2

Dans cet exemple, nous obtenons comme minimum 11.25 et la nouvelle fonction $f_2^{og}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$. L'image $[-0.75, 6.01]$ obtenue par regroupement d'occurrences est plus étroite que celles obtenues par les évaluations naturelle et de Horner. Dans ce cas, le programme linéaire en 0,1 de la partie 3 donne le regroupement naïf.

On notera que le programme linéaire continu non seulement rend le problème d'optimisation polynomial, mais peut aussi améliorer le minimum (les conditions d'intégrité étant relâchées).

5 Un algorithme efficace de regroupement d'occurrences

L'algorithme 1 trouve les valeurs r_{a_i}, r_{b_i}, r_{c_i} qui minimisent G sous les contraintes de monotonie. Il génère aussi la nouvelle fonction f^{og} qui remplace chaque occurrence x_i de f par $[r_{a_i}]x_a + [r_{b_i}]x_b + [r_{c_i}]x_c$. On notera que les valeurs sont représentées par de petits intervalles (quelques u.l.p.), pour prendre en compte les erreurs d'arrondis des calculs en virgule flottante.

L'algorithme 1 utilise un vecteur $[g_*]$ de taille k contenant les intervalles des dérivées partielles de f par rapport à chaque *occurrence* x_i de x. Chaque composante de $[g_*]$ est notée $[g_i]$ et correspond à l'intervalle $\left[\frac{\partial f}{\partial x_i}\right]_N([B])$.

Un symbole indicé par une astérisque (*) représente un vecteur ($[g_*]$, $[r_{a_*}]$).

Nous illustrons l'algorithme avec les deux fonctions de nos exemples : $f_1(x) = -x^3 + 2x^2 + 6x$ et $f_2(x) = x^3 - x$ pour les domaines de x : $[-1.2, 1]$ et $[0.5, 2]$ respectivement. Pour ces exemples, les intervalles des dérivées de f_2 par rapport aux occurrences de x sont $[g_1] = [0.75, 12]$ et $[g_2] = [-1, -1]$; ceux des dérivées de f_1 sont $[g_1] = [-4.32, 0]$, $[g_2] = [-4.8, 4]$ et $[g_3] = [6, 6]$.

A la ligne 1, nous calculons la dérivée partielle $[G_0]$ de f par rapport à x en sommant les dérivées partielles de f par rapport à chaque occurrence de x. A la ligne 2, $[G_m]$ donne la valeur de la dérivée partielle de f par rapport aux occurrences monotones de x. Sur les exemples, pour f_1 : $[G_0] = [g_1]+[g_2]+[g_3] = [-3.12, 10]$ et $[G_m] = [g_1] + [g_3] = [1.68, 6]$, et pour f_2 : $[G_0] = [G_m] = [g_1] + [g_2] = [-0.25, 11]$.

Algorithm 1 `OccurrenceGrouping`(**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`$(f, [r_{a_*}], [r_{b_*}], [r_{c_*}])$

Selon les valeurs de $[G_0]$ et $[G_m]$, on peut distinguer 3 cas. Le premier cas est bien connu ($0 \notin [G_0]$ à la ligne 3) et apparaît quand x est une variable monotone. La procédure `OG_case1` ne réalise aucun regroupement d'occurrences : *toutes les occurrences* de x sont remplacées par x_a (si $[G_0] \geq 0$) ou par x_b (si $[G_0] \leq 0$). L'évaluation par monotonie de f^{og} est équivalente à l'évaluation par monotonie de f.

Dans le second cas, quand $0 \in [G_m]$ (ligne 5), la procédure `OG_case2` (Algorithme 2) réalise un regroupement des occurrences de x. Les occurrences croissantes sont remplacées par $(1 - \alpha_1)x_a + \alpha_1 x_b$, les occurrences décroissantes par $\alpha_2 x_a + (1 - \alpha_2)x_b$ et les non monotones par x_c (lignes 7 à 13 de l'algorithme 2). f_2 tombe dans ce cas de figure : $\alpha_1 = \frac{1}{45}$ et $\alpha_2 = \frac{11}{15}$ sont calculées aux lignes 3 et 4 de l'algorithme 2 en utilisant $[G^+] = [g_1] = [0.75, 12]$ et $[G^-] = [g_2] = [-1, -1]$. La nouvelle fonction devient : $f_2^{og}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$.

Le troisième cas apparaît quand $0 \notin [G_m]$ et $0 \in [G_0]$. Sans perte de généralité, supposons $\underline{G_m} \geq 0$. La procédure `OG_case3`$^+$ (Algorithme 3) regroupe d'abord toutes les occurrences monotones dans le groupe croissant (lignes 2–5). Les occurrences non monotones sont alors remplacées par x_a dans un ordre déterminé par le tableau $index$[1] (ligne 7) tant que la contrainte $\sum_{i=1}^{k} r_{a_i}\underline{g_i} \geq 0$ est satisfaite (lignes 9–13). La première occurrence non monotone $x_{i'}$ qui rendrait la contrainte non satisfaite est remplacée par

[1] x_{i_1} est traitée avant x_{i_2} si $|\overline{g_{i_1}}/\underline{g_{i_1}}| \leq |\overline{g_{i_2}}/\underline{g_{i_2}}|$. $index[j]$ rend l'indice de la $j^{ème}$ occurrence dans cet ordre.

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**

$\alpha x_a + (1 - \alpha)x_c$, avec α tel que la contrainte est satisfaite et égale à 0, $(\sum_{i=1, i \neq i'}^{k} r_{a_i} \underline{g_i}) + \alpha \underline{g_{i'}} = 0$ (lignes 15–17). Le reste des occurrences non monotones sont remplacées par x_c (lignes 20–22). f_1 tombe dans ce cas de figure. Les première et troisième occurrences de x sont monotones et sont remplacées par x_a. La deuxième occurrence de x, qui est non monotone, est remplacée par $\alpha x_a + (1 - \alpha)x_c$, où $\alpha = 0.35$ est obtenu en forçant la contrainte (4) à valoir 0 : $\underline{g_1} + \underline{g_3} + \alpha \underline{g_2} = 0$. On obtient ainsi la nouvelle fonction : $f_1^{og} = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$.

Finalement, la procédure `Generate_New_Function` (ligne 15 de l'algorithme 1) crée de manière symbolique la nouvelle fonction f^{og}.

Observations

L'algorithme 1 respecte les quatre contraintes (4)–(7). Nous avons démontré que le minimum de la fonction objectif (3) est atteint pour `OG_case2` et sommes en train de finaliser la preuve pour `OG_case3`.

On peut utiliser un algorithme du simplexe standard à la place de l'algorithme 1, à condition que l'implantation prenne en compte les erreurs d'arrondis dues aux calculs en virgule flottante. Nous présentons à la partie 6.3 une comparaison des performances respectives de l'algorithme 1 et du simplexe.

Complexité en temps

La complexité en temps de `Occurrence_Grouping` pour une variable avec k occurrences est en $O(k\, log_2(k))$. Elle est dominée par la complexité de

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: $[g_a] \leftarrow [g_a] + [g_i]$ /* Toutes les dérivées positives et négatives sont absorbées dans $[g_a]$ */
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: $[\alpha] \leftarrow -\frac{g_a}{\underline{g_i}}$
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**

`ascending_sort` dans la procédure `OG_case3`. Comme le montrent les expérimentations de la partie suivante, le temps utilisé en pratique par `Occurrence_Grouping` est négligeable quand la procédure est utilisée dans la résolution de systèmes d'équations.

6 Expérimentations

Le regroupement d'occurrences a été implanté dans le résolveur par intervalles `Ibex` [5, 4] en `C++`. Le but principal de ces expérimentations est de montrer les améliorations en temps de calcul apportées par le regroupement d'occurrences dans la résolution de systèmes d'équations. Seize problèmes tests proviennent du site [9]. Ils correspondent à des systèmes carrés avec un nombre fini de solutions ayant au moins deux contraintes avec des occurrences multiples et demandant plus d'une seconde pour être résolus. Deux instances (`<name>-bis`) ont été simplifiées pour réduire leur temps de résolution : les domaines initiaux des variables ont été arbitrairement réduits.

6.1 Regroupement d'occurrences pour améliorer le test d'existence par monotonie

Tout d'abord, le regroupement d'occurrences a été implanté pour être utilisé dans un test d'existence par

Problème	3BCID	¬OG	OG	Problème	3BCID	¬OG	OG
Brent	18.9	19.5	19.1	Butcher-bis	351	360	340
10 1008	3941	3941	3941	8 3	228305	228303	228245
Caprasse	2.51	2.56	2.56	Fourbar	13576	6742	**1091**
4 18	1305	1301	1301	4 3	8685907	4278767	963113
Hayes	39.5	41.1	40.7	Geneig	593	511	**374**
8 1	17701	17701	17701	6 10	205087	191715	158927
I5	55.0	56.3	56.7	Pramanik	100	66.6	**37.2**
10 30	10645	10645	10645	3 2	124661	98971	69271
Katsura	74.1	74.5	75.0	Trigexp2	82.5	87.0	86.7
12 7	4317	4317	4317	11 0	14287	14287	14287
Kin1	1.72	1.77	1.77	Trigo1	152	155	156
6 16	85	85	85	10 9	2691	2691	2691
Eco9	12.7	13.5	13.2	Virasoro-bis	21.1	21.5	19.8
9 16	6203	6203	6203	8 224	2781	2781	2623
Redeco8	5.61	5.71	5.66	Yamamura1	9.67	10.04	9.86
8 8	2295	2295	2295	8 7	2883	2883	2883

TAB. 1 – Résultats expérimentaux obtenus en utilisant le test d'existence par monotonie. Les colonnes 1 et 5 indiquent le nom du problème, avec son nombre de variables (à gauche) et de solutions (à droite). Les colonnes 2 et 6 donnent les temps de calcul en secondes (en haut) et le nombre de nœuds (en bas) obtenus sur un `Intel 6600 2.4 GHz` avec une stratégie basée sur `3BCID`. Les colonnes 3 et 7 donnent les résultats obtenus par la stratégie utilisant un test d'existence par monotonie standard et `3BCID`. Les colonnes 4 et 8 montrent les résultats de notre stratégie utilisant un test d'existence basé sur le regroupement d'occurrences et `3BCID`.

monotonie (`OG` dans le tableau 1). Un regroupement d'occurrences transformant f en f^{og} est appliqué après une bissection et avant une contraction. Ensuite, le test d'existence par monotonie est appliqué à f^{og} : si l'évaluation par monotonie de f^{og} ne contient pas 0, alors la branche est coupée dans l'arbre de recherche.

La stratégie concurrente (¬`OG`) applique directement le test d'existence par monotonie de f sans regroupement d'occurrences.

Les contracteurs utilisés dans les deux cas sont les mêmes : `3BCID` [12] et Newton sur intervalles.

On peut observer à partir de ces premiers résultats que `OG` est nettement meilleur que ¬`OG` sur trois problèmes seulement (`Fourbar`, `Geneig` et `Pramanik`). Sur les autres, l'évaluation par regroupement d'occurrences apparaît inutile. En effet, dans la plupart des problèmes, ce test d'existence ne coupe pas de branche dans l'arbre de recherche. On peut remarquer cependant que `OG` ne demande pas de temps supplémentaire par rapport à ¬`OG`. Cela souligne que le temps de calcul du regroupement d'occurrences est négligeable.

6.2 Regroupement d'occurrences dans un contracteur par monotonie

`Mohc` [2, 1] est un nouveau contracteur basé sur de la propagation de contraintes (comme `HC4` ou `Box`) qui utilise la monotonie d'une fonction pour améliorer la contraction des variables de la contrainte. Appelée à l'intérieur d'un algorithme de propagation, la procédure `Mohc-revise(`f`)` améliore le filtrage obtenu par `HC4-revise(`f`)` en effectuant principalement deux appels additionnels `HC4-revise(`$f_{min} \leq 0$`)` et `HC4-revise(`$f_{max} \geq 0$`)` (f_{min} et f_{max} sont introduits à la définition 1). Il appelle aussi une version monotone de la procédure `BoxNarrow` utilisée par `Box` [3].

Le tableau 2 montre les résultats de `Mohc` sans l'algorithme de regroupement d'occurrences `OG` (¬`OG`), et avec (`OG`), la fonction f étant transformée en f^{og} avant d'appeler `Mohc-revise(`f^{og}`)`.

On observe que pour 7 des 16 benchmarks, le regroupement d'occurrences est capable d'améliorer les résultats de `Mohc` sur `Butcher-bis`, `Fourbar`, `Virasoro-bis` et `Yamamura1` pour lesquels les gains en temps de calcul ($\frac{\neg OG}{OG}$) sont respectivement 30, 11, 5.6 et 5.4.

6.3 Comparaison avec le simplexe

Nous avons comparé les performances de deux implantations du regroupement d'occurrences, l'une utilisant notre algorithme ad hoc (`Occurrence_Grouping`) et l'autre la méthode du simplexe. L'algorithme du simplexe utilisé a été pris sur la page `pagesperso-orange.fr/jean-pierre.moreau/Cplus/tsimplex_cpp.txt`. Il n'est pas fiable, c.-à-d. ne prend pas en compte les erreurs d'arrondis de l'arithmétique en virgule flottante.

Deux résultats importants ont été obtenus. Tout d'abord, nous avons vérifié expérimentalement que notre algorithme est correct, c.-à-d. obtient le mini-

Problème	Mohc			Problème	Mohc		
	¬OG	OG	#OG calls		¬OG	OG	#OG calls
Brent	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	100	103		Trigexp2	90.57	88.24	
	3711	3625	39659		14299	14301	338489
Kin1	1.82	1.79		Trigo1	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	11.59	**2.15**	
	2285	2209	56312		2663	343	43589

TAB. 2 – Résultats expérimentaux avec Mohc. Les colonnes 1 et 3 indiquent le nom de chaque problème, les colonnes 2 et 6 montrent les résultats obtenus par la stratégie 3BCID(Mohc) sans OG. Les colonnes 3 et 7 présentent les résultats de notre stratégie utilisant 3BCID(OG+Mohc). Les colonnes 4 et 8 indiquent le nombre d'appels au regroupement d'occurrences.

mum de la fonction objectif G. Ensuite, comme nous l'espérions, la performance de l'algorithme du simplexe général est moins bonne que celle de notre algorithme. Il prend entre 2.32 (Brent) et 10 (Virasoro) fois plus de temps.

6.4 Comparaison des diamètres lors de l'évaluation

Le tableau 3 présente une comparaison entre l'évaluation par regroupement d'occurrences ($[f]_{og}$) et un ensemble d'évaluations sur intervalles incluant Taylor, Hansen [6] et des extensions par monotonie. Les différentes colonnes sont liées aux différentes extensions $[f]_{ext}$ et donnent la moyenne des rapports $\rho_{ext} = \frac{Diam([f]_{og}([B]))}{Diam([f]_{ext}([B]))}$ calculés dans chaque procédure (Mohc-)Revise d'une fonction f à chaque nœud de l'arbre de recherche d'une stratégie de résolution utilisant l'algorithme Mohc.

La liste des extensions aux intervalles correspondant aux colonnes du tableau sont : l'extension naturelle $[f]_N$, l'extension de Taylor $[f]_T$, l'extension de Hansen $[f]_H$, l'évaluation par monotonie $[f]_M$, l'évaluation récursive par monotonie $[f]_{MR}$, l'évaluation récursive par monotonie qui calcule les bornes en utilisant l'extension de Hansen $[f]_{MR+H}$ et une évaluation récursive par monotonie qui calcule les bornes en utilisant l'extension par regroupement d'occurrences $[f]_{MR+og}$.

Il est bien connu que les extensions de Taylor et de Hansen ne sont pas comparables avec l'extension naturelle. C'est pourquoi, pour obtenir des comparaisons plus raisonnables, nous avons redéfini $[f]_T([B]) = [f]'_T([B]) \cap [f]_N([B])$ et $[f]_H([B]) = [f]'_H([B]) \cap [f]_N([B])$, où $[f]'_T$ et $[f]'_H$ sont les véritables extensions de Taylor et Hansen.

Le tableau montre que $[f]_{og}$ calcule, en général, des évaluations plus étroites que tous ses concurrents. (Seul $[f]_{MR+og}$ obtient des évaluations plus étroites, mais il utilise aussi le regroupement d'occurrences.) Les améliorations par rapport aux deux évaluations par monotonie $[f]_M$ et $[f]_{MR}$ confirment les avantages de notre approche. Par exemple, sur Fourbar, $[f]_{og}$ produit un diamètre d'intervalle qui vaut 42.7% de celui produit par $[f]_{MR}$.

$[f]_{MR+H}$ obtient les meilleures évaluations sur trois problèmes (Caprasse, Fourbar et Virasoro). Cependant, $[f]_{MR+H}$ est plus coûteux que $[f]_{og}$. $[f]_{MR+H}$ requiert le calcul de $2n$ dérivées partielles, traversant ainsi $4n$ fois l'arbre de l'expression si on utilise une méthode de dérivation automatique [6].

$[f]_{MR+og}$ produit une évaluation nécessairement meilleure que (ou égale à) $[f]_{og}$. Cependant, les expérimentations sur nos problèmes tests soulignent que le gain en diamètre n'est que de 1.6% en moyenne (entre 0% et 6.2%), si bien que nous pensons que cela ne compense pas son coût supplémentaire.

7 Conclusion

Nous avons proposé une nouvelle méthode pour améliorer l'évaluation par monotonie d'une fonction f. Cette méthode, basée sur un regroupement d'occurrences, crée pour chaque variable trois variables auxiliaires sur lesquelles f est respectivement croissante,

NCSP	$[f]_N$	$[f]_T$	$[f]_H$	$[f]_M$	$[f]_{MR}$	$[f]_{MR+H}$	$[f]_{MR+og}$
Brent	0.857	0.985	0.987	0.997	0.998	0.999	1.000
Butcher-bis	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
MOYENNE	0.564	0.822	0.888	0.845	0.874	0.973	1.016

TAB. 3 – Différentes évaluations comparées à $[f]_{og}$

décroissante et non monotone. Elle transforme alors f en une fonction f^{og} qui remplace les occurrences d'une variable par une combinaison linéaire convexe de ces variables auxiliaires. Il en résulte l'*évaluation par regroupement d'occurrences* de f, c.-à-d. l'évaluation par monotonie de f^{og}, qui est meilleure que l'évaluation par monotonie de f.

L'extension de monotonie par regroupement d'occurrences a montré de bonnes performances quand elle est utilisée comme test d'existence et quand elle est incluse dans le contracteur `Mohc` qui exploite la monotonie des fonctions.

Références

[1] Ignacio Araya, Gilles Trombettoni, and Bertrand Neveu. Exploitation de la monotonie des fonctions dans la propagation de contraintes sur intervalles. In *Actes JFPC*, 2010.

[2] Ignacio Araya, Gilles Trombettoni, and Bertrand Neveu. Exploiting Monotonicity in Interval Constraint Propagation. In *Proc. AAAI (to appear)*, 2010.

[3] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.

[4] Gilles Chabert. `www.ibex-lib.org`, 2009.

[5] Gilles Chabert and Luc Jaulin. Contractor Programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[6] Eldon Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker inc., 1992.

[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`.

[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.