HAL open science

Exploitation de la monotonie des fonctions dans la propagation de contraintes sur intervalles

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni

To cite this version:

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni. Exploitation de la monotonie des fonctions dans la propagation de contraintes sur intervalles. JFPC 2010 - Sixièmes Journées Francophones de Programmation par Contraintes, Jun 2010, Caen, France. pp.23-31. inria-00520371

HAL Id: inria-00520371

https://inria.hal.science/inria-00520371v1

Submitted on 23 Sep 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Exploitation de la monotonie des fonctions dans la propagation de contraintes sur intervalles

Ignacio Araya[1] **Gilles Trombettoni**[1] **Bertrand Neveu**[2]

[1] COPRIN, INRIA Sophia Antipolis, Université Nice Sophia

[2] Imagine, LIGM, Université Paris-Est, France

rilianx@gmail.com {neveu,trombe}@sophia.inria.fr

Résumé

Nous proposons un nouvel algorithme de propagation de contraintes sur intervalles, appelé *consistance d'enveloppe monotone* (Mohc), qui exploite la monotonie des fonctions. La propagation est standard, mais la procédure de *révision* Mohc-Revise, utilisée pour contracter le domaine d'une variable par rapport à une contrainte individuelle, utilise des versions monotones des procédures classiques HC4-Revise et BoxNarrow. Mohc-Revise semble être la première procédure de révision adaptative en programmation par contraintes (sur intervalles). Quand une fonction est monotone en toutes ses variables, nous montrons que Mohc-Revise calcule la boîte englobante optimale de la contrainte correspondante (*Hull-consistance*). Des résultats expérimentaux très prometteurs suggèrent que Mohc a le potentiel de devenir une alternative aux algorithmes classiques HC4 et Box.

Abstract

We propose a new *interval* constraint propagation algorithm, called *MOnotonic Hull Consistency* (Mohc), that exploits monotonicity of functions. The propagation is standard, but the Mohc-Revise procedure, used to filter/contract the variable domains w.r.t. an individual constraint, uses monotonic versions of the classical HC4-Revise and BoxNarrow procedures.

Mohc-Revise appears to be the first *adaptive* revise procedure ever proposed in constraint programming. Also, when a function is monotonic w.r.t. every variable, Mohc-Revise is proven to compute the optimal/sharpest box enclosing all the solutions of the corresponding constraint (hull consistency). Very promising experimental results suggest that Mohc has the potential to become an alternative to the state-of-the-art HC4 and Box algorithms.

1 Introduction

Les résolveurs de contraintes sur intervalles traitent les systèmes d'équations et d'inégalités sur les réels. Leur fiabilité et leur performance croissantes leur permettent de résoudre des systèmes qui apparaissent dans divers domaines comme la robotique [12], les systèmes dynamiques de commande robuste ou la localisation de robots autonomes [9].

Deux principaux types d'algorithmes de contraction permettent de réduire les domaines des variables. Les algorithmes de type *Newton sur intervalles* sont des généralisations aux intervalles des méthodes standard d'analyse numérique [13]. Les algorithmes de contraction/filtrage provenant de la programmation par contraintes sont aussi au cœur des résolveurs sur intervalles. Les algorithmes de propagation de contraintes HC4 et Box [3, 15] sont très souvent utilisés dans les stratégies de résolution. Ils réalisent une boucle de propagation et réduisent les domaines des variables (c-à-d améliorent leurs bornes) avec des procédures de *révision* spécifiques (appelées HC4-Revise et BoxNarrow) qui traitent les contraintes individuellement.

HC4-Revise calcule la boîte optimale englobant toutes les solutions d'une contrainte c quand la fonction correspondante est continue et que chaque variable apparaît une seule fois dans c. Si *une* variable apparaît plusieurs fois dans c, HC4-Revise n'est généralement *pas* optimal. Dans ce cas, BoxNarrow calcule une enveloppe plus étroite. Le nouvel algorithme présenté dans cet article, appelé Mohc-Revise, essaie de traiter le cas général où *plusieurs* variables ont des occurrences multiples dans c.

Quand une fonction est monotone par rapport à une variable x dans une boîte, il est bien connu que l'ex-

tension aux intervalles de f basée sur la monotonie ne produit pas de surestimation due aux occurrences multiples de x. `Mohc-Revise` exploite cette propriété pour améliorer la contraction. La monotonie n'est généralement vraie que pour quelques paires (f, x) au début de la recherche, mais peut être détectée pour plus de paires quand on traite des boîtes plus petites en descendant dans l'arbre de recherche.

Après quelques rappels sur les CSP numériques, nous présentons l'algorithme `Mohc-Revise` et établissons quelques conditions qui en augmentent l'efficacité. Nous montrons que, si une fonction est monotone en toutes ses variables, `Mohc-Revise` calcule alors la boîte optimale englobant toutes les solutions de la contrainte (propriété de *Hull-consistance*). Des expérimentations soulignent les performances de `Mohc`.

2 Intervalles et CSP numériques

Les intervalles permettent des calculs fiables en gérant les arrondis des calculs sur les nombres en virgule flottante.

Définition 1 (Définitions de base, notations)
Un **intervalle** $[v] = [a, b]$ *est l'ensemble* $\{x \in \mathbb{R},\ a \leq x \leq b\}$. $\mathbb{IR}$ *est l'ensemble de tous les intervalles.*
$\underline{v} = a$ *(resp.* $\overline{v} = b$*) est le nombre à virgule flottante qui est la* **borne gauche** *(resp. la* **borne droite***) de* $[v]$.
$\texttt{Mid}([v])$ *est le* **milieu** *de* $[v]$.
$\texttt{Diam}([v]) := \overline{v} - \underline{v}$ *est le* **diamètre***, ou taille, de* $[v]$.
Une **boîte** $[V] = [v_1], ..., [v_n]$ *représente le produit cartésien* $[v_1] \times ... \times [v_n]$.

L'*arithmétique d'intervalles* a été définie pour étendre à $\mathbb{IR}$ les fonctions élémentaires sur $\mathbb{R}$ [13]. Par exemple, la somme sur intervalles est définie par $[v_1]+[v_2] = [\underline{v_1}+\underline{v_2}, \overline{v_1}+\overline{v_2}]$. Quand une fonction f est la composition de fonctions élémentaires, une *extension* de f aux intervalles doit être définie pour assurer un calcul d'image conservatif.

Définition 2 (Extension d'une fonction à $\mathbb{IR}$)
Soit une fonction $f : \mathbb{R}^n \to \mathbb{R}$.
$[f] : \mathbb{IR}^n \to \mathbb{IR}$ *est une* **extension** *de* f *aux intervalles si :*

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

L' **extension naturelle** $[f]_N$ d'une fonction réelle f correspond à l'utilisation directe de l'arithmétique d'intervalles. L'*extension par monotonie* est particulièrement utile quand une fonction f est *monotone* par rapport à une variable v dans une boîte donnée $[V]$, ce qui est le cas si l'évaluation de la dérivée partielle de f par rapport à v est positive (ou négative) en tout point de $[V]$. Par abus de langage, nous dirons parfois que v est monotone.

Définition 3 (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 :*

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

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

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

La monotonie des fonctions est généralement utilisée comme un **test d'existence** calculant si 0 appartient à l'intervalle image d'une fonction. Elle a aussi été utilisée dans les CSP numériques quantifiés pour contracter facilement une variable quantifiée universellement qui est monotone [8].

Considérons par exemple :
$f(x_1, x_2, w) = -x_1^2 + x_1x_2 + x_2w - 3w$ dans la boîte $[V] = [6, 8] \times [2, 4] \times [7, 15]$.

$[f]_N([x_1], [x_2], [w]) = -[6, 8]^2 + [6, 8] \times [2, 4] + [2, 4] \times [7, 15] - 3 \times [7, 15] = [-83, 35]$.

$\frac{\partial f}{\partial x_1}(x_1, x_2) = -2x_1 + x_2$, et $[\frac{\partial f}{\partial x_1}]_N([6, 8], [2, 4]) = [-14, -8]$. Comme $[-14, -8] < 0$, nous en déduisons que f est décroissante par rapport à x_1. Avec le même raisonnement, nous déduisons que x_2 est croissante. Finalement, $0 \in [\frac{\partial f}{\partial w}]_N([x_1], [x_2], [w]) = [-1, 1]$, ainsi w n'est pas détectée monotone. Suivant la définition 3, l'évaluation par monotonie donne :
$[f]_M([V]) = \left[\underline{[f](\overline{x_1}, \underline{x_2}, [w])}, \overline{[f](\underline{x_1}, \overline{x_2}, [w])}\right]$
$= \left[\underline{[f](8, 2, [7, 15])}, \overline{[f](6, 4, [7, 15])}\right] = [-79, 27]$

2.1 Le problème de dépendance (occurrences multiples)

Le *problème de dépendance* est le point d'achoppement de l'arithmétique d'intervalles. Il est dû au fait que les *occurrences multiples* d'une même variable dans une expression sont traitées comme des variables différentes par l'arithmétique d'intervalles. Dans notre

exemple, il explique pourquoi l'intervalle image calculé par $[f]_M$ est différent et plus étroit que celui produit par $[f]_N$. De plus, si on utilisait une forme factorisée comme $-x_1^2 + x_1x_2 + (x_2 - 3)w$, on obtiendrait une image encore meilleure. Le problème de dépendance rend en fait NP-difficile le problème de trouver l'intervalle image optimal d'un polynôme [10]. (L'extension correspondante est notée $[f]_{opt}$.) Le fait que l'extension par monotonie remplace les intervalles par leurs bornes explique la proposition suivante.

Proposition 1 *Soit f une fonction continue sur $[V]$. Alors,*

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

De plus, si f est monotone dans la boîte $[V]$ par rapport à toutes ses variables apparaissant plusieurs fois *dans f, alors l'extension par monotonie calcule l'image optimale :*

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

2.2 CSP numériques

L'algorithme `Mohc` présenté dans cet article contribue à la résolution de systèmes de contraintes non linéaires ou CSP numériques.

Définition 4 (NCSP)
Un **CSP numérique** $P = (V, C, [V])$ *comprend un ensemble de contraintes C, un ensemble V de n variables ayant pour domaines $[V] \in \mathbb{IR}^n$.*
Une **solution** $S \in [V]$ *de P satisfait toutes les contraintes de C.*

Pour trouver toutes les solutions d'un NCSP avec des techniques par intervalles, le processus de résolution commence avec une boîte initiale représentant l'espace de recherche et construit un arbre de recherche, suivant un schéma *Brancher & Contracter*.

- *Brancher :* la boîte courante est **bissectée** sur une dimension (variable) et produit deux sous-boîtes.
- *Contracter :* les algorithmes de filtrage (aussi appelés de **contraction**) réduisent les bornes de la boîte sans perdre de solution.

Le processus se termine avec des **boîtes atomiques** de taille au plus ω sur chaque dimension.

Les algorithmes de contraction comprennent les algorithmes de type *Newton sur intervalles* issus de la communauté numérique d'*analyse par intervalles* [13] ainsi que des algorithmes venant de la programmation par contraintes. L'algorithme de contraction présenté dans cet article prend en compte la monotonie des fonctions, en adaptant les procédures classiques `HC4-Revise` et `BoxNarrow` de programmation par contraintes sur intervalles. L'algorithme `HC4` effectue une boucle de propagation de type AC3. Sa procédure de révision, appelée `HC4-Revise`, traverse deux fois l'arbre représentant l'expression mathématique de la contrainte pour contracter les intervalles des variables de la contrainte. Un exemple est donné à la figure 1.

`Box` est un autre algorithme de propagation. Pour chaque paire (f, x), où f est une fonction du NCSP considéré et x est une variable de f, `BoxNarrow` remplace d'abord les a autres variables de f par leurs intervalles $[y_1], ..., [y_a]$. Ensuite, la procédure réduit les bornes de $[x]$ de telle sorte que la nouvelle borne gauche (resp. droite) soit la solution la plus à gauche (resp. à droite) de l'équation $f(x, [y_1], ..., [y_a]) = 0$. Les procédures existantes utilisent un principe de *rognage* qui élimine de $[x]$ les sous-intervalles $[x_i]$ aux bornes de $[x]$ qui ne satisfont pas la contrainte.

Contracter de manière optimale une boîte par rapport à une contrainte individuelle revient à atteindre la propriété que l'on appelle **hull-consistance**. Comme pour le calcul de l'intervalle image optimal, la *hull*-consistance n'est pas atteignable en temps polynomial, à cause du problème de la dépendance. `HC4-Revise` calcule la hull-consistance des contraintes sans variable avec occurrences multiples, à condition que la fonction et ses fonctions de projection soient continues. La *Box-consistance* obtenue par l'algorithme `BoxNarrow` est plus forte [7] et produit la *hull*-consistance quand la contrainte ne contient qu'*une* variable avec occurrences multiples. En effet, le processus de rognage réalisé par `BoxNarrow` sur une variable x limite fortement l'effet de surestimation sur x. Par contre, il n'est *pas optimal* dans le cas où d'autres variables y_i possèdent aussi des occurrences multiples.

Ces algorithmes sont parfois utilisés dans nos expérimentations comme des sous-contracteurs de `3BCID` [14], une variante de `3B` [11]. `3B` utilise un principe de réfutation par rognage. Un sous-intervalle $[x_i]$ à une borne d'un intervalle $[x]$ est supprimé si l'appel au sous-contracteur (par exemple `HC4`) sur le sous-problème correspondant (où $[x]$ est remplacé par $[x_i]$) aboutit à un échec (sous-problème sans solution). On procède ainsi de chaque côté jusqu'à ce qu'on obtienne une tranche ne pouvant pas être supprimée ou que toutes les tranches l'aient été.

3 L'algorithme Mohc

L'algorithme de consistance d'enveloppe monotone (*MOnotonic Hull-Consistency*, `Mohc`) est un nouvel algorithme de propagation de contraintes qui exploite la monotonie des fonctions pour mieux contracter une boîte. La boucle de propagation est exactement le

même algorithme de type AC3 mis en œuvre par HC4 et Box. Sa nouveauté réside dans la procédure Mohc-Revise traitant une contrainte $f(V) = 0$ individuellement[1] et décrite à l'algorithme 1.

Algorithm 1 Mohc-Revise (**in-out** $[V]$; **in** f, V, ρ_{mohc}, τ_{mohc}, ϵ)

```
HC4-Revise (f(V) = 0, [V])
if MultipleOccurrences(V) and ρ_mohc[f] < τ_mohc
then
    (X, Y, W, f_max, f_min, [G]) ← PreProcessing(f, V, [V])
    MinMaxRevise ([V], f_max, f_min, Y, W)
    MonotonicBoxNarrow ([V], f_max, f_min, X, [G], ε)
end if
```

Mohc-Revise commence par appeler la procédure bien connue et peu coûteuse HC4-Revise. Les procédures de contraction par monotonie (MinMaxRevise et MonotonicBoxNarrow) ne sont appelées que si V contient au moins une variable apparaissant plusieurs fois (fonction MultipleOccurrences). L'autre condition rend Mohc-Revise adaptatif. Cette condition dépend d'un paramètre utilisateur τ_{mohc} détaillé dans la partie suivante. Le second paramètre ϵ de Mohc-Revise est un ratio de précision utilisé par MonotonicBoxNarrow.

La procédure PreProcessing calcule le gradient de f. Le gradient est stocké dans le vecteur $[G]$ et utilisé pour répartir les variables de V en trois sous-ensembles X, Y et W :

- les variables de X sont monotones et ont des occurrences multiples dans f,
- les variables de Y n'ont qu'une seule occurrence dans f (elles peuvent être monotones),
- les variables w de W apparaissent plusieurs fois dans f et ne sont pas détectées monotones, c.-à-d. $0 \in [\frac{\partial f}{\partial w}]_N([V])$.

La procédure PreProcessing détermine aussi les deux fonctions f_{min} et f_{max}, introduites dans la définition 3, qui approximent f en utilisant sa monotonie.

Les deux procédures suivantes sont au cœur de Mohc-Revise et sont détaillées plus loin. Utilisant les monotonies de f_{min} et f_{max}, MinMaxRevise contracte $[Y]$ et $[W]$ tandis que MonotonicBoxNarrow contracte $[X]$.

HC4-Revise, MinMaxRevise et MonotonicBoxNarrow calculent quelquefois une boîte vide $[V]$, prouvant alors l'absence de solution. Une exception terminant la procédure est alors levée.

A la fin, si Mohc-Revise a contracté un intervalle de $[W]$ (de plus qu'un ratio donné par le paramètre utilisateur τ_{propag}), alors la contrainte est mise dans la queue de propagation pour être traitée de nouveau par un appel suivant à Mohc-Revise. Sinon, nous savons qu'un point fixe en terme de filtrage a été atteint (voir lemmes 2 et 4).

[1] La procédure peut facilement être étendue pour traiter une inégalité.

3.1 La procédure MinMaxRevise

Nous savons que :

$(\exists X \in [X])(\exists Y \in [Y])(\exists W \in [W]) : \ f(X \cup Y \cup W) = 0$
$\Longrightarrow \ f_{min}(Y \cup W) \leq 0 \ \ et \ \ 0 \leq f_{max}(Y \cup W)$

La contraction apportée par MinMaxRevise est simplement obtenue en appelant HC4-Revise sur les contraintes $f_{min}(Y \cup W) \leq 0$ et $0 \leq f_{max}(Y \cup W)$ pour réduire les intervalles des variables de Y et W (voir l'algorithme 2).

Algorithm 2 MinMaxRevise (**in-out** $[V]$; **in** f_{max}, f_{min}, Y, W)

```
HC4-Revise(f_min(Y∪W) ≤ 0, [V]) /* MinRevise */
HC4-Revise(f_max(Y∪W) ≥ 0, [V]) /* MaxRevise */
```

La figure 1 illustre comment MinMaxRevise contracte la boîte $[x] \times [y] = [4, 10] \times [-80, 14]$ par rapport à la contrainte : $f(x, y) = x^2 - 3\,x + y = 0$.

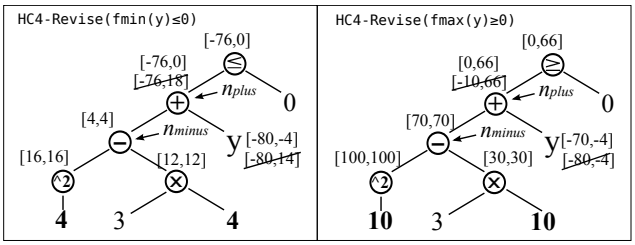

FIG. 1 – MinRevise (à gauche) et MaxRevise (à droite) appliquées à $x^2 - 3\,x + y = 0$.

La figure 1-gauche montre la première étape de MinMaxRevise. L'arbre représente l'inégalité $f(4, y) = f_{min}(y) \leq 0$. HC4-Revise procède en deux phases. La phase d'évaluation évalue chaque nœud de bas en haut avec l'arithmétique d'intervalles et attache le résultat au nœud. La seconde phase, à cause de l'inégalité, commence par intersecter l'intervalle du haut $[-76, 18]$ avec $[-\infty, 0]$ et, si le résultat est non vide, redescend dans l'arbre en appliquant *les fonctions de projection ("inverses")*. Par exemple, comme $n_{plus} = n_{minus} + y$, la fonction inverse de cette somme est la différence $[y] \leftarrow [y] \cap ([n_{plus}] - [n_{minus}]) = [-80, 14] \cap ([-76, 0] - [4, 4]) = [-80, -4]$. Suivant le même principe, MaxRevise applique HC4-Revise à $f(10, y) = f_{max}(y) \geq 0$ et réduit $[y]$ à $[-70, -4]$ (cf. la figure 1-droite).

Notons qu'un appel direct à HC4-Revise sur la contrainte $x^2 - 3\,x + y = 0$ (sans utiliser la monotonie de f) n'aurait apporté aucune contraction à $[x]$ ni à $[y]$.

3.2 La procédure MonotonicBoxNarrow

Cette procédure effectue une boucle sur chaque variable monotone x_i de X pour contracter $[x_i]$.

À chaque itération, elle travaille avec deux fonctions sur intervalles, dans lesquelles toutes les variables de X, sauf x_i, ont été remplacées par une borne de l'intervalle correspondant :

$[f^{x_i}_{min}](x_i) = [f]_N(x_1^-, ..., x_{i-i}^-, x_i, x_{i+1}^-, ..., x_n^-, [Y], [W])$

$[f^{x_i}_{max}](x_i) = [f]_N(x_1^+, ..., x_{i-i}^+, x_i, x_{i+1}^+, ..., x_n^+, [Y], [W])$

Comme Y et W ont été remplacées par leurs domaines, $[f^{x_i}_{max}]$ et $[f^{x_i}_{min}]$ sont des fonctions intervalles de la seule variable x_i (cf. la figure 2).

`MonotonicBoxNarrow` appelle deux sous-procédures :

- Si x_i est croissante, alors sont appelées :
  - `LeftNarrowFmax` sur $[f^{x_i}_{max}]$ pour améliorer $\underline{x_i}$,
  - `RightNarrowFmin` sur $[f^{x_i}_{min}]$ pour améliorer $\overline{x_i}$.
- Si x_i est décroissante, alors sont appelés :
  - `LeftNarrowFmin` sur $[f^{x_i}_{min}]$ pour améliorer $\underline{x_i}$,
  - `RightNarrowFmax` $[f^{x_i}_{max}]$ pour améliorer $\overline{x_i}$.

Nous détaillons dans l'algorithme 3 comment la borne gauche de $[x]$ est améliorée par la procédure `LeftNarrowFmax` qui utilise $[f^x_{max}]$.

Algorithm 3 `LeftNarrowFmax` (**in-out** $[x]$; **in** $[f^x_{max}]$, $[g]$, ϵ)

if $\overline{[f^x_{max}]_N(\underline{x})} < 0$ /* test d'existence */ **then**
  $size \leftarrow \epsilon \times$ `Diam`$([x])$
  $[l] \leftarrow [x]$
  while `Diam`$([l]) > size$ **do**
    $x_m \leftarrow$ `Mid`$([l])$; $z_m \leftarrow \overline{[f^x_{max}](x_m)}$
    /* $z_m \leftarrow \underline{[f^x_{min}](x_m)}$ dans `{Left|Right}NarrowFmin` */
    $[l] \leftarrow [l] \cap x_m - \frac{z_m}{[g]}$ /* itération de Newton */
  end while
  $[x] \leftarrow [\underline{l}, \overline{x}]$
end if

Le processus est illustré par la fonction représentée graphiquement à la figure 2. Le but est de contracter $[l]$ (initialisé à $[x]$) pour obtenir un encadrement étroit du point L. L'utilisateur spécifie le paramètre de précision ϵ (comme un rapport de diamètres d'intervalles) donnant la qualité de l'approximation. `LeftNarrowFmax` conserve seulement $\underline{l}$ à la fin, comme le montrent la dernière ligne de l'algorithme 3 et l'étape 4 à la figure 2.

Un test d'existence préliminaire vérifie que $\overline{[f^x_{max}]_N(\underline{x})} < 0$, c.-à-d. que le point A sur la figure 2 est en dessous de zéro. Sinon, $[f^x_{max}]_N \geq 0$ est satisfait en $\underline{x}$ et $[x]$ ne peut être réduit, ce qui aboutit à la terminaison de la procédure. Nous effectuons un processus dichotomique jusqu'à ce que l'on obtienne

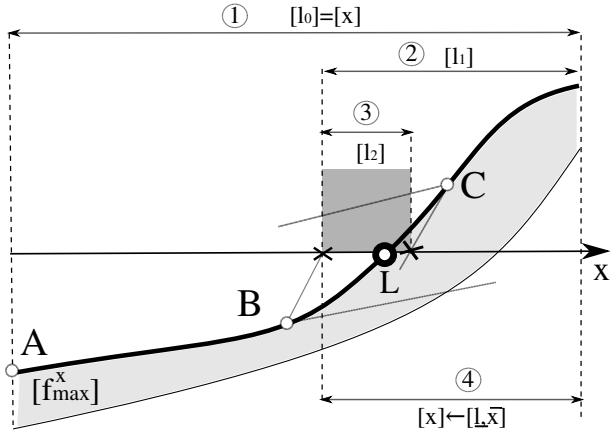

FIG. 2 – Itérations de Newton sur intervalles pour réduire $\underline{x}$.

`Diam`$([l]) \leq size$. Des itérations de l'algorithme classique *Newton sur intervalles* univarié sont lancées à partir du point milieu x_m de $[l]$, comme sur la figure 2 :

1. à partir du point B (milieu de $[l_0]$, c.-à-d., $[l]$ à l'étape 0) et
2. à partir du point C (milieu de $[l_1]$).

Graphiquement, une itération de *Newton sur intervalles* univarié intersecte $[l]$ avec la projection sur l'axe x d'un cône (deux demi-droites à partir de B, puis de C). Les pentes de ces droites sont égales aux bornes de la dérivée partielle $[g] = [\frac{\partial f^x_{max}}{\partial x}]_N([x])$. Notons que le cône forme un angle d'au plus 90 degrés car la fonction est monotone et $[g]$ est positif. Ceci explique pourquoi `Diam`$([l])$ est divisé au moins par 2 à chaque itération.

Lemme 1 *Soit ϵ la précision exprimée comme un rapport de diamètres d'intervalles. Alors,* `LeftNarrowFmax` *et les procédures symétriques terminent et s'exécutent en un temps en $O(log(\frac{1}{\epsilon}))$.*

Observons que les itérations de Newton appelées à l'intérieur de `LeftNarrowFmax` et `RightNarrowFmax` travaillent sur $z_m = \overline{[f^x_{max}](x_m)}$, qui est une courbe (en gras sur la figure), et non la fonction sur intervalles $[f^x_{max}](x_m)$.

3.3 Points notables de Mohc-Revise

Comment rendre Mohc-Revise adaptatif

Le paramètre utilisateur $\tau_{mohc} \in [0, 1]$ permet d'appeler plus ou moins souvent, au cours de la recherche arborescente, les procédures utilisant la monotonie (voir l'algorithme 1). Pour chaque contrainte, les procédures exploitant la monotonie de f ne sont appelées que si $\rho_{mohc}[f] < \tau_{mohc}$. Cette condition demande que l'image d'une fonction par l'évaluation par monotonie

soit plus étroite que celle donnée par l'évaluation naturelle d'un facteur $\rho_{mohc}[f]$ au moins égal au paramètre τ_{mohc} :

$$\rho_{mohc}[f] = \frac{Diam([f]_M([V]))}{Diam([f]_N([V]))}$$

Comme nos expérimentations le confirment, ce ratio est pertinent pour les phases d'évaluation de `MinRevise` et `MaxRevise`, et aussi pour `MonotonicBoxNarrow` qui effectue beaucoup d'évaluations.

ρ_{mohc} est calculé dans une procédure de prétraitement appelée après chaque bissection. Comme de plus en plus de cas de monotonie apparaissent au fur et à mesure que l'on descend dans l'arbre de recherche (les boîtes devenant plus petites), `Mohc-Revise` est capable d'activer de manière adaptative la machinerie liée à la monotonie. `Mohc-Revise` apparaît ainsi comme la première procédure de révision adaptative en programmation par contraintes (sur intervalles).

Le regroupement d'occurrences pour utiliser plus de monotonie

Un appel à une nouvelle procédure appelée *Regroupement d'occurrences* a en fait été ajouté à `Mohc-Revise` juste après le prétraitement. Si f n'est *pas* monotone par rapport à une variable x, il est toutefois possible que f soit monotone par rapport à un sous-groupe des occurrences de x. Pour trouver de tels sous-groupes d'occurrences croissantes et décroissantes, cette procédure utilise une approximation de f basée sur l'évaluation de Taylor et résout à la volée un programme linéaire. Ceci permet d'améliorer l'évaluation par monotonie de f. Des détails et une évaluation expérimentale se trouvent dans [2].

4 Propriétés

Proposition 2 (Complexité en temps) *Considérons une contrainte c comprenant n variables et e opérateurs unaires et binaires ($n \leq e$). Soit ϵ la précision exprimée comme un ratio de diamètres d'intervalles. Alors,*
la complexité en temps de Mohc-Revise est en $O(n\, e\, log(\frac{1}{\epsilon})) = O(e^2\, log(\frac{1}{\epsilon}))$.

La complexité en temps est dominée par `MonotonicBoxNarrow` (voir le lemme 1). Un appel à `HC4-Revise` et un calcul du gradient sont tous deux en $O(e)$ [3].

Proposition 3 *Soit $c : f(X) = 0$ une contrainte telle que f est continue, dérivable et monotone par rapport à chaque variable dans la boîte $[X]$. Alors, avec une précision ϵ,* `MonotonicBoxNarrow` *calcule la hull-consistance de c.*

On peut trouver les preuves dans [1] et [5]. Cependant, la nouvelle proposition 4 ci-après est plus forte car les variables apparaissant une seule fois (Y) sont traitées par `MinMaxRevise` et non par `MonotonicBoxNarrow`.

Proposition 4 *Soit $c : f(X, Y) = 0$ une contrainte, où les variables dans Y apparaissent une seule fois dans f. Si f est continue, dérivable et monotone par rapport à chaque variable dans la boîte $[X \cup Y]$, alors, avec une précision ϵ,* `Mohc-Revise` *calcule la hull-consistance of c.*

On peut trouver la démonstration complète dans [1]. Il est aussi prouvé qu'aucune hypothèse de monotonie n'est requise pour les variables de Y, à condition que `Mohc-Revise` utilise la variante combinatoire `TAC-Revise` [6] de `HC4-Revise`.

Les lemmes 2, 3 et 4 ci-dessous permettent de démontrer les propositions 3 et 4. Ils démontrent aussi la correction de `Mohc-Revise`.

Lemme 2 *Quand* `MonotonicBoxNarrow` *réduit l'intervalle d'une variable $x_i \in X$ en utilisant $[f^{x_i}_{max}]$ (resp. $[f^{x_i}_{min}]$), alors, pour tout $j \neq i$, $[f^{x_j}_{min}]$ (resp. $[f^{x_j}_{max}]$) ne peut pas apporter de contraction additionnelle à l'intervalle $[x_j]$.*

Le lemme 2 est une généralisation de la proposition 1 de [5] aux fonctions sur intervalles ($[f^{x_i}_{max}]$ et $[f^{x_i}_{min}]$).

Lemme 3 *Si $0 \in [z] = [f_{max}]([Y \cup W])$ (resp. $0 \in [z] = [f_{min}]([Y \cup W])$), alors* `MonotonicBoxNarrow` *ne peut pas contracter un intervalle $[x_i]$ ($x_i \in X$) en utilisant $[f^{x_i}_{min}]$ (resp. $[f^{x_i}_{max}]$).*

Lemme 4 *Si* `MonotonicBoxNarrow` *(suivant un appel à* `MinMaxRevise`*) contracte $[x_i]$ (avec $x_i \in X$), alors un second appel à* `MinMaxRevise` *ne peut pas contracter davantage $[Y \cup W]$.*

Les lemmes 2 et 4 expliquent pourquoi il n'est pas nécessaire d'effectuer de boucle dans `MohcRevise` pour atteindre un point fixe en termes de filtrage.

4.1 Démonstrations des lemmes 3 et 4

La figure 3 illustre ces démonstrations dans le cas où f est croissante. Nous distinguons deux cas suivant la borne droite initiale de l'intervalle $[x_i]$.

Dans le lemme 3, nous avons $0 \in [z] = [f_{max}]([Y \cup W])$, c.-à-d. $\underline{z} \leq 0 \leq \overline{z}$. Cette condition est en particulier vraie quand `MaxRevise` apporte une contraction. On peut vérifier que $\overline{x_i}$ ne peut être amélioré par `RightNarrowFmin` : puisque $\overline{x_i}$ est une solution de $[f^{x_i}_{max}](x_i) = 0$ (le segment en gras sur la figure 3),

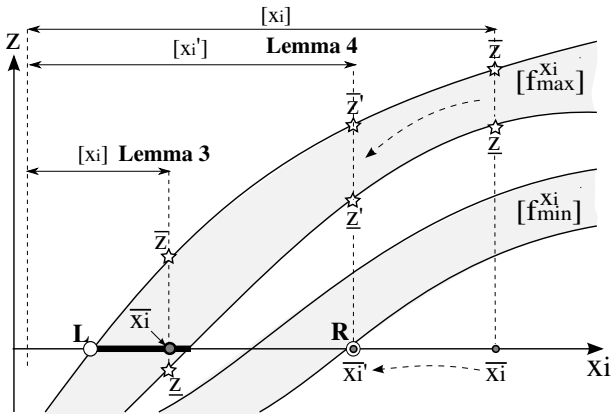

FIG. 3 – Démonstration des lemmes 3 et 4 montrant la dualité de `MinMaxRevise` et `MonotonicBoxNarrow`. $[z]$ est l'image de $[f_{max}]$ obtenue par la phase d'évaluation de `MaxRevise`.

$\overline{x_i}$ satisfait aussi la contrainte $[f_{min}^{x_i}](x_i) \leq 0$ qui est utilisée par `RightNarrowFmin`.

Dans le lemme 4, nous avons $0 < [z] = [f_{max}]([Y \cup W])$ (`MaxRevise` n'apporte pas de contraction). Après la contraction réalisée par `RightNarrowFmin`, la borne droite de l'intervalle devient $\overline{x_i'}$. Une nouvelle évaluation de $[f_{max}]([Y \cup W])$ donne $[z']$ qui est encore au dessus de 0, de telle sorte qu'un deuxième appel à `MaxRevise` n'apporterait pas de contraction additionnelle. □

4.2 Amélioration de MonotonicBoxNarrow

Finalement, les lemmes 2 et 3 apportent des conditions simples pour éviter des appels à `LeftNarrowFmax` (et aux procédures symétriques) à l'intérieur de `MonotonicBoxNarrow`.

Avec l'ajout de ces conditions, comme le montrent les tests détaillés dans [1], 35% du temps CPU de `Mohc-Revise` est passé dans `MinMaxRevise` alors que seulement 9% est passé dans la procédure plus coûteuse `MonotonicBoxNarrow` (entre 1% et 18% selon l'instance).

5 Expérimentations

Nous avons implanté `Mohc` dans la bibliothèque en `C++ Ibex` [4] de résolution par intervalles. Tous les algorithmes concurrents, `HC4`, `Box`, `Octum` [5], `3BCID(HC4)`, `3BCID(Box)` et `3BCID(Octum)` sont aussi disponibles dans `Ibex`, ce qui rend la comparaison équitable.

`Mohc` et ses compétiteurs ont été testés sur la même machine `Intel 6600 2.4 GHz` sur 17 NCSP avec un nombre fini de solutions ponctuelles disponibles sur le site de COPRIN[2]. Nous avons sélectionné tous les NCSP avec des variables à occurrences multiples qui se trouvent dans les deux premières parties (systèmes *polynomiaux* et *non polynomiaux*) du site. Nous avons ajouté `Brent`, `Butcher`, `Direct Kin.` et `Virasaro` qui proviennent de la partie appelée *problèmes difficiles*.

Toutes les stratégies de résolution utilisent comme choix de variable le tour de rôle. Entre deux branchements dans l'arbre de recherche, trois procédures sont appelées en séquence. D'abord, un test d'existence utilisant la monotonie, amélioré par le regroupement d'occurrences, vérifie que l'image de chaque fonction contient zéro[3]. Ensuite, le contracteur est appelé : `Mohc`, `3BCID(Mohc)`, ou l'un des concurrents cités. Enfin, l'algorithme *Newton sur intervalles* est appelé si la boîte courante a un diamètre de 10 ou moins. Tous les paramètres ont été fixés à des valeurs par défaut. Le ratio de précision du rognage dans `3B` et `Box` est 10% ; une contrainte est mise dans la queue de propagation si l'intervalle d'une de ses variables est réduit de plus de plus de $\tau_{propag} = 1\%$ avec tous les contracteurs sauf `3BCID(HC4)` et `3BCID(Mohc)` (10%). Pour `Mohc`, le paramètre τ_{mohc} a été fixé à 70% ou 99%.
ϵ vaut 3% dans `Mohc` et 10% dans `3BCID(Mohc)`.

5.1 Résultats

Le tableau 1 compare le temps CPU utilisé et le nombre de points de choix obtenus par `Mohc` et `3BCID(Mohc)` avec ceux obtenus par leurs concurrents. La dernière colonne donne le gain obtenu par `Mohc`, c.-à-d. $Gain = \frac{tempsCPU\,(meilleur\ compétiteur)}{tempsCPU\,(meilleure\ stratégie\ de\ type\ Mohc)}$

Le tableau montre les très bons résultats obtenus par `Mohc`, en termes de pouvoir de filtrage (petit nombre de points de choix) et de temps CPU. Les résultats obtenus par `3BCID(Box)`, `Octum` et `3BCID(Octum)` ne sont pas indiqués car ces méthodes ne se sont pas avérées compétitives avec `Mohc`. Par exemple, `Octum` est moins rapide que `Mohc` d'un ordre de grandeur. La supériorité de `Mohc` sur `Box` montre qu'il vaut mieux un plus grand effort de filtrage (appel à BoxNarrow) moins souvent, c.-à-d. quand on a détecté qu'une variable était monotone. `Mohc` et `HC4` obtiennent des résultats similaires sur 9 des 17 problèmes testés. Avec $\tau_{mohc} = 70\%$, on peut noter que la perte en performance de `Mohc` (resp. `3BCID(Mohc)`) par rapport à `HC4` (resp. `3BCID(HC4)`) sur ces problèmes est négligeable. Elle est inférieure à 5%, sauf pour `Katsura` (25%).

Sur 6 NCSP, `Mohc` obtient un gain compris entre 2.4 et 8. Sur `Butcher` et `Direct kin.`, on observe même un gain de resp. 163 et 49. Sans le test d'existence par

[2] www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html

[3] Ce test d'existence ne prend qu'une petite part du temps total (en général moins de 10% et moins de 1% pour `3BCID`), tout en améliorant grandement la performance des concurrents.

TAB. 1 – Résultats expérimentaux. La première colonne inclut le nom du système, le nombre d'équations et le nombre de solutions. Les autres colonnes donnent le temps CPU en secondes (en haut) et le nombre de points de choix (en bas) pour tous les compétiteurs. Les meilleurs temps sont en gas.

NCSP	HC4	Box	3BCID(HC4)	Mohc 70%	Mohc 99%	3BCID(Mohc) 70%	3BCID(Mohc) 99%	Gain
Butcher	>4e+5	>4e+5	282528	>4e+5	>4e+5	5431	**1722**	163
8 3			1.8e+8			2.2e+6	288773	623
Directkin.	>2e+4	>2e+4	17507	2560	2480	428	**356**	49.1
11 2			1.4e+6	777281	730995	8859	5503	253
Virasoro	>2e+4	>2e+4	7173	1180	1089	1051	**897**	8.00
8 224			2.5e+6	805047	715407	71253	38389	66.8
Yamam.1	32.4	12.6	11.7	19.2	27.0	**2.2**	2.87	5.30
8 7	29513	3925	3017	24767	29973	345	295	10.2
Geneig	1966	3721	390	463	435	107	**81.1**	4.81
6 10	4.1e+6	1.3e+6	161211	799439	655611	13909	6061	26.6
Hayes	163	323	41.6	30.9	27.6	17.0	**13.8**	3.02
8 1	541817	214253	17763	73317	49059	4375	1679	10.6
Trigo1	93	332	151	**30**	30.6	57.7	73.2	3.10
10 9	5725	6241	2565	1759	1673	459	443	5.79
Fourbar	863	2441	1069	361	**359**	366	373	2.40
4 3	1.6e+6	1.1e+6	965343	437959	430847	58571	45561	21.2
Pramanik	26.9	91.9	35.9	30.3	25.0	**20.8**	21.3	1.29
3 2	103827	81865	69259	87961	69637	12691	8429	8.22
Caprasse	2.04	11.5	2.73	**1.87**	2.69	2.64	4.35	1.09
4 18	7671	5957	1309	4577	3741	867	383	3.42
Kin1	6.91	26.9	1.96	5.68	5.65	**1.79**	3.43	1.09
6 16	1303	689	87	1055	931	83	83	1.05
Redeco8	3769	9906	6.28	3529	2936	**6.10**	10.65	1.03
8 8	1.0e+7	7.9e+6	2441	6.8e+6	4.6e+6	2211	1489	1.64
Trigexp2	1610	>2e+4	**86.9**	1507	1027	**87**	165	1.00
11 0	1.6e+6		14299	1417759	935227	14299	7291	1.96
Eco9	39.9	94.1	**13.9**	46.8	44.2	**14.0**	26.6	0.99
9 16	115445	110423	6193	97961	84457	6025	4309	1.44
I5	9310	>2e+4	**55.9**	7107	7129	57.5	84.1	0.97
10 30	2.4e+7		10621	1.6e+7	1.5e+7	9773	8693	1.22
Brent	497	151	**18.9**	244	232	19.9	41.4	0.95
10 1008	1.8e+6	23855	3923	752533	645337	3805	3189	1.23
Katsura	182	2286	**77.8**	106	143	104	251	0.75
12 7	271493	251727	4251	98779	94249	3573	3471	1.22

monotonie appelé avant les contracteurs, on a obtenu un gain de 37 sur le système `Fourbar`.

En conclusion, la combinaison `3BCID(Mohc)` semble être très prometteuse.

5.2 Travaux connexes

Un algorithme de propagation de contraintes exploitant la monotonie se trouve dans le résolveur par intervalles ALIAS[4]. Sa procédure de révision n'utilise pas un arbre pour représenter une expression f (contrairement à `HC4-Revise`). A la place, une fonction de projection f^o_{proj} est créée pour réduire l'intervalle de chaque occurrence o et est évaluée avec une extension par monotonie $[f^o_{proj}]_M$. C'est plus coûteux que `MinMaxRevise` et n'est pas optimal puisqu'aucune procédure `MonotonicBoxNarrow` n'est utilisée.

L'article [5] décrit un algorithme de propagation de contraintes appelé `Octum`. `Mohc` et `Octum` ont été conçus indépendamment pendant le premier semestre de 2009. Pour le décrire rapidement, `Octum` appelle `MonotonicBoxNarrow` quand *toutes* les variables de la contrainte sont monotones. Comparé à `Octum` :

- `Mohc` ne demande pas qu'une fonction soit monotone par rapport à toutes ses variables simultanément ;
- `Mohc` utilise `MinMaxRevise` pour contracter rapidement les intervalles des variables (de Y) apparaissant une seule fois (voir la proposition 4) ;
- `Mohc` utilise un algorithme de regroupement d'occurrences pour détecter plus de cas de monotonie.

Une première étude expérimentale (non décrite ici) montre que la bien meilleure performance de `Mohc` est due principalement à la condition établie dans le lemme 3 (et testée durant `MinMaxRevise`), qui permet d'éviter des appels à `LeftNarrowFmax` et à ses procédures symétriques.

[4] www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html

6 Conclusion

Cet article a présenté un algorithme de propagation de contraintes utilisant la monotonie des fonctions. En utilisant les ingrédients présents dans les procédures existantes `HC4-Revise` et `BoxNarrow`, `Mohc` a le potentiel pour remplacer avantageusement `HC4` et `Box`, comme le montrent nos premières expérimentations. De plus, `3BCID(Mohc)` semble constituer une combinaison très prometteuse.

La procédure `Mohc-Revise` utilise deux paramètres utilisateurs, notamment τ_{mohc} pour régler la sensibilité à la monotonie. Un travail futur consistera à rendre `Mohc-Revise` auto-adaptatif en permettant de régler automatiquement τ_{mohc} durant la recherche combinatoire.

Références

[1] I. Araya. *Exploiting Common Subexpressions and Monotonicity of Functions for Filtering Algorithms over Intervals.* PhD thesis, Université de Nice–Sophia, 2010.

[2] I. Araya, B. Neveu, and G. Trombettoni. Une nouvelle extension de fonctions aux intervalles basée sur le regroupement d'occurrences. In *Proc. JFPC*, 2010.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.

[4] G. Chabert. `www.ibex-lib.org`, 2010.

[5] G. Chabert and L. Jaulin. Hull Consistency Under Monotonicity. In *Proc. CP, LNCS 5732*, pages 188–195, 2009.

[6] G. Chabert, G. Trombettoni, and B. Neveu. Box-Set Consistency for Interval-based Constraint Problems. In *Proc. SAC (ACM)*, pages 1439–1443, 2005.

[7] H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliable Computing*, 5(3) :213–228, 1999.

[8] A. Goldsztejn, C. Michel, and M. Rueher. Efficient Handling of Universally Quantified Inequalities. *Constraints*, 14(1) :117–135, 2009.

[9] M. Kieffer, L. Jaulin, E. Walter, and D. Meizel. Robust Autonomous Robot Localization Using Interval Analysis. *Reliable Computing*, 3(6) :337–361, 2000.

[10] V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations.* Kluwer, 1997.

[11] O. Lhomme. Consistency Techniques for Numeric CSPs. In *Proc. IJCAI*, pages 232–238, 1993.

[12] J-P. Merlet. Interval Analysis and Robotics. In *Symp. of Robotics Research*, 2007.

[13] R. E. Moore. *Interval Analysis.* Prentice-Hall, 1966.

[14] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.

[15] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica : A Modeling Language for Global Optimization.* MIT Press, 1997.