HAL
open science

Régions intérieures et linéarisations par intervalles en optimisation globale

Gilles Trombettoni, Ignacio Araya, Bertrand Neveu, Gilles Chabert

To cite this version:

Gilles Trombettoni, Ignacio Araya, Bertrand Neveu, Gilles Chabert. Régions intérieures et linéarisations par intervalles en optimisation globale. JPFC 2011, Jun 2011, Lyon, France. pp.299-306. hal-00654307

HAL Id: hal-00654307

https://enpc.hal.science/hal-00654307v1

Submitted on 21 Dec 2011

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.

Régions intérieures et linéarisations par intervalles en optimisation globale

Gilles Trombettoni, Ignacio Araya, Bertrand Neveu, Gilles Chabert
INRIA, I3S, Université Nice–Sophia (France), UTFSM (Chili),
Imagine LIGM Université Paris–Est (France), LINA, EMN (France)
Gilles.Trombettoni@inria.fr, iaraya@inf.utfsm.cl, Bertrand.Neveu@enpc.fr, Gilles.Chabert@emn.fr

Abstract

Les communautés d'analyse par intervalles et de programmation (logique) par contraintes ont exploité les intervalles pour leur capacité à représenter des ensembles infinis de solutions dans les systèmes de contraintes continus. En particulier, les boîtes ou régions *intérieures* permettent de représenter des sous-ensembles de l'espace de recherche dans lesquels *tout* point est solution. Notre première contribution est l'utilisation d'algorithmes récents et nouveaux d'extraction de régions intérieures dans la phase d'amélioration du majorant (faisable) en optimisation globale sous contraintes.

La *relaxation linéaire* est également un ingrédient majeur, utilisé notamment pour minorer la fonction objectif. Nous avons adapté la *taylorisation sur intervalles convexe* – relaxation linéaire proposée par Lin et Stadtherr – pour produire des approximations polyhédrales fiables intérieure et extérieure de l'ensemble solution ainsi qu'une linéarisation de la fonction objectif. Enfin, d'autres ingrédients originaux font partie de notre optimiseur, comme un algorithme de propagation de contraintes sur intervalles exploitant la monotonie des fonctions.

Nous proposons au final un nouveau schéma fiable d'optimisation globale continue sous contraintes. Une implantation est disponible en tant qu'extension de l'outil `Ibex` (bibliothèque libre en C++ de résolution par intervalles). En termes de performances, notre stratégie dépasse de manière significative les meilleurs optimiseurs globaux fiables.

1 Introduction

Les algorithmes de B&B sur intervalles sont utilisés pour résoudre des problèmes d'optimisation globale sous contraintes[1] de manière fiable. Ils fournissent soit une solution optimale (et le coût associé avec une erreur bornée), soit une preuve d'infaisablité. Historiquement, le B&B sur intervalles est né avec l'analyse par intervalles [13]. Des travaux pionniers sont décrits dans [13], [7] ou [9]. Au milieu des années 1990, Kearfott a conçu le résolveur `GlobSol`. Parallèlement, des chercheurs en programmation par contraintes ont développé les résolveurs `Numerica` [21] et `Icos` [11], introduisant respectivement de la propagation de contraintes sur intervalles et des relaxations linéaires fiables. Plus récemment, la communauté de programmation mathématique a proposé un résolveur, appelé ici `IBBA+`, qui intègre de la propagation de contraintes et de l'arithmétique affine [16].

Pour concilier fiabilité et bonne performance, les intervalles font face principalement à deux difficultés.

Améliorer le majorant dans l'espace faisable

La *recherche locale* est l'approche la plus utilisée pour trouver un point *faisable*[2] (une solution satisfaisant les contraintes) qui améliore la borne courante pour la fonction objectif f. Cependant, pour assurer la faisabilité en cas de contraintes d'égalité, il est nécessaire, dès lors que l'espace exploré contient potentiellement des points infaisables, d'appliquer un processus de correction et de certification après chaque itération de la recherche locale. Ce processus est basé sur des techniques par intervalles coûteuses [11]. Cette correction rend du coup impossible, en termes de perfor-

[1] Nous considérons dans cet article des problèmes de minimisation.

[2] Une seconde approche consiste à chercher les points qui annulent le gradient de f. Le minimum de f est alors soit une solution de ce problème, soit sur la frontière du domaine. Malheureusement, la prise en compte des contraintes dans cette formulation (conditions de Kuhn-Tucker) aboutit souvent à une fonction d'agrégation conséquente et inadaptée aux calculs par intervalles [7].

mance, la compétition avec des résolveurs *non* fiables d'optimisation globale, comme `Baron` [19].

Dans cet article, nous proposons une approche radicalement différente où l'amélioration de la borne se fait en explorant des régions intérieures, c.-à-d. des régions où l'on prouve en amont que tous les points sont faisables. La notion de boîte intérieure a déjà été étudiée en programmation par contraintes, que ce soit pour le pavage de l'espace solution [6, 2] ou pour minimiser le nombre de contraintes numériques d'inégalité non satisfaites [17]. En revanche, cette approche n'a pas encore été exploitée dans un cadre général d'optimisation globale sous contraintes d'égalité et d'inégalité.

Calcul d'un minorant par une relaxation linéaire fiable

Tous les résolveurs existants calculent, à chaque nœud du B&B, une approximation convexe (en général polyhédrale) de l'espace solution. La meilleure solution de cette relaxation fournit un minorant du coût, ce dernier étant indispensable pour terminer la recherche. Rappelons que l'approximation extérieure doit contenir l'ensemble solution en dépit des erreurs de calcul dues aux nombres à virgule flottante. Or, la plupart des relaxations linéaires sont trop sophistiquées pour pouvoir être rendues fiables facilement. Des techniques spécifiques de reformulation-linéarisation (RLT) [18] sont présentées dans [9] et [11]. Elles introduisent de nouvelles variables, représentant les opérateurs de puissance et de produit, et définissent des contraintes linéaires entre ces variables. Ninin et al. utilisent, eux, l'arithmétique affine pour effectuer une linéarisation fiable de chaque opérateur [16]. Nous proposons, pour notre part, une linéarisation fiable basée sur l'évaluation de Taylor par intervalles, au premier ordre. La simplicité de cette relaxation nous a permis de mettre aussi au point une version duale pouvant extraire une région polyhédrale cette fois intérieure de l'ensemble solution et ainsi améliorer la borne courante.

1.1 Intervalles et optimisation globale sous contraintes

Un intervalle $[x_i] = [\underline{x_i}, \overline{x_i}]$ est l'ensemble des nombres réels x_i tels que $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ représente l'ensemble de tous les intervalles. La taille ou largeur de $[x_i]$ est $w([x_i]) = \overline{x_i} - \underline{x_i}$. Une boîte $[x]$ est le produit cartésien des intervalles $[x_1] \times ... \times [x_i] \times ... \times [x_n]$. Sa largeur est définie par $\max_i w([x_i])$. $\mathrm{Mid}([x])$ est le milieu de $[x]$. Un problème continu d'optimisation globale sous contraintes se définit ainsi :

Définition 1 (Optim globale sous contraintes) *Soient x un vecteur de variables $x = (x_1, ..., x_i, ...x_n)$ dans une boîte $[x]$, f une fonction à valeurs réelles $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ et $h : \mathbb{R}^n \to \mathbb{R}^p$, des fonctions à valeurs vectorielles.*

Étant donné le système $S = (f, g, h, x, [x])$, le problème d'optimisation globale sous contraintes consiste à trouver :

$$\min_{x \in [x]} \{f(x) \ t.q. \ g(x) \leq 0 \wedge h(x) = 0\}.$$

f est la **fonction objectif***; g et h sont les* **contraintes** *d'inégalité et d'égalité. Un point est dit* **faisable** *s'il satisfait les contraintes.*

Notre algorithme d'optimisation extrait des *régions intérieures* dans les boîtes extérieures classiques.

Définition 2 *Soit un système $(f, g, \emptyset, x, [x]^{out})$ ne comprenant que des contraintes d'inégalité. Une* **région intérieure** *r^{in} est un sous-ensemble faisable de $[x]^{out}$, c.-à-d. $r^{in} \subset [x]^{out}$ et tous les points $x \in r^{in}$ satisfont $g(x) \leq 0$.*

Une région intérieure $[x]^{in}$ qui est une boîte est appelée **boîte intérieure**.

Un des opérateurs de notre stratégie effectue des *linéarisations intérieures* et extrait ainsi de l'espace faisable des *polytopes intérieurs*.

L'*arithmétique d'intervalles* [13] étend à $\mathbb{IR}$ les fonctions élémentaires sur $\mathbb{R}$. Par exemple, la somme d'intervalles $([x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}])$ contient l'image de la fonction somme sur ses arguments, et cette propriété d'inclusion définit ce que nous appelons une extension aux intervalles.

Définition 3 (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 ssi :*

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

Dans notre contexte, l'expression d'une fonction f est toujours la composition de fonctions élémentaires. L'**extension naturelle** $[f]_N$ est alors simplement la composition des opérateurs correspondants de l'arithmétique d'intervalles.

Les linéarisations extérieure et intérieure proposées dans cet article sont liées à l'**extension de Taylor** au premier ordre [13], définie comme suit :

$$[f]_T([x]) = f(\dot{x}) + \sum_i \left[\frac{\partial f}{\partial x_i}\right]_N ([x]) * ([x_i] - \dot{x_i})$$

où $\dot{x}$ est un point quelconque de $[x]$, par exemple, $\mathrm{Mid}([x])$.

Exemple. Soit $f(x_1, x_2) = 3x_1^2 + x_2^2 + x_1 * x_2$ sur la boîte $[x] = [-1, 3] \times [-1, 5]$. L'évaluation naturelle donne : $[f]_N([x_1], [x_2]) = 3 * [-1, 3]^2 + [-1, 5]^2 + [-1, 3] * [-1, 5] = [0, 27] + [0, 25] + [-5, 15] = [-5, 67]$. Les dérivées partielles sont : $\frac{\partial f}{\partial x_1}(x_1, x_2) = 6x_1 + x_2$, $[\frac{\partial f}{\partial x_1}]_N([-1, 3], [-1, 5]) = [-7, 23]$, $\frac{\partial f}{\partial x_2}(x_1, x_2) = x_1 + 2x_2$, $[\frac{\partial f}{\partial x_2}]_N([x_1], [x_2]) = [-3, 13]$. L'évaluation de Taylor avec $\dot{x} = (1, 2)$ produit : $[f]_t([x_1], [x_2]) = 9 + [-7, 23] * [-2, 2] + [-3, 13] * [-3, 3] = [-76, 94]$.

1.2 Transformer les équations en inégalités

Pour traiter les égalités, une première option est suivie par la communauté d'analyse par intervalles. Elle consiste à trouver *approximativement* un point qui satisfait *exactement* les contraintes. Le résolveur renvoie une petite boîte de largeur ϵ_{sol} dans laquelle l'existence d'un point faisable à valeurs réelles est (souvent) garantie par des méthodes de type Newton par intervalles. Une seconde option consiste à trouver *exactement* un point qui satisfait *approximativement* les contraintes. Les équations sont traitées avec une erreur de précision admissible ϵ_{eq}, c'est-à-dire qu'un point faisable à virgule flottante x doit vérifier $h(x) \in [-\epsilon_{eq}, +\epsilon_{eq}]$. Toutes les contraintes peuvent alors être vues comme des inégalités : $\{g(x) \leq 0,\ h(x) - \epsilon_{eq} \leq 0,\ -h(x) - \epsilon_{eq} \leq 0\}$. Ninin et al. ont été guidés vers ce choix par l'arithmétique affine, mais nous pensons que c'est une approche pertinente pour tout résolveur d'optimisation globale. Remarquons tout d'abord que les deux approches ont un statut équivalent en termes de fiabilité. Ensuite, une précision ϵ_{eq} sur les *images* des fonctions h répond mieux au problème de la faisabilité qu'une précision ϵ_{sol} sur les inconnues. D'autre part, la plupart des équations définies par les utilisateurs sont déjà "épaisses" et n'ont pas besoin d'une relaxation additionnelle à ϵ_{eq} près. En effet, les contraintes contiennent souvent des coefficients connus avec une incertitude bornée (par exemple une imprécision dans une mesure) ou des constantes irrationnelles comme π qui ne peuvent être entrées que sous forme d'intervalles. Enfin, nos expérimentations ont mis en évidence que le traitement de ces égalités épaisses peut être efficace en pratique. La raison derrière cette bonne surprise est que cette approche permet aux résolveurs d'extraire des régions intérieures dans des continuums de solutions. *Les algorithmes d'extraction de régions intérieures et de contraction peuvent focaliser la recherche dans le petit espace solution défini par les équations épaisses.*

2 Notre B&B sur intervalles

Notre stratégie `IbexOpt` suit le schéma bien connu de séparation-évaluation (B&B) décrit dans [8] pour résoudre un problème d'optimisation globale sous contraintes. L'algorithme effectue récursivement, à partir d'une boîte initiale, des découpages jusqu'à obtenir une solution qui minimise la fonction objectif. Pendant la recherche, un minorant (généralement non faisable) de la fonction objectif est maintenu incrémentalement à partir de la liste des boîtes traitées ou en attente. Nous notons lb (pour *lower bound*) le plus petit de ces minorants. De même, ub (pour *upper bound*) désigne le coût du meilleur point faisable trouvé au cours de la recherche. Un critère d'arrêt est atteint quand $ub - lb$ est inférieur à la précision requise ϵ_{obj},[3] et le point flottant x_{ub} de coût ub est retourné. Notons que les boîtes dont la largeur est inférieure à la précision ϵ_{sol} ne sont pas remises dans la liste et que leurs minorants prennent part au calcul de lb.

A chaque itération, l'algorithme choisit dans la liste la boîte $[x]$ avec le plus petit minorant, réalisant ainsi une recherche en *meilleur d'abord*. La variable $x_i \in x$ est choisie par une heuristique, son domaine $[x_i]$ est bissecté et la procédure principale `Contract&Bound` est appliquée sur les deux sous-boîtes. On notera que l'arbre de recherche, c.-à-d. la "liste" des boîtes à traiter, est implanté par une structure de *tas* qui permet d'accéder au plus petit élément en temps constant. On trouvera plus de détails sur ce schéma dans [16].

La première tâche de notre algorithme est l'introduction d'une nouvelle variable y dans le système $(f, g, h, x, [x])$, à l'instar de Numerica [21] par exemple. Cette variable est liée aux autres par la contrainte supplémentaire $y = f(x)$. Son domaine $[y]$ est donc un intervalle qui contient l'image de la fonction objectif sur $[x]$. Cette extension du système permet de propager et rétro-propager automatiquement les contractions entre $[x]$ et les bornes globales sur le minimum. Ainsi, la boîte étendue $[x] \times [y]$ définit l'état courant de l'optimiseur. S'y ajoutent trois variables partagées par tous les nœuds de l'arbre de recherche et mises à jour de manière globale pendant la recherche : la meilleure solution courante x_{ub}, son coût ub ($f(x_{ub}) = ub$) et lb le minimum des minorants des boîtes non traitées.

2.1 Stratégie de bissection

A chaque nœud de l'arbre de recherche, une variante de l'heuristique bien connue de la *smear function* [10] permet de choisir la prochaine variable à bissecter. Étant donné un système $(f, g, h, x, [x])$, la stratégie *smear* standard choisit la variable x_i de x qui maximise

[3] Conformément aux implantations standard, ϵ_{obj} est un pourcentage de ub si $|ub| \geq 1$ et une distance absolue si $|ub| \leq 1$.

suivant les cas $\texttt{smearMax}(x_i) = Max_{f_j}\ \texttt{smear}(x_i, f_j)$ ou $\texttt{smearSum}(x_i) = \sum_{f_j} \texttt{smear}(x_i, f_j)$, où f_j représente soit la fonction objectif f, soit une contrainte de g ou h.

$\texttt{smear}(x_i, f_j)$ est une mesure de l'impact de la variable x_i sur la fonction f_j. Son calcul implique la dérivée partielle de f_j par rapport à x_i et la largeur de $[x_i]$. Plus précisément :

$$\texttt{smear}(x_i, f_j) = \left| \left[\frac{\partial f_j}{\partial x_i}\right]_N ([x]) \right| * w([x_i]).$$

Nous proposons une variante, $\texttt{smearRel}(x_i, f_j)$, qui mesure un impact *relatif*, à valeur dans $[0, 1]$:

$$\texttt{smearRel}(x_i, f_j) = \frac{\texttt{smear}(x_i, f_j)}{\sum_{x_k \in x} \texttt{smear}(x_k, f_j)}.$$

Finalement, notre stratégie de bissection `SmearSumRel` choisit la variable x_i de x avec le plus grand impact :

$$\texttt{smearSumRel}(x_i) = \sum_{f_j} \texttt{smearRel}(x_i, f_j).$$

Même si elle n'est pas toujours la meilleure, cette stratégie apparaît plus robuste que ses concurrentes sur l'ensemble des problèmes testés.

2.2 La procédure Contract&Bound

L'algorithme principal `Contract&Bound` (cf. algorithme 1) est appelé à chaque nœud de notre B&B. La première ligne introduit dans le système courant le meilleur coût ub trouvé (comme dans tout `B&B`). La procédure `OuterContractLB` contracte les domaines et améliore le minorant. `InnerExtractUB` extrait une région intérieure dans $[x]$, prélève en cas de succès un point x dans cette région et, si x améliore le critère, remplace x_{ub} par x et le coût ub par $f(x)$.

Algorithm 1 `Contract&Bound` (**in** S, $[x]$; **in-out**: ub)

$\overline{y} \leftarrow ub - \epsilon_{obj}$
`OuterContractLB` (S, $[x] \times [y]$) /* contraction */
if $[x] \times [y] = \emptyset$ **then** exit **endif** /* no solution */
`InnerExtractUB` (S, $[x]$, ub, x_{ub}) /* inner regions */

`OuterContractLB` appelle principalement deux procédures. La première est l'algorithme `Mohc` [1] qui contracte la boîte $[x]\times[y]$ en donnant, par effet de bord, un minorant à la fonction objectif. Cet algorithme récent de propagation de contraintes sur intervalles exploite la monotonie des fonctions. Il utilise une procédure *Revise* efficace qui contracte de manière optimale la boîte par rapport à une contrainte (par exemple, $g_j(x) \leq 0$) si $g_j(x)$ est détectée comme étant monotone suivant toutes les variables et en tout point de la boîte, même si $g_j(x)$ contient des occurrences multiples de variables. On notera que plus la boîte est petite (ou de façon équivalente, plus on descend en profondeur dans l'arbre de recherche), plus les fonctions deviennent monotones.

La seconde procédure appelée par `OuterContractLB` est une relaxation linéaire, nommée ici `OuterLinearization`, qui calcule un minorant de la fonction objectif et met à jour $\underline{y}$.

2.3 Relaxation linéaire extérieure

La relaxation linéaire ci-dessous est une adaptation directe de la forme de Taylor du premier ordre sur intervalles [12]. Soit une fonction $f : \mathbb{R}^n \to \mathbb{R}$ définie sur une boîte $[x]$. Pour toute variable $x_i \in x$, soit $[a_i]$: $\left[\frac{\partial f}{\partial x_i}\right]_N([x])$. Le principe consiste à minorer $f(x)$ par des fonctions linéaires :

$$\forall x \in [x], f(\underline{x}) + \underline{a_1} * x_1^l + ... + \underline{a_n} * x_n^l \ \leq f(x) \quad (1)$$

$$\forall x \in [x], f(\overline{x}) + \overline{a_1} * x_1^r + ... + \overline{a_n} * x_n^r \ \leq f(x) \quad (2)$$

avec : $x_i^l = x_i - \underline{x_i}$ et $x_i^r = x_i - \overline{x_i}$.

La forme de Taylor au premier ordre sur intervalles peut s'appliquer avec n'importe quel point $\dot{x}$ à l'intérieur de la boîte. Au lieu du milieu, choisi habituellement, nous prenons ici un *coin* de la boîte : $\underline{x}$ dans la formule (1) ou $\overline{x}$ dans la formule (2). Si nous considérons une inégalité $g_j(x) \leq 0$, l'expression (1) (ou (2)) définit ainsi un hyperplan $g_j^l(x)$ bornant inférieurement l'espace solution : $g_j^l(x) \leq g_j(x) \leq 0$. En appliquant, par exemple, la formule (1) à la fonction objectif $f(x)$ et aux inégalités $g_j(x) \leq 0$ ($j = 1...m$), on peut générer un problème linéaire LP^{lb} qui est une relaxation du problème initial :

$$\begin{array}{lll} LP^{lb} = \min & & f(\underline{x}) + \underline{a_1} * x_1^l + ... + \underline{a_n} * x_n^l \\ \textit{sous} : & \forall j & g_j(\underline{x}) + \underline{a_1^j} * x_1^l + ... + \underline{a_n^j} * x_n^l \leq 0 \\ & \forall i & 0 \leq x_i^l, \quad x_i^l \leq w([x_i]) \\ \textit{avec} : & & x_i^l = x_i - \underline{x_i} \end{array}$$

`OuterLinearization` invoque ensuite l'algorithme du simplexe pour résoudre LP^{lb}. Il calcule la valeur optimale y^l ou détecte une infaisabilité. L'infaisabilité indique que $[x]$ ne contient pas de solution et peut être éliminée. Si au contraire $y^l \geq \underline{y}$, alors le minorant de l'objectif sur la boîte est mis à jour : $\underline{y} \leftarrow y^l$.

Proposition 1 *Les linéarisations par intervalles (1) et (2) sont correctes et fiables, c.-à-d., elles peuvent être rendues robustes par rapport aux erreurs d'arrondis sur les nombres à virgule flottante.*

La fiabilité est assurée par la taylorisation sur intervalles [14]. La correction de la relation (1) repose sur le choix d'un coin comme point d'expansion. Elle tient au fait que toute variable x_i^l est positive puisque son domaine est $[0, d_i]$, avec $d_i = w([x_i]) = \overline{x_i} - \underline{x_i}$. Ainsi, le minimum de chaque terme $[a_i] * x_i^l$ pour un point $x_i^l \in [0, d_i]$ est obtenu avec $\underline{a_i}$. Symétriquement, la relation (2) est correcte puisque $x_i^r \in [-d_i, 0] \leq 0$, et le minimum de chaque terme est obtenu avec $\overline{a_i}$ [12].

Il faut noter que, bien que nos linéarisations soient fiables, les erreurs de calcul dus aux nombres flottants dans l'algorithme du simplexe peuvent rendre ses résultats non fiables. Pour rendre la solution du simplexe fiable, nous avons ajouté un post-traitement peu coûteux, proposé dans [15], utilisant l'arithmétique d'intervalles.

Nous avons apporté une amélioration à notre relaxation linéaire pour calculer un polytope plus petit. Nous minorons une fonction $f(x)$ avec les expressions (1) et (2) simultanément, en utilisant une forme développée :

1. $f(\underline{x}) + \sum_i \underline{a_i}(x_i - \underline{x_i}) = f(\underline{x}) + \sum_i(\underline{a_i}x_i - \underline{a_i}\,\underline{x_i})$
   $= \sum_i \underline{a_i}x_i + f(\underline{x}) - \sum_i \underline{a_i}\,\underline{x_i}$
2. $f(\overline{x}) + \sum_i \overline{a_i}(x_i - \overline{x_i}) = f(\overline{x}) + \sum_i(\overline{a_i}x_i - \overline{a_i}\,\overline{x_i})$
   $= \sum_i \overline{a_i}x_i + f(\overline{x}) - \sum_i \overline{a_i}\,\overline{x_i}$

2.4 Trouver des majorants dans des régions intérieures

L'appel à `OuterContractLB` est suivi par un appel à `InnerExtractUB` (cf. algorithme 2). La procédure commence par appeler une adaptation d'un algorithme récent, nommé ici `InHC4` [4], pour extraire une boîte intérieure à partir de la boîte extérieure $[x]^{out}$.[4] Appliqué à une contrainte, `InHC4` renvoie une boîte intérieure pour cette contrainte. Les différentes boîtes retournées pour les différentes contraintes sont intersectées pour obtenir une boîte intérieure. Comme `HC4` [3], l'algorithme raisonne sur l'arbre syntaxique des expressions et utilise des *projections* pour les opérateurs unaires, avec arrondi vers l'intérieur. De plus, dans le cas d'unions disjointes d'intervalles (par exemple, pour les opérateurs x^2 et *sinus*), on ne garde qu'un seul intervalle puisque les trous contiennent des points incohérents, ce qui rend l'algorithme heuristique. Pour les opérateurs binaires, les projections dans la phase de rétro-propagation (*projection*) sont différentes et conduisent aussi à des choix heuristiques. Pour plus de détails, se référer à la section 3 de [4].

Si `InHC4` trouve une boîte intérieure $[x]^{in}$, alors `MonotonicityAnalysis` analyse la monotonie de la fonction objectif par rapport à chaque variable x_i. Si la dérivée partielle $[a_i] = \left[\frac{\partial f}{\partial x_i}\right]_N([x]^{in}) \geq 0$, alors f est croissante et $[x_i]$ est remplacé par l'intervalle dégénéré $[\underline{x_i}, \underline{x_i}]$ dans $[x]^{in}$ pour minimiser $f(x)$ dans $[x]^{in}$. Si $[a_i] \leq 0$, f est décroissante et $[x_i]$ est remplacé par $[\overline{x_i}, \overline{x_i}]$ dans $[x]^{in}$.

Algorithm 2 `InnerExtractUB` (**in :** S, $[x]^{out}$; **inout :** ub, x_{ub})

```
[x]^in ← InHC4 (S, [x]^out)   /* Inner box extraction */
if [x]^in ≠ ∅ then
  [x]^in ← MonotonicityAnalysis (f, [x]^in)
  x ← RandomProbing([x]^in)
else
  x ← RandomProbing([x]^out)
end if
cost ← ⌈[f]_N([x,x])⌉   /* Cost evaluation */
if cost < ub and ([x]^in ≠ ∅ or ⌈[g]_N([x,x])⌉ ≤ 0)
then
  ub ← cost ; x_ub ← x
end if
LP^ub ← InnerLinearization (S, [x]^out)
x^l ← Simplex(LP^ub)
if x^l ≠ ⊥ then
  cost ← ⌈[f]_N([x^l, x^l])⌉
  if cost < ub then ub ← cost ; x_ub ← x^l end if
end if
```

Ensuite, nous prenons un point aléatoirement dans la boîte[5] et remplaçons x_{ub} par x si x satisfait les contraintes et améliore le meilleur coût ub. Deux cas différents peuvent se produire. Si `InHC4` a extrait une boîte intérieure $[x]^{in}$, on prend alors un point dans $[x]^{in}$ sans avoir besoin de tester la faisabilité puisque $[x]^{in}$ ne contient que des points faisables. Si aucune boîte intérieure n'a été trouvée, un point est choisi au hasard dans la boîte extérieure et les contraintes doivent être vérifiées. Le remplacement de ce simple tirage aléatoire par une descente de gradient n'apporte pas d'amélioration en pratique. Cela s'explique facilement en présence d'équations puisque les boîtes intérieures sont très petites. C'est en revanche plus surprenant pour les problèmes d'optimisation ne contenant que des contraintes d'inégalité.

La dernière étape de `InnerExtractUB` consiste à linéariser le système pour en extraire cette fois une région polyhédrale intérieure.

2.5 Linéarisation intérieure par intervalles

De manière symétrique à la relation (1) utilisée dans la linéarisation extérieure, on a pour la linéarisation

[4] L'algorithme publié traite en fait un problème dual de recherche de boîtes infaisables, c.-à-d. des boîtes où tous les points satisfont la négation d'au moins une contrainte.

[5] Sélectionner plusieurs points au lieu d'un seul s'est avéré contre-productif dans nos expérimentations.

intérieure :

$$\forall x \in [x],\ f(x) \leq f^l(x) = f(\underline{x}) + \sum_i \overline{a_i} * (x_i - \underline{x_i}). \quad (3)$$

Si on traite une inégalité $f(x) \leq 0$, la relation (3) permet de construire un hyperplan $f^l(x)$ tel que $f(x) \leq f^l(x) \leq 0$. Cette fonction linéaire $f^l(x)$ peut donc être utilisée pour décrire une région intérieure de $[x]$. En appliquant cette idée à la fonction objectif $f(x)$ et aux inégalités $g_j(x) \leq 0$, on peut en déduire le programme linéaire LP^{ub} suivant :

$$\begin{array}{lll} LP^{ub} = \min & & f(\underline{x}) + \sum_i \overline{a_i} * (x_i - \underline{x_i}) \\ \textit{sous}: & \forall j & g_j(\underline{x}) + \sum_i \overline{a_i^j} * (x_i - \underline{x_i}) \leq 0 \\ & \forall i & \underline{x_i} \leq x_i \wedge x_i \leq \overline{x_i} \end{array}$$

De nouveau, l'algorithme du simplexe résout LP^{ub} et retourne la solution optimale x^l ou lève une infaisabilité (cf. algorithme 2). L'infaisabilité cette fois ne prouve rien car le système linéarisé est plus contraint que le système original. Si l'algorithme du simplexe retourne une solution optimale de l'approximation intérieure, alors x^l est aussi une solution du système original, mais généralement pas la solution optimale. On évalue donc la fonction objectif (originale) au point x^l, celle-ci devant être inférieure à ub pour pouvoir mettre à jour x_{ub} et ub.

3 Expérimentations

Nous avons implanté notre stratégie dans le logiciel libre `Ibex` (Interval-Based EXplorer) [5]. Cette bibliothèque a facilité l'implantation de notre optimiseur global en fournissant un certain nombre de briques telles que l'algorithme `Mohc`, diverses stratégies de branchement, la dérivation automatique, etc. Tous les paramètres ont été fixés à un ensemble de valeurs communes à toutes les instances testées. La précision a été fixée à $\epsilon_{obj} =$ `1.e-8` et $\epsilon_{sol} = \frac{\epsilon_{obj}}{10}$. Enfin, l'erreur admissible ϵ_{eq} sur les équations épaisses $h(x) \in [-\epsilon_{eq}, +\epsilon_{eq}]$ a été fixée à $\epsilon_{eq} =$`1.e-8` dans toutes les expérimentations.

Les expérimentations ont été réalisées sur l'ensemble des 74 systèmes de la base de problèmes Coconut[6] sélectionnés par notre meilleur compétiteur fiable `IBBA+` [16]. Le tableau 1 présente une étude qualitative analysant quels sont les ingrédients qui améliorent la performance.

On peut faire quelques observations intéressantes. Tout d'abord, les cinq ingrédients originaux intégrés à notre stratégie s'avèrent tous utiles en pratique. En particulier, la linéarisation extérieure simple que nous avons proposée est souvent cruciale pour la phase de calcul d'un minorant. Des travaux futurs devront comparer cette taylorisation convexe sur intervalles avec l'arithmétique affine et l'opérateur de RLT `Quad` utilisé dans `Icos`. Enfin, l'extraction de régions intérieures est aussi très utile dans la phase de recherche d'un majorant faisable. Le tableau 1 souligne qu'il suffit souvent de réaliser cette extraction de boîte intérieure avec `InHC4` *ou* `InnerLinearization`, mais que leur introduction conjointe est parfois bénéfique et jamais contre-productive sur l'ensemble des problèmes testés.

TAB. 1 – **Étude qualitative.** Les colonnes indiquent le nombre de systèmes dont la perte en performance $\frac{temps\,CPU(stratégie \backslash \{ingrédient\})}{temps\,CPU(stratégie)}$, causée par le retrait d'un seul ingrédient de notre stratégie, appartient à un intervalle donné (première ligne). Les retraits testés sont : `Mohc` remplacé par `HC4` (Mohc/HC4) ; `OuterLinearization` (OuterLinear.) ; `InnerExtractUB` remplacé par un simple tirage aléatoire dans la boîte extérieure (Inner/Probing) ; `InnerLinearization` (InnerLinear.) ; `InHC4` ; `SmearSumRel` remplacé resp. par `SmearMax` (SSR/SM) ; `Round Robin` (SSR/RR) ; `LF` (SSR/LF) ; l'heuristique `LargestFirst` choisit la variable avec l'intervalle le plus large.

Gain	0.02	[0.1, 0.5]	[0.5, 2]	[2, 10]	[10, 100]	>100
Mohc/HC4	0	1	62	5	0	2
OuterLinear.	0	1	35	9	5	20
Inner/Probing	0	0	33	24	9	4
InnerLinear.	0	0	62	7	0	1
InHC4	0	0	66	4	0	0
SSR/SM	0	2	59	4	1	4
SSR/RR	0	1	42	13	11	3
SSR/LF	1	0	40	9	16	4

Nous avons aussi comparé notre stratégie avec des résolveurs d'optimisation globale fiables et disponibles, `Globsol`, `Icos` et `IBBA+`[7], ainsi qu'avec le résolveur complet mais *non fiable* `Baron`. Remarquons que `Baron` ne garantit pas la solution renvoyée qui peut parfois être non faisable et avoir un coût trop bas.

La figure 1 montre les profils de performance de `IbexOpt`, `Baron` et de notre meilleur concurrent fiable `IBBA+`.

Nous donnons également des résultats détaillés sur les 25 systèmes qui sont résolus par `IbexOpt` en plus d'une seconde. Le tableau 2 correspond aux 12 systèmes résolus par `IbexOpt` entre 1 et 10 secondes. Le tableau 3 contient 13 systèmes résolus en plus de 10 secondes. Trois systèmes (`ex6_2_5`, `ex6_2_7` et `ex7_2_3`) ne sont résolus par aucun résolveur, y compris `Baron`.

[6] `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/`

[7] `IBBA+` correspond à la stratégie la plus efficace dans [16].

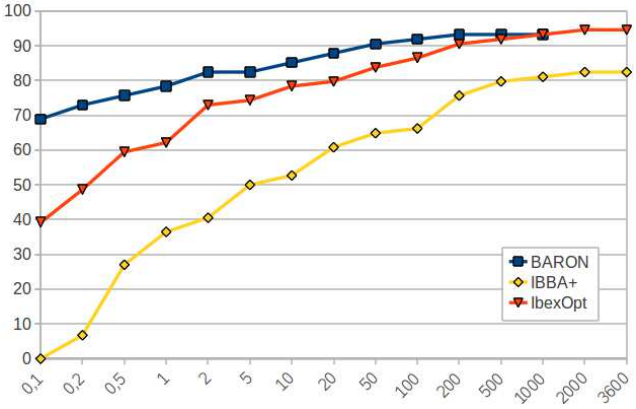

FIG. 1 – **Profils de performance.** Pour un algorithme donné, un point (t, p) sur la courbe correspondante indique le pourcentage p de systèmes résolus en moins de t secondes.

Les résultats pour `Globsol`, `IBBA+`, `Icos` et `IbexOpt` ont été obtenus sur des processeurs similaires (`Intel X86`, `3Ghz`). `Baron` 9.0.7 a été lancé sur le serveur `Neos` (cf. `www.neos-server.org/neos/`), également un processeur `X86`, ce qui rend la comparaison relativement équitable.

Les profils de performance et les tableaux montrent que `IbexOpt` dépasse souvent ses concurrents fiables d'un ou plusieurs ordres de grandeur. Les profils illustrent en particulier le fait que `IbexOpt` a des performances intermédiaires entre `IBBA+` et `Baron` et qu'il peut résoudre les mêmes systèmes que `Baron` en 1000 secondes (540 s en fait). Les résultats obtenus par `Baron` sont impressionnants, même si l'on peut noter que plusieurs instances sont résolues durant un prétraitement (le nombre de nœuds est 1).

On notera aussi que `IbexOpt` est meilleur que `Baron` sur 6 des 25 systèmes difficiles (voir tableaux 2 et 3), spécialement sur la série `ex6_2_*` de problèmes qui ont des fonctions objectifs hautement non polynomiales. A notre connaissance, aucun autre résolveur fiable n'est compétitif avec `Baron` sur des instances non triviales que `Baron` résout en plus d'une seconde.

Nous avons aussi testé une variante de notre stratégie où `Mohc` est remplacé par `3BCID(Mohc)` [20]. Bien que généralement contre-productive en termes de performances, cette variante est plus robuste et peut résoudre l'instance `ex7_2_3` en 38 secondes et 6235 branchements, tandis que `Baron` explose en mémoire.

4 Conclusion

Nous avons proposé un nouveau cadre pour l'optimisation globale fiable qui exploite des régions intérieures dans la phase de recherche de majorant faisable, évitant ainsi le recours à la recherche locale. En définissant les équations avec une petite erreur admissible, cette approche permet aussi de traiter les contraintes d'égalité. Notre stratégie comprend cinq ingrédients utiles. Trois d'entre eux, `Mohc`, `InHC4` et `OuterLinearization` n'avaient jamais été utilisés en optimisation globale. Deux d'entre eux, `SmearSumRel` et `InnerLinearization` sont nouveaux. Tous les cinq ont montré leur efficacité sur un ensemble de problèmes d'optimisation globale non triviaux. Ils confirment la pertinence de l'exploitation des régions intérieures et des approximations polyhédrales basées sur une taylorisation convexe sur intervalles.

Ce nombre de nouveaux ingrédients laisse un espace significatif à des améliorations futures avec l'espoir d'atteindre à long terme les performances de `Baron`.

TAB. 2 – **Comparaison sur les systèmes de difficulté moyenne.** Les deux premières lignes indiquent le nom de chaque compétiteur avec la précision ϵ_{obj} sur le coût. Chaque entrée contient généralement le temps CPU en secondes (première ligne d'une multi-ligne) et le nombre de branchements (deuxième ligne). Une limite de temps de 1 heure (>3600) est commune à `IBBA+`, `GlobSol` et `IbexOpt`. Elle est de 10 mn (>600) pour `Icos`, 1000 secondes pour `Baron` (imposée par le serveur `Neos`). Une case vide indique que l'information n'est pas disponible. En particulier, `GlobSol` s'est restreint à des problèmes ayant moins de 9 variables.

Système ϵ_{obj}	n	Baron 1.e-8	GlobSol 1.e-8	IBBA+ 1.e-8	Icos 1.e-3	IbexOpt 1.e-3	IbexOpt 1.e-8
ex2_1_7	20	0.33 89		16.75 1574	>600	5.52 2102	6.24 2320
ex2_1_8	24	0.07 7		26.78 1916	>600	5.78 1540	6.50 1702
ex3_1_1	8	0.51 453	>3600	116 131195	180 8930	0.48 605	1.31 1516
ex6_1_4	6	0.25 242	14	2.70 1622	4.28 1109	0.37 471	1.11 1053
ex6_2_14	4	**5.2** 1824	32	208 95170	>600	0.77 765	**1.59** 1237
ex7_2_1	7	0.05 1		24.72 8419	>600	0.80 825	1.17 1197
ex7_2_6	3	0.06 7	1	1.23 1319	2.68 986	0.02 73	5.35 16171
ex7_3_4	12	0.93 268		>3600	>600	1.27 771	1.31 775
ex14_2_1	5	0.03 1	4	36.73 16786	>600	0.82 533	1.09 704
ex14_2_3	6	0.03 1	11	173 46673	>600	2.57 996	2.92 1048
ex14_2_4	5	0.03 1		127 30002	>600	0.95 435	1.02 449
ex14_2_6	5	0.03 1		237 74630	>600	1.20 498	1.29 515

TAB. 3 – **Comparaison sur les systèmes difficiles.** En cas de limite de temps atteinte par `Baron` ou `IbexOpt`, la seconde ligne indique la précision obtenue.

Système ϵ_{obj}	n	Baron 1.e-8	GlobSol 1.e-8	IBBA+ 1.e-8	Icos 1.e-3	IbexOpt 1.e-3	IbexOpt 1.e-8
ex2_1_9	10	1.52 2050		154 60007	59.9 1549	13 13370	30 30444
ex6_1_1	8	7.64 5616	3203	>3600	>600	13 12811	17 14725
ex6_1_3	12	19.2 11217		>3600	>600	46.74 26137	540 204439
ex6_2_6	3	26 26765	306	1575 922664	>600	36.75 34318	173 163227
ex6_2_8	3	19 29469	220	458 265276	>600	29.40 27513	111 97554
ex6_2_9	4	**170** 92143	465	522 203775	>600	12.94 9873	**37** 27461
ex6_2_10	6	**>1000** **2.e-3**	>3600	>3600	>600	**431** 224484	1955 820902
ex6_2_11	3	**55** 45085	273	140 83457	>600	4.02 4487	**22** 24264
ex6_2_12	4	30 19182	193	113 58231	>600	4.37 4173	122 86722
ex6_2_13	6	**>1000** **2.e-2**	>3600	>3600	>600	**1099** 545676	>3600 2.e-4
ex7_3_5	13	1.11 309		>3600	136 3699	50.50 40936	55 44147
ex14_1_7	10	1.27 181		>3600	>600	451 177464	464 181136
ex14_2_7	6	0.03 1		>3600	>600	84.73 17463	85 16759

Références

[1] I. Araya, G. Trombettoni, and B. Neveu. Exploiting Monotonicity in Interval Constraint Propagation. In *Proc. AAAI*, pages 9–14, 2010.

[2] F. Benhamou and F. Goualard. Universally Quantified Interval Constraints. In *Proc. CP*, pages 67–82, 2000.

[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 and N. Beldiceanu. Sweeping with Continuous Domains. In *Proc. CP, LNCS 6308*, pages 137–151, 2010.

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

[6] H. Collavizza, F. Delobel, and M. Rueher. Extending Consistent Domains of NCSP. In *IJCAI*, pages 406–413, 1999.

[7] E. R. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker Inc., 2003.

[8] R. Horst and H. Tuy. *Global Optimization : Deterministic Approaches.* Springer, 1966.

[9] R. B. Kearfott. *Rigourous Global Search : Continuous Problems.* Kluwer Academic Publishers, 1996.

[10] R.B. Kearfott and M. Novoa III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2) :152–157, 1990.

[11] Y. Lebbah, C. Michel, and M. Rueher. An Efficient and Safe Framework for Solving Optimization Problems. *J. of Computational and Applied Mathematics*, 199 :372–377, 2007.

[12] Y. Lin and M. Stadtherr. LP Strategy for the Interval-Newton Method in Deterministic Global Optimization. *Ind. & eng. chemistry research*, 43 :3741–3749, 2004.

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

[14] A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge Univ. Press, 1990.

[15] A. Neumaier and O. Shcherbina. Safe Bounds in Linear and Mixed-Integer Programming. *Mathematical Programming*, 99 :283–296, 2004.

[16] J. Ninin, F. Messine, and P. Hansen. A Reliable Affine Relaxation Method for Global Optimization. *Mathematical Programming, accepted for publication*, 2011.

[17] J.-M. Normand, A. Goldsztejn, M. Christie, and F. Benhamou. A Branch and Bound Algorithm for Numerical Max-CSP. *Constraints*, 15(2) :213–237, 2010.

[18] H. Sherali and W. Adams. *Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishers, 1999.

[19] M. Tawarmalani and N. V. Sahinidis. A Polyhedral Branch-and-Cut Approach to Global Optimization. *Mathematical Programming*, 103(2) :225–249, 2005.

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

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