TopSearch : Une Nouvelle Stratégie de Recherche pour les CSPs Continus

Heikel Batnini Michel Rueher Claude Michel

Université de Nice Sophia-Antipolis
Projet COPRIN I3S-CNRS/INRIA/CERMICS
INRIA, 2004 Route des Lucioles
BP 93, 06902 Sophia-Antipolis
hbatnini@sophia.inria.fr rueher@essi.fr cpjm@essi.fr

Abstract

Les méthodes classiques de résolution de CSPs continus sont basées sur un algorithme de recherche combinant bissections et filtrages. En général, le filtrage est basé sur la Hull-consistance ou la Box-consistance. Certains de ces algorithmes de filtrage identifient souvent des trous dans les domaines, c'est-à-dire des portions d'intervalles où certaines contraintes ne sont pas satisfaites. Ces trous sont cependant utilisés uniquement pour déterminer le plus petit intervalle englobant les solutions.

Ce papier présente une stratégie de recherche, nommée `TopSearch` (Topological Search), qui exploite les trous identifiés par ces filtrages. `TopSearch` utilise ces informations pour sélectionner le domaine à couper ainsi que pour définir le point de coupe dans le domaine sélectionné. Les premiers résultats expérimentaux montrent que cette heuristique de recherche dont le sur-coût est minime, peut améliorer de manière très significative les performances des techniques classiques de recherche basées sur la bissection.

1 Introduction

Cet article présente une nouvelle stratégie de recherche pour la résolution de CSPs (Constraint Satisfaction Problem) numériques, résolution dont l'objectif est de trouver une approximation fine des solutions de ces systèmes de contraintes non-linéaires.

Les méthodes classiques pour résoudre les CSPs numériques sont basées sur un algorithme de recherche combinant bissections et filtrages par consistance locale. L'algorithme de recherche sélectionne une variable, ou direction de coupe, puis divise l'intervalle associé en deux parties égales (bissection). Les sous-problèmes correspondants sont résolus de manière indépendante.

Parmi les stratégies de sélection de domaine, la méthode considérée comme étant la plus efficace en moyenne [16] est "Round Robin" (`RR`), qui traite les domaines des variables à tour de rôle. D'autres stratégies de sélection utilisent des informations syntaxiques ou sémantiques. La stratégie Largest First (`LF`), aussi appelée *geometric splitting* [17], consiste à choisir la variable dont le domaine est le plus grand. La stratégie Maximal Smear (`MS`) [8, 12] sélectionne le domaine qui maximise la *smear function*[1]. De manière informelle, `MS` identifie la variable pour laquelle la pente de la fonction associée à une des contraintes est maximale.

En général, les techniques de filtrage utilisées sont basées sur la Hull-consistance [14, 2] ou la Box-consistance [3, 19]. HC4 [2], un des algorithmes les plus efficaces pour calculer un filtrage par Hull-consistance, identifie souvent des

[1] La *smear function* de x_k est :

$$s_k = \max_{1 \le j \le m} \{\max\{|\underline{J}_{i,j}|, |\overline{J}_{i,j}|\} w(\mathbf{x}_i)\},$$

où $\mathbf{J}_{i,j} = [\underline{J}_{i,j}, \overline{J}_{i,j}]$ est la (i,j)-ème entrée de l'extension aux intervalles de la matrice Jacobienne du système.

trous dans les domaines, c'est-à-dire des portions d'intervalles où certaines contraintes ne sont pas satisfaites. Ces trous sont cependant utilisés uniquement pour déterminer le plus petit intervalle englobant les solutions.

Ce papier présente une stratégie de recherche, nommée `TopSearch` (Topological Search), qui exploite les trous identifiés par ce filtrage. `TopSearch` utilise ces informations pour sélectionner le domaine à diviser ainsi que pour définir le point de coupe dans ce domaine. Si aucun trou n'est identifié, une stratégie standard de séléction de domaine et une bissection sont mises en oeuvre. `TopSearch` récupère ces informations avec un sur-coût négligeable puis les exploite durant la phase de recherche. Plusieurs heuristiques exploitant ces informations peuvent être utilisées.

Un retour-arrière chronologique est, la plupart du temps, utilisé pour traiter les sous-CSPs générés par la bissection. D'autres stratégies plus sophistiquées peuvent cependant être utilisées, par exemple une stratégie de retour-arrière dynamique telle que celle qui a été présentée par [11]. `TopSearch` peut être combinée avec n'importe quelle stratégie de retour-arrière.

Notons que des consistances plus fortes basées sur l'utilisation des trous identifiés ont été proposées par [9, 6]. L'originalité de `TopSearch` réside dans l'exploitation de ces trous pour la recherche de solutions.

Cet article est organisé de la manière suivante : La section 2 rappelle les notions de base sur les CSPs numériques. Le schéma général de `TopSearch` est décrit dans la section 3. La section 4 présente la technique utilisée pour collecter les trous dans les domaines. Enfin, la section 5 présente les premiers résultats expérimentaux sur des problèmes issus de la littérature des contraintes numériques.

2 Notions de base

Cette section présente quelques rappels sur les CSPs numériques, ainsi que les notations utilisées dans le reste du papier. Le lecteur pourra se reporter à [15, 8, 2, 5, 19] pour plus de détails sur la résolution de CSPs numériques.

2.1 Notations

L'intervalle $\mathbf{x} = [\underline{x}, \overline{x}]$ dénote l'ensemble des réels x tels que $\underline{x} \leq x \leq \overline{x}$. Les variables sont notées x, y, les vecteurs de variables X, Y et les vecteurs d'intervalles - ou *boîtes* - $\mathbf{X}, \mathbf{Y}$. Le milieu $m(\mathbf{x})$ de l'intervalle $\mathbf{x}$ est $(\overline{x} + \underline{x})/2$ et sa taille est le nombre positif $\overline{x} - \underline{x}$. Nous notons $\mathbf{U_x} = \bigcup \mathbf{x}_{(j)}$, une union d'intervalles où $\mathbf{x}$ est le plus petit intervalle englobant tous les sous-intervalles $\mathbf{x}_{(j)}$. Précisons que ces $\mathbf{x}_{(j)}$ sont disjoints et triés par borne inférieure croissante, c'est-à-dire $\overline{x}_{(j)} < \underline{x}_{(j+1)}$. Nous considérons dans ce papier les CSPs numériques définis par un ensemble de contraintes $\mathbf{C} = \{c_1, \ldots, c_m\}$, mettant en jeu le vecteur de variables $X = (x_1, \ldots, x_n)$ dont les valeurs sont restreintes à la boîte $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

2.2 Arithmétique des intervalles [8, 12, 15]

Soit f, une fonction à valeurs réelle et à n inconnues $X = (x_1, \ldots, x_n)$. L'évaluation par intervalles de f pour une boîte donnée $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ est un intervalle $\mathbf{y}$ tel que :

$$\underline{y} \leq f(x_1, \ldots, x_n) \leq \overline{y}, \ \forall x_i \in \mathbf{x}_i, 1 \leq i \leq n$$

En d'autres termes, $\mathbf{y}$ est un intervalle contenant les valeurs de f lorsque les valeurs des inconnues sont restreintes à la boîte $\mathbf{X}$.

La manière la plus simple de calculer cet intervalle est de substituer tous les opérateurs mathématiques, les constantes et les variables par leur extension aux intervalles. De manière similaire, une relation peut être approximée par un intervalle ou par une union d'intervalles.

Par exemple, soit $c : y = x^2$ avec $x \in \mathbf{x} = [-2, 4]$ et $y \in \mathbf{y} = [1, 16]$. L'ensemble des valeurs de x qui vérifient c lorsque $y \in [1, 16]$, peut être approximé par l'intervalle $[-3, 3]$ ou plus précisemment par l'union d'intervalles $[-3, -2] \cup [2, 3]$ (c.f. figure 1).

2.3 Consistances locales

La plupart des solveurs de contraintes sont basés sur la Hull-consistance [14, 2] ou la Box-consistance [19, 3]. La Hull-consistance établit une propriété sur les bornes des domaines. De manière informelle, soit c_j une contrainte n-aire définie par $f_j(x_1, \ldots, x_n) = 0$, c_j est Hull-consistante si pour chacune de ses variables x_i, les bornes $\underline{x}_i$ et $\overline{x}_i$ possèdent un support dans les domaines des autres variables.

Le filtrage des domaines consiste à appliquer itérativement un opérateur de réduction associé à chaque couple (c_j, x_k). Ces opérateurs réduisent les bornes des domaines de x_k en calculant une approximation par intervalle de la

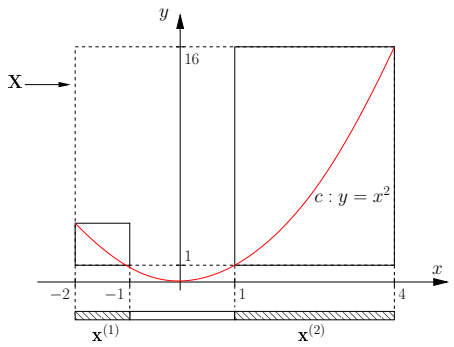

FIG. 1 – Considérons la contrainte $c : y = x^2$ avec $x \in [-2, 4]$ et $y \in [1, 16]$. La projection π_c^x est approximée par l'union $[-2, -1] \cup [1, 4]$ alors que l'approximation calculée par la Hull-consistance est $[-2, 4]$.

fonction de projection $\pi_{c_j}^{x_k}$, qui exprime la variable x_k en fonction des autres variables de c_j. Notons que l'évaluation exacte de $\pi_{c_j}^{x_k}$ n'est en général pas un simple intervalle, mais souvent une union d'intervalles (c.f. figure 1).

L'une des implantations les plus efficaces de la Hull-consistance (*Realpaver* [7]) est basée sur la procédure HC4revise. Cette procédure calcule une réduction des domaines en parcourant une représentation arborescente de la contrainte. La figure 2 illustre l'exécution de HC4revise sur la contrainte $c : y = x^2$ avec $x \in [-2, 4]$ et $y \in [1, 16]$. La propagation ascendante (figure 2(a)) effectue une évaluation par intervalles de l'expression associée à un noeud en utilisant les domaines ou les valeurs associés à ses sous-arbres. La propagation descendante (figure 2(a)) parcours l'arbre dans le sens inverse pour réduire les domaines des variables en utilisant les fonctions de projections associés aux opérateurs.

Par exemple, pour l'évaluation du noeud associé au terme x^2, l'arithmétique des intervalles [8] permet de calculer l'union d'intervalles pour x : $[-2, -1] \cup [1, 4]$. Ce résultat est obtenu en évaluant l'inverse de l'opérateur puissance sur les domaines de x et de y (resp. $[-2, 4]$ et $[1, 16]$). HC4revise identifie ainsi dans le domaine de x l'intervalle ouvert $(-1, 1)$, qui n'a pas de support pour c lorsque y est dans l'intervalle $[1, 16]$.

Actuellement, HC4 identifie les trous lorsque les opérateurs produit ou puissance apparaissent dans une contrainte. Une extension à d'autres opérateurs pourrait toutefois être envisagée.

Le calcul du filtrage par Hull-consistance basé sur HC4revise identifie donc des trous dans les domaines des variables, c'est-à-dire des intervalles sur lesquels plusieurs contraintes ne sont pas satisfaites. Ces trous sont utilisés par le filtrage pour effectuer dans certains cas des réductions plus fines. Toutefois, des algorithmes comme HC4 ne conservent pas ces trous après le filtrage afin d'éviter une gestion coûteuse des unions d'intervalles.

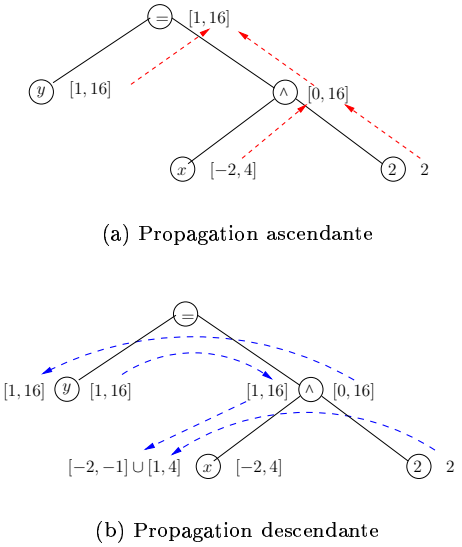

FIG. 2 – L'opérateur de réduction HC4revise.

La section suivante montre comment ces trous peuvent être exploités pour orienter la recherche de solutions.

3 Une stratégie de recherche guidée par les trous

Cette section présente un nouvel algorithme de recherche, nommé `TopSearch` (Topological Search). Contrairement aux stratégies classiques de recherche basées sur la bissection, `TopSearch` s'appuie sur les trous identifiés pendant l'étape de filtrage dans les domaines de chaque variable, c'est-à-dire un ensemble d'intervalles disjoints qui ne contiennent aucune solution du CSP.

Ces trous apportent une information pertinente à la fois pour choisir la direction de coupe

et le point de coupe dans le domaine choisi. `TopSearch` traite en priorité les domaines pour lesquels des trous ont été identifiés. Un trou est alors éliminé du domaine sélectionné ; deux sous-problèmes disjoints sont ainsi créés.

Le schéma général de ce nouvel algorithme de résolution est donné par la figure 3. Le domaine courant est tout d'abord réduit par la procédure `Prune` (ligne 5), qui est un algorithme standard de filtrage qui utilise HC4 pour calculer la Hull-consistance[2].

Si la boîte courante $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ n'est pas une solution du CSP, `TopSearch` collecte les trous dans les domaines $\mathbf{x}_i$ (ligne 10).

$\mathbf{U_X}$ dénote le vecteur d'union d'intervalles $(\mathbf{U}_{\mathbf{x}_1}, \ldots, \mathbf{U}_{\mathbf{x}_n})$ retourné par la procédure `ComputeGaps` (voir section 4). Pour chaque variable x_i, cette procédure calcule une union d'intervalles

$$\mathbf{U}_{\mathbf{x}_i} = \bigcup_j \mathbf{x}_{i(j)},$$

où $\mathbf{x}_{i(j)} = [\underline{x}_{i(j)}, \overline{x}_{i(j)}]$ dénote le j-ième sous-domaine de la variable x_i.

Le p-ième trou de $\mathbf{U}_{\mathbf{x}_i}$ est l'intervalle ouvert entre le p-ième et le $p+1$-ième sous-domaine :

$$\mathbf{g}_{i(p)} = (\overline{x}_{i(p)}, \underline{x}_{i(p+1)})$$

`TopSearch` sélectionne un domaine contenant au moins un trou (ligne 12). Puis, les points de coupe sont déterminés de manière à éliminer un trou $\mathbf{g}_{k(p)}$ (lignes 14 et 15). Différentes heuristiques pour sélectionner la direction de coupe (ligne 12) ainsi que le trou à éliminer (ligne 13) peuvent être utilisées (c.f. section 5).

Si aucun trou n'a été identifié, une stratégie standard de sélection et une bissection sont mises en oeuvre (lignes 17-19). La stratégie de choix de la direction de coupe utilisée est une des heuristiques décrites dans la section 1 (`RR`, `LF`, ou `MS`).

La section suivante présente la procédure `ComputeGaps` qui collecte les trous calculés pendant la phase de filtrage.

4 Calcul des trous

Cette section présente un algorithme pour calculer les trous dans les domaines des variables. `TopSearch` collecte et intersecte les unions d'intervalles qui ont été calculés dans chaque domaine par HC4revise durant la phase de filtrage. Le sur-coût engendré est donc limité à des opérations d'intersections entre ces unions d'intervalles.

Function `TopSearch`(in :$\mathbf{X}_0, \mathbf{C}$ out : S)

```
    %% X_0 = (x_1, ..., x_n), les domaines
    %% C = (c_1, ..., c_n), les contraintes
    %% S, l'ensemble des solutions
1 : Q ← X_0
2 : S ← ∅
3 : while Q ≠ ∅ do
4 :   Extraire X de Q
5 :   X ← Prune(X, C)
6 :   if X ≠ ∅ then
7 :     if w(X) ≤ ω_sol then
          %% X est une solution
8 :       S ← S ∪ X
9 :     else
10 :      U_X ←ComputeGaps(X, C)
11 :      if des trous ont été indentifiés then
12 :       Select U_{x_k} from U_X
13 :       Select g_{k(p)} from U_{x_k}
14 :       x_k^1 ← x̄_{k(p)}
15 :       x_k^2 ← x_{k(p+1)}
16 :     else
          %% Sélection standard du domaine
          %% de coupe
17 :       Select x_k from X
          %% Bissection standard
18 :       x_k^1 ← m(x_k)
19 :       x_k^2 ← m(x_k)
20 :     endif
21 :     Q ← Q ∪ (x_1, ..., [x_k, x_k^1], ..., x_n)
22 :     Q ← Q ∪ (x_1, ..., [x_k^2, x̄_k], ..., x_n)
23 :   endif
24 : endif
25 : endwhile
26 : return S
```

FIG. 3 – L'algorithme `TopSearch`.

En d'autre termes, pour chaque couple (c_j, x_i), HC4revise calcule une union d'intervalles qui approxime la fonction de projection $\pi_{c_j}^{x_i}$. Pour chaque variable x_i, l'algorithme `ComputeGaps` calcule l'intersection de toutes les approximations des fonctions de projections associées aux contraintes mettant en jeu x_i (voir figure 4).

[2] L'étape de filtrage peut combiner plusieurs techniques ; par exemple Hull-consistance et Box-consistance. Toutefois, on ne peut à l'heure actuelle identifier les trous que si HC4 est utilisé.

Plus précisément, soient c_j une contrainte et x_i une variable de c_j et soit $\mathbf{U}_{\mathbf{x}_i}^{(j)}$ l'union d'intervalles associée à $\pi_{c_j}^{x_i}$ et qui a été calculée pendant la phase de filtrage. L'union d'intervalles calculées pour la variable x_i par l'algorithme `ComputeGaps` (voir figure 5) est l'intersection des $\mathbf{U}_{\mathbf{x}_i}^{(j)}$, pour toutes les contraintes c_j dans lesquelles x_i apparaît.

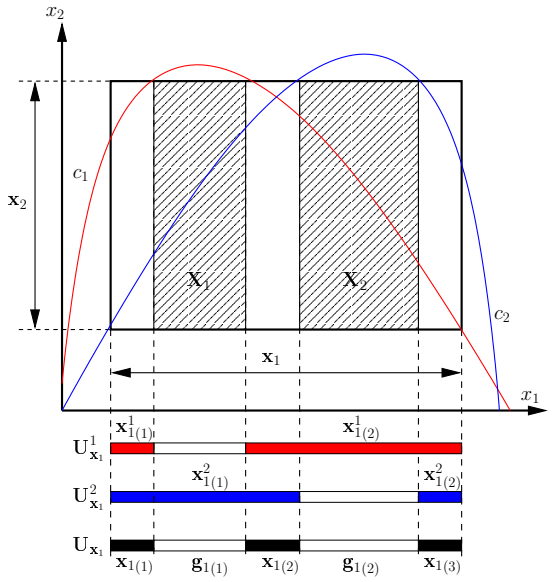

FIG. 4 – Les approximations des projections de c_1 et c_2 sur la variable x_1 sont respectivement $\mathbf{U}^1_{\mathbf{x}_1} = \mathbf{x}^1_{1(1)} \cup \mathbf{x}^1_{1(2)}$ et $\mathbf{U}^2_{\mathbf{x}_1} = \mathbf{x}^2_{1(1)} \cup \mathbf{x}^2_{1(2)}$. L'union d'intervalles générée par `ComputeGaps` pour la variable x_1 est l'intersection de $\mathbf{U}^1_{\mathbf{x}_1}$ et $\mathbf{U}^2_{\mathbf{x}_1}$ soit $\mathbf{U}_{\mathbf{x}_1} = \mathbf{x}_{1(1)} \cup \mathbf{x}_{1(2)} \cup \mathbf{x}_{1(3)}$. Les deux trous identifiés, $\mathbf{g}_{1(1)}$ et $\mathbf{g}_{1(1)}$, permettent d'éliminer les boîtes inconsistantes $\mathbf{X}_1$ et $\mathbf{X}_2$, dans lesquelles une des deux contraintes n'est pas vérifiée.

Le calcul effectif des $\mathbf{U}_{\mathbf{x}_i}^{(j)}$ a été réalisé par une légère modification de la procédure HC4revise. Pendant l'évaluation descendante de l'arbre, lorsque le noeud considéré est une variable x_i et que des trous ont été détectés dans son domaine $\mathbf{x}_i$, l'union d'intervalles associé à x_i est intersecté avec $\mathbf{U}_{\mathbf{x}_i}$.

La section suivante présente une comparaison entre l'algorithme de recherche `TopSearch` et l'algorithme standard de recherche basé sur la bissection sur une série de CSPs numériques.

Function `ComputeGaps`(in :$\mathbf{X}, \mathbf{C}$ out : $\mathbf{U}_\mathbf{X}$)

%% $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\mathbf{U}_\mathbf{X} = (\mathbf{U}_{\mathbf{x}_1}, \ldots, \mathbf{U}_{\mathbf{x}_n})$
1 : $\mathbf{U}_\mathbf{X} \leftarrow \mathbf{X}$
%% Set $\mathbf{U}_{\mathbf{x}_i} = \mathbf{x}_i$ for $1 \leq i \leq n$
2 : **for each** $c_j \in \mathbf{C}$ **do**
3 : **for each** $x_i \in \mathrm{Vars}(c_j)$ **do**
4 : $\mathbf{U}_{\mathbf{x}_i} \leftarrow \mathbf{U}_{\mathbf{x}_i} \cap \mathbf{U}_{\mathbf{x}_i}^{(j)}$
%% $\mathbf{U}_{\mathbf{x}_i}^{(j)}$ is computed by HC4revise
5 : **endfor**
6 : **endfor**
7 : **return** $\mathbf{U}_\mathbf{X}$

FIG. 5 – Calcul des trous

5 Résultats expérimentaux

Cette section présente les résultats obtenus avec `TopSearch` sur une série de problèmes classiques :
- *i1* et *i4* [19] sont deux problèmes issus de l'arithmétique des intervalles.
- *kin1* [19] et *robot1* [4] sont deux applications de la cinématique directe de robots.
- *nbody5* [13] est une instance du problème à 5 corps issu de la mécanique céleste.
- *eco7* et *eco8* [19] sont deux application de modélisation économique.
- *sierpinski*, *ponts* [10], *ext-penta* et *ext-penta2* [1] sont des problèmes de contraintes géométriques, où *ext-penta2* est une instance particulière de *ext-penta*.
- *caprasse* [18] est un système d'équations polynomiales issu des problèmes de Posso.

	n	m	s	$\mathbf{X}$
eco8	8	8	8	$[-10^3, 10^3]$
eco7	7	7	8	$[-10^3, 10^3]$
sierpinski	26	26	2^{25}	$[-10^2, 10^2]$
ext-penta2	18	19	320	$[-10, 10]$
ponts	26	26	128	$[-10^2, 10^2]$
kin1	12	12	16	$[-1, 1]$
ext-penta	19	19	448	$[-10, 10]$
i1	10	10	5	$[-10^3, 10^3]$
nbody5	6	6	12	$[-10^8, 10^8]$
i4	10	10	1024	$[-1, 1]$
robot1	9	9	24	$[-10^8, 10^8]$
caprasse	4	6	28	$[-10, 10]$

TAB. 1 – n est le nombre de variables, m est le nombre de contraintes, s est le nombre solutions dans la boîte initiale $\mathbf{X}$.

La table 1 donne, pour chacun de ces problèmes, le nombre de variables, contraintes et solutions ainsi que les domaines initiaux.

Nous avons comparé un algorithme de recherche basé sur la bissection classique avec `TopSearch`; tous les deux sont combinés avec `RR` et implantées avec avec Realpaver [7]. Le filtrage est réalisé par la Hull-consistance combinée avec la Box-consistance et l'algorithme de Newton par intervalles.

Différentes heuristiques de choix de direction de coupe et de choix de point de coupe ont été étudiées :
- *Absolute Largest Gap* (`ALG`) : Le domaine sélectionné $\mathbf{x}_k$ contient le plus grand trou identifié par `ComputeGaps` :

$$k = \max_{1 \leq i \leq n} w(\mathbf{g}_{i(j)})$$

Le point de coupe choisi correspond au trou le plus grand. Cette heuristique minimise la taille des sous-problèmes générés.
- *Relative Largest Gap* (`RLG`) : Une variation de `ALG` qui consiste à maximiser le rapport entre la taille du plus grand trou et la taille du domaine :

$$k = \max_{1 \leq i \leq n} \frac{w(\mathbf{g}_{i(j)})}{w(\mathbf{x}_i)}$$

Cette heuristique maximise la réduction du domaine sélectionné en fonction de sa taille.
- *Absolute Smallest Gap* (`ASG`) et *Relative Smallest Gap* (`RSG`) sont basées sur la même idée que (`ALG`) et (`RLG`), mis à part que les trous éliminés minimisent les critères respectifs.

Les tables 2 et 3 présentent les résultats obtenus pour `TopSearch` avec l'heuristique `ALG`. Les résultats obtenus avec les autres heuristiques sont très proches (moins de 2% de différence en terme de temps de calcul).

Tous les résultats ont été obtenus sur un PC muni d'un processeur cadencé à 3Ghz et de 512Mo de RAM. Pour chaque problème, toutes les solutions ont été recherchées excepté pour *sierpinski* pour lequel on s'est restreint à trouver la première solution.

Le tableau 2 montre que `TopSearch` permet un gain de performances en termes de temps de calcul. Des améliorations significatives ont notamment été obtenues pour *eco7,eco8,sierpinski* et *ponts*, pour lesquels le temps de calcul est fortement réduit : de 32.5% à 91.7% de réduction. Le tableau 3 montre que le nombre de bissections a également été réduit : de 31% à 78.7% de réduction. Ce nombre de bissections a été réduit pour tous les problèmes présentés excepté pour *i4*. Toutefois pour ce problème, 1023 trous ont été identifiés et éliminés par `TopSearch` sur les 1023 bissections, ce qui a amélioré l'efficacité de processus de filtrage. Les limites de `TopSearch` sont illustrées par *caprasse*, pour lequel aucun trou n'a été identifié.

	`RR`	`TS(ALG)+RR`	Réduction
eco8	679.21	56.38	**-91.7**%
eco7	108.36	24.99	**-76.9**%
sierpinski	3.48	1.02	**-70.7**%
ext-penta2	1.64	0.73	**-55.5**%
ponts	22.10	14.90	**-32.5**%
kin1	0.49	0.4	**-18.3**%
ext-penta	429.34	372.98	**-13.1**%
i1	45.78	40.29	**-12**%
nbody5	87.63	80.01	**-8.7**%
i4	1.41	1.29	**-8.5**%
robot1	46.12	45.55	**-1.2**%
caprasse	0.93	0.93	0%

TAB. 2 – Temps de calcul en secondes

	`RR`	`TS(ALG)+RR`	Réduction
eco8	158341	122322	**-22.7**%
eco7	253915	54028	**-78.7**%
sierpinski	2210	760	**-65.6**%
ext-penta2	1492	1029	**-31**%
ponts	16321	10787	**-33.9**%
kin1	535	431	**-19.4**%
ext-penta	503015	421896	**-16.1**%
i1	170028	150983	**-11.2**%
nbody5	411731	366614	**-11**%
i4	1023	1023	0%
robot1	24787	24635	**-0.6**%
caprasse	979	979	0%

TAB. 3 – Nombre de bissections

6 Conclusion

Nous avons introduit dans ce papier un nouvel algorithme de recherche pour la résolution de CSPs non-linéaires. Cette stratégie de recherche exploite les trous qui sont identifiés durant la phase de filtrage. Ces trous, calculés avec un sur-coût minime, apportent des informations significatives à la fois pour le choix de la direction de coupe que pour le choix du point de

coupe dans le domaine sélectionné. Les résultats expérimentaux montrent que dans le cas où des trous ont été identifiés pendant la recherche de solution, le nombre de bissections ainsi que le temps de résolution peuvent être réduits de manière très significative. Les travaux futurs concernent l'extension de cette technique pour d'autres méthodes de filtrage.

Références

[1] H. Batnini and M. Rueher. Décomposition sémantique pour la résolution de systèmes d'équations de distances. *JEDAI(Journal Electronique d'Intelligence Artificielle)*, 2(1), 2004. Édition spéciale JNPC 2003.

[2] F. Benhamou, F. Goualard, L. Granvilliers, and J.F. Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244, 1999.

[3] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(intervals) revisited. In Maurice Bruynooghe, editor, *Proceedings of the 1994 International Symposium*, pages 124–138. MIT Press, 1994.

[4] D. Bini and B. Mourrain. *Handbook of Polynomial Systems*. November 1996.

[5] H. Collavizza, F. Delobel, and M. Rueher. A note on partial consistencies over continuous domains. In M. Maher and J-F. Puget, editors, *Proc. of CP'98 : 4th International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1520, pages 147–162, Pisa, Italy, November 1998. Springer Verlag.

[6] B. Faltings. Arc-consistency for continuous variables. *Artif. Intell.*, 65(2) :363–376, 1994.

[7] L. Granvilliers. On the combination of interval constraint solvers. *Reliable Computing*, 7(6) :467–483, 2001.

[8] E.R. Hansen. *Global optimization using interval analysis*. Marcel Deckler, 1992.

[9] E. Hyvönen. Constraint reasoning based on interval arithmetic : the tolerance propagation approach. *Artificial Intelligence*, 58(1) :71–112, December 1992.

[10] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A constraint programming approach for solving rigid geometric systems. In *Proc. of CP'00 : Sixth International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1894, pages 233–248, Singapore, September 2000. Springer Verlag.

[11] Narendra Jussien and Olivier Lhomme. Dynamic domain splitting for numeric CSPs. In *European Conference on Artificial Intelligence*, pages 224–228, 1998.

[12] R.B. Kearfott. *Rigorous global search : continuous problems*. Kluwer, 1996.

[13] I. Kotsireas and I. Lazard. Central configurations of the 5-body problem with equal masses in three dimensional space. In *Proceedings of CASC*, 1998.

[14] O. Lhomme. Consistency techniques for numerical csps. In *IJCAI-93*, pages 232–238, 1993.

[15] R. Moore. *Interval analysis*. Prentice-Hall, 1977.

[16] COCONUT Project. Algorithms for solving nonlinear constrained and optimization problems : The state of the art. http ://www.mat.univie.ac.at/~neum/ms/ StArt.ps.gz, 2001.

[17] D. Ratz. Box-splitting strategies for the interval Gauss–Seidel step in a global optimization method. *Computing*, 53 :337–354, 1994.

[18] C. Traverso. The posso test suite examples, 1993. Available at http ://www.inria.fr/saga/POL/index.html.

[19] P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM, Journal of Numerical Analysis*, 34(2) :797–827, April 1997.