CID : Disjonction Constructive sur Intervalles

Gilles Trombettoni **Gilles Chabert**

Université de Nice-Sophia, INRIA Sophia-Antipolis, Projet COPRIN,
INRIA 2004 route des Lucioles, BP 93, 06902 Sophia Antipolis cedex
trombe@sophia.inria.fr, gilchab@gmail.com

Résumé

Le "rognage" (*shaving*) et la disjonction constructive sont deux principes de réfutation utilisés en programmation par contraintes. Le rognage est utilisé pour calculer la propriété de *singleton* arc-cohérence sur les CSP en domaines finis et la 3B-cohérence sur les CSP numériques. Il est également au cœur de l'algorithme SATZ [9] pour montrer la satisfiabilité d'une formule booléenne. Si l'on considère les domaines comme des contraintes unaires disjonctives, on peut adapter la *disjonction constructive*, proposée par Van Hentenryck et al. dans les années 1990, pour produire un opérateur de filtrage du modèle classique (où le problème est vu comme une conjonction de contraintes). Un avantage sur le rognage est que les étapes de (sous-)filtrage exécutées durant la disjonction constructive sont mieux réutilisées.

Pour traiter les CSP numériques (systèmes comportant des contraintes sur les réels et résolus par des techniques d'intervalles), cet article présente deux nouveaux opérateurs de filtrage, appelés `CID` et `3BCID`, basés sur la disjonction constructive, et une nouvelle heuristique de bissection basée sur `CID`. `CID` est exclusivement basé sur la disjonction constructive alors que `3BCID` est un algorithme hybridant rognage et disjonction constructive. Pendant que l'opérateur de filtrage `CID` établit la CID-cohérence, l'heuristique de bissection basée sur `CID` est capable d'apprendre sans surcoût la prochaine variable à bissecter.

Des expérimentations ont été menées sur 20 benchmarks. `CID` and `3BCID` produisent sur plusieurs benchmarks des gains en performance d'un ou plusieurs ordres de grandeur par rapport à une stratégie standard. `CID` se compare avantageusement par rapport à l'opérateur de 3B tout en étant plus simple à implanter. Ces expérimentations suggèrent de fixer dans `3BCID` le paramètre lié à `CID`, offrant ainsi en quelque sorte à l'utilisateur une variante prometteuse de l'opérateur 3B.

1 Introduction

En programmation par contraintes et en recherche opérationnelle, le rognage est basé sur un simple principe de réfutation. Une valeur est temporairement affectée à une variable (les autres valeurs sont temporairement éliminées) et une cohérence partielle est calculée sur le sous-problème correspondant. Si une inconsistance est obtenue, la valeur peut être éliminée du domaine de la variable de manière certaine. Dans le cas contraire, la valeur est maintenue dans le domaine. Ce principe de réfutation a deux défauts. Contrairement à l'arc-cohérence, il n'est pas incrémental [2]. En effet, le travail de l'algorithme de réfutation sous-jacent portant sur *tout* le sous-problème est la raison expliquant pourquoi une *valeur unique* peut être éliminée. C'est pourquoi obtenir *l'arc-cohérence singleton* (SAC) sur les CSP en domaines finis requiert un algorithme de point-fixe coûteux où toutes les variables doivent être traitées à nouveau à chaque fois qu'une seule valeur est éliminée [3]. SAC2 [1], SAC-optim [2] et d'autres variantes SAC obtiennent de meilleures complexités en moyenne ou au pire en gérant de lourdes structures de données ou en dupliquant le CSP pour chaque valeur. Utiliser ces opérateurs de filtrage au sein d'un schéma de retour arrière est loin d'être compétitif avec l'algorithme MAC standard dans l'état actuel de la recherche. Dans son algorithme `QuickShaving` [8], Lhomme utilise ce principe de rognage de manière pragmatique, c'est-à-dire sans surcoût, en apprenant les variables prometteuses durant la recherche. Des chercheurs et des ingénieurs traitant des problèmes d'ordonnancement utilisent aussi le principe de rognage depuis longtemps. Les variables sont traitées une seule fois, évitant ainsi l'obtention d'un point fixe.

Sur les CSP numériques, la *2B-cohérence* est l'algorithme de réfutation utilisé dans la 3B-cohérence [7], comme l'arc-cohérence est utilisée pour réfuter les valeurs dans la propriété SAC. Cette propriété plus faible, établie aux bornes des intervalles seulement, explique que le filtrage par 3B-cohérence produit souvent des gains en performance. Le deuxième défaut du ro-

gnage est que la réduction obtenue par l'opérateur de réfutation est perdue, ce qui n'est pas le cas avec la disjonction constructive[1].

La *disjonction constructive* produit un filtrage significatif quand on traite des disjonctions de contraintes, et non pas seulement des conjonctions comme dans le modèle de CSP standard [18]. L'idée consiste à propager indépendamment chaque terme de la disjonction, puis de calculer l'union des différents espaces de recherche obtenus. En d'autres termes, une valeur qui est éliminée par chaque processus de propagation (déclenché pour chaque terme/contrainte de la disjonction) peut être éliminée de manière certaine du problème original. Cette idée est fructueuse dans différents domaines tels que l'ordonnancement où une contrainte courante exprime que deux tâches ne doivent pas se chevaucher, ou dans des problèmes de placement (*bin/strip packing*) où deux éléments ne doivent pas s'intersecter.

Il est connu, mais pas si répandu, que la disjonction constructive peut aussi s'appliquer au modèle standard des CSP. En effet, chaque domaine de variable peut se voir comme une contrainte unaire disjonctive imposant une seule valeur parmi un ensemble possible ($x = v_1 \vee ... \vee x = v_n$, où x est une variable et $v_1, ..., v_n$ sont les différentes valeurs). Dans ce cas, comme pour le rognage, le principe de disjonction constructive s'applique de la manière suivante. Chaque valeur est affectée itérativement et temporairement à chaque variable du système (les autres valeurs étant temporairement écartées), on calcule une cohérence partielle sur le problème correspondant, puis on calcule l'union des différents espaces de recherche obtenus. Cette disjonction constructive de "domaine" n'est pas très exploitée aujourd'hui alors qu'elle produit parfois des gains en performance impressionnants. En particulier, si l'on ajoute aux contraintes *all-diff* [14] des contraintes de disjonction de domaine dans le fameux jeu du Sudoku (déclenchées quand par exemple il reste deux valeurs seulement dans une case/variable), on obtient souvent une résolution sans retour en arrière. Le même phénomène a été observé avec le rognage, mais à un coût plus important [15].

C'est cette observation qui a précisément motivé les recherches décrites dans cet article qui étudie comment comment appliquer la disjonction constructive de domaine aux CSP numériques. La nature continue des domaines intervalle se prête particulièrement bien à la disjonction constructive de domaine. En découpant un intervalle en plusieurs sous-intervalles plus petits, on peut définir de manière simple la *disjonction constructive d'intervalle* (en anglais : *Constructive Interval Disjunction – CID*) introduite dans cet article.

[1] Notons que des implantations optimisées de SAC réutilisent les domaines obtenus par sous-filtrage lors des appels ultérieurs à "`VarShaving`" [6].

Après les notations et les définitions de la section 2, les sections 3 et 4 décrivent la cohérence partielle *CID* et son opérateur de filtrage correspondant. Un algorithme hybridant rognage et `CID` est présenté à la section 5. La section 6 décrit un nouvelle stratégie de bissection basée sur le filtrage `CID`. Finalement, des expérimentations très encourageantes sont détaillées à la section 7.

2 Définitions

Les algorithmes présentés dans cet article ont pour but de résoudre des systèmes d'équations sur les réels.

Definition 1 *Un* **CSP numérique** *(NCSP)* $P = (X, C, B)$ *contient un ensemble* C *de contraintes et un ensemble* X *de* n *variables. Chaque variable* $x_i \in X$ *peut prendre une valeur réelle dans l'intervalle* $\mathbf{x_i}$ *(la* **boîte** $\mathbf{B = x_1 \times ... \times x_n}$*). Une solution de* P *est une affectation des variables de* P *telle que les contraintes de* C *sont satisfaites.*

Parce que les nombres réels ne peuvent pas être représentés sur un ordinateur (aux ressources finies), les bornes d'un intervalle $\mathbf{x_i}$ sont des nombres flottants.

Le filtrage par CID utilise une opération d'union entre deux boîtes.

Definition 2 *Soit* B_l *et* B_r *deux boîtes correspondant au même ensemble* X *de variables.*

L'opération de **hull** *(box) de* B_l *et* B_r*, noté* `Hull`(B_l, B_r)*, est la boîte minimale incluant* B_l *et* B_r*.*

Pour calculer un point de bissection basé sur une nouvelle heuristique, on aura besoin de connaître la *taille* d'une boîte. Dans cet article, la taille d'une boîte est fournie par son périmètre.

Definition 3 *Soit* $B = \mathbf{x_1} \times ... \times \mathbf{x_n}$ *une boîte. La* **taille** *de* B *est* $\sum_{i=1}^{n}(\overline{\mathbf{x_i}} - \underline{\mathbf{x_i}})$*, où* $\overline{\mathbf{x_i}}$ *et* $\underline{\mathbf{x_i}}$ *sont respectivement les bornes supérieure et inférieure de l'intervalle* $\mathbf{x_i}$*.*

La *2B-cohérence* (ou *hull-consistency*) [7] et la *Box-cohérence* [17] sont une forme d'arc-cohérence restreinte aux bornes des domaines.

3 La CID-cohérence

La CID-cohérence est une nouvelle cohérence partielle pour les CSP numériques. En suivant le principe donné dans l'introduction, la CID(2)-cohérence se définit comme suit :

Definition 4 (CID(2)-cohérence)
Soit $P = (X, C, B)$ *un NCSP. Soit* F *une cohérence partielle.*

Soit B_i^l la sous-boîte de B dans laquelle $\mathbf{x_i}$ est remplacé par $[\underline{\mathbf{x_i}}, \check{\mathbf{x}}_\mathbf{i}]$ (où $\check{\mathbf{x}}_\mathbf{i}$ est le point milieu de l'intervalle $\mathbf{x_i}$). Soit B_i^r la sous-boîte de B dans laquelle $\mathbf{x_i}$ est remplacé par $[\check{\mathbf{x}}_\mathbf{i}, \overline{\mathbf{x_i}}]$.

Une variable x_i de X est CID(2)-cohérente vis à vis de P et F si $B = Hull(F(X, C, B_i^l), F(X, C, B_i^r))$.
Le NCSP P est CID(2)-cohérent si toutes les variables de X sont CID(2)-cohérentes.

Pour toute dimension, le nombre de tranches considéré dans la CID(2)-cohérence est égal à 2. Cette définition se généralise à la CID(s)-cohérence pour laquelle chaque variable est découpée en s tranches par la procédure `VarCID`.

En pratique, comme la 3B-w-cohérence, la CID-cohérence s'obtient avec une précision qui évite une convergence lente vers un point-fixe. Nous considérerons qu'une variable est CID(2,w)-cohérente si l'union (hull) des boîtes gauche et droite résultant du sous-filtrage ne réduit aucune variable de plus de w.

Definition 5 (CID(2,w)-cohérence)
Soit $P = (X, C, B)$ un NCSP et $B' = Hull(F(X, C, B_i^l), F(X, C, B_i^r))$.

Une variable x_i de X est CID(2,w)-cohérente si $\forall i \in [1..n], |\mathbf{x_i}| - |\mathbf{x_i'}| \leq w$, où $|\mathbf{x_i}|$ est la taille de $\mathbf{x_i}$ dans B et $|\mathbf{x_i'}|$ est la taille de $\mathbf{x_i}$ dans B'.
Le NCSP P est CID(2,w)-cohérent si toutes les variables de X sont CID(2,w)-cohérentes.

L'algorithme `CID` détaille le filtrage par CID(s,w)-cohérence. Comme l'algorithme de 3B-cohérence, `CID` itère sur toutes les variables jusqu'à ce qu'un critère d'arrêt soit rempli.

Chaque variable x_i est traitée par la procédure `VarCID` (x_i est dite "*varcidée*"). L'intervalle de x_i est découpé en s tranches de taille $\frac{|\mathbf{x_i}|}{s}$ chacune par la procédure `SubBox`. L'opérateur de cohérence partielle F (ex : 2B, Box-cohérence) filtre les sous-boîtes correspondantes, et l'union des boîtes résultantes est calculée par l'opérateur `Hull`.

Notons que si l'opérateur de sous-filtrage F appliqué à une sous-boîte donnée détecte une inconsistance, alors `sliceBox'` est vide. Il n'est alors pas utile de faire l'union de `sliceBox'` avec la boîte courante.

Le critère d'arrêt lié à w est donné à la définition 5 : la boucle `Repeat` est interrompue quand aucun intervalle n'est réduit de plus de w. Ce critère d'arrêt ne garantit pas la convergence vers un point fixe *unique*. En effet, les spécialistes des solveurs de contraintes sur intervalles savent bien que, quand une précision w est utilisée, la boîte finale dépend de l'ordre d'application des opérations de filtrage. De plus, d'un point de vue théorique, pour que le critère d'arrêt entraîne l'obtention d'un point fixe (non unique), il est nécessaire de retourner la boîte obtenue juste avant le dernier filtrage, c'est-à-dire la boîte P_{old} dans l'algorithme `CID`.

```
algorithm CID (s : number of slices, w : precision, in-out P = (X, C, B) : an NCSP, F : subfiltering operator and its parameters)
    repeat
        P_old ← P
        LoopCID (X, s, P, F)
    until StopCriterion(w, P, P_old)
end.
procedure LoopCID (X, s, in-out P, F)
    for every variable xi ∈ X do
        VarCID (xi, s, P, F)
    end
end.
procedure VarCID (xi, s, (X, C, in-out B), F)
    B' ← empty box
    for j ← 1 to s do
        sliceBox ← SubBox (j, s, xi, B) /* the j^th sub-box of B on xi */
        sliceBox' ← F(X, C, sliceBox) /* perform a partial consistency */
        B' ← Hull(B', sliceBox') /* Union with previous sub-boxes */
    end
    B ← B'
end.
```

k-CID cohérence

De même que la k-B-cohérence [7] généralise la 3-B-cohérence, il est possible d'utiliser
$(k-1)$-`CID` comme opérateur de sous-filtrage dans l'algorithme `k-CID`. `k-CID` est récursivement défini de la manière suivante :

Definition 6 (opérateur k-CID)
`1-CID` *est l'opérateur* `CID` *établissant la CID-cohérence.*
`k-CID` *est l'opérateur* `CID` *utilisant* $(k-1)$-`CID` *comme sous-filtrage.*

Comme la 4B-cohérence, la 2-CID-cohérence demeure une cohérence partielle théorique qui est rarement utile en pratique. En effet, son pouvoir de réduction est significatif, mais pour autant le temps de calcul requis pour obtenir les solutions avec `2-CID` plus bissection est souvent non compétitif avec le temps requis par `1-CID` plus bissection.

4 Un schéma de résolution basé sur CID

Pour trouver toutes les solutions d'un CSP numérique, nous proposons une stratégie incluant un opérateur de bissection, un filtrage CID et un opérateur de Newton sur intervalles. Entre deux bissections, deux opérations sont déclenchées en séquence :

1. un appel à `CID(s, w-hc4, n')` ;
2. un appel à un opérateur de Newton sur intervalles.

L'opérateur de sous-filtrage appelé par `CID` est `Box` ou `2B`.

Le choix de paramètres pour le filtrage CID

Nous détaillons dans cette section les paramètres choisis pour l'opérateur `CID`.

Le paramètre utilisateur standard `w-hc4` est utilisé par l'opérateur de sous-filtrage (`Box` ou `2B`) appelé par CID : une contrainte est empilée dans la queue de propagation si la projection sur ses variables réduit les intervalles correspondants de plus de `w-hc4` (pourcentage de la largeur de l'intervalle).

Le paramètre s est le nombre de tranches utilisé par CID, c'est-à-dire par la procédure `VarCID`.

Quoique utile pour calculer la propriété de CID-cohérence, la boucle `repeat` de point-fixe utilisée par l'algorithme `CID` ne paie pas en pratique, comme l'a montré le premier article sur CID [16]. Autrement dit, exécuter `LoopCID` plus d'une fois sur l'ensemble des variables est toujours contre-productif, même si la valeur de w est finement réglée.

Ces expérimentations nous ont conduits à fixer w à ∞, c'est-à-dire à l'enlever des paramètres de l'utilisateur.

Muni des paramètres s et `w-hc4`, nous verrons que le filtrage CID s'avère être un opérateur généraliste efficace qui a le potentiel pour remplacer d'autres opérateurs de filtrage tels que 2B/Box (seule) ou 3B. Pour quelques (rares) instances cependant, l'utilisation de 2B/Box seulement est plus pertinente.

C'est pourquoi nous introduisons un troisième paramètre n' qui offre un compromis entre 2B/Box seule et `CID` : si n' est réglé à la valeur 0, alors un simple filtrage 2B/Box est déclenché entre deux bissections ; si $n = n'$ (n est le nombre de variables dans le système), alors `CID` est appelé entre deux bissections. Dans le cas général, n' est défini comme le nombre de variables qui sont varcidées entre deux bissections, à tour de rôle : étant donné un ordre prédéfini entre les variables, une opération de `VarCID` est appliquée sur n' variables (modulo n), en commençant à la dernière variable varcidée (plus un). L'indice de la dernière variable varcidée est transmis aux différents nœuds de l'arbre de recherche. Les expérimentations montreront que `CID(s, w-hc4, n')` est généralement bien meilleur que la combinaison standard entre 2B/Box et Newton sur intervalles.

Valeurs par défaut des paramètres de CID

Une contribution de cet article est de proposer une combinaison généraliste de CID, Newton sur intervalles et bissection qui est efficace. Une telle stratégie est pertinente si des valeurs par défaut utiles sont disponibles : dans notre résolveur, les valeurs par défaut donnent l'opérateur : `CID(s=4, w-hc4=10%, n'=n)`. Les expérimentations nous ont en effet conduits à sélectionner $s = 4$ (cf. section 7.2) :

- Selon le benchmark considéré, le nombre optimal de tranches est compris entre 2 et 8.
- Sélectionner $s = 4$ produit généralement la meilleure performance. Dans les autres cas, la performance obtenue n'est pas très éloignée de celle obtenue avec le nombre de tranches optimal.

Variantes écartées

Plusieurs variantes de CID et combinaisons d'opérateurs ont été écartées par nos expérimentations. Nous estimons utile de les mentionner brièvement.

Nous avons d'abord essayé différentes combinaisons de bissection, CID et Newton sur intervalles. La combinaison décrite ci-dessus est la meilleure, mais ajouter un Newton sur intervalles à un sous-filtrage 2B/Box (c'est-à-dire à l'intérieur d'un filtrage CID) produit également des résultats intéressants, alors qu'une telle combinaison avec du rognage est contreproductive.

Ensuite, notre recommandation était d'oublier le paramètre de point-fixe w dans `CID`. En fait, de nombreuses expériences ont confirmé qu'il n'est pratiquement jamais utile de relancer plusieurs fois la procédure `LoopCID` entre deux bissections. Ce phénomène est clair si le critère de continuation est l'existence d'au moins une variable dont l'intervalle est réduit de plus de w. Il l'est tout autant si le critère concerne plusieurs dimensions, c'est-à-dire si la réduction de la taille (périmètre ou volume) de la boîte courante est suffisamment importante. Une troisième expérience qui exécute `LoopCID` deux fois entre deux bissections donne la même conclusion. Finalement, des expériences similaires ont été menées sans succès pour déterminer quelles variables spécifiques devraient être varcidées à nouveau de manière adaptative. Toutes ces expériences suggèrent fortement qu'un seul appel à `LoopCID`, c'est-à-dire varcider les variables une seule fois entre deux bissections, est en quelque sorte un effort maximal à investir dans le filtrage CID.

Finalement, dans le premier article sur CID [16], nous avions proposé une variante `CID246` de CID dans laquelle le nombre s de tranches était modifié entre deux bissections. Il était égal alternativement à 2, 4 ou 6 : $s = ((i \, modulo \, 3) + 1) \times 2)$, où i indique le i^e appel à `LoopCID`. La comparaison avec l'opérateur CID standard n'était en fait pas équitable parce que le paramètre s de CID était réglé à la valeur 2. Or, il se trouve que CID avec $s = 4$ est aussi performant que `CID246` tout en restant plus simple.

5 3B, CID et une version hybride 3BCID

Comme mentionné dans l'introduction, la cohérence partielle CID a plusieurs points communs avec la fameuse 3B-cohérence proposée par Lhomme [7].

Definition 7 (3B-cohérence)
Soit $P = (X, C, B)$ un NCSP.

Considérons $P_{\underline{x_i}}$ comme étant P où le domaine de $x_i \in X$ est réduit à sa borne inférieure. Considérons $P_{\overline{x_i}}$ comme étant P où le domaine de $x_i \in X$ est réduit à sa borne supérieure. Soit $P'_{\underline{x_i}}$ la clôture de $P_{\underline{x_i}}$ par 2B-cohérence. Soit $P'_{\overline{x_i}}$ la clôture de $P_{\overline{x_i}}$ par 2B-cohérence.

Une variable x_i de X est **3B-cohérent** *si $P'_{\underline{x_i}}$ et $P'_{\overline{x_i}}$ sont tous deux non vides (c'est-à-dire si les bornes ne peuvent pas être réfutées par 2B-cohérence).*

Le NCSP P est 3B-cohérent si toutes les variables de X sont 3B-cohérentes.

Pour des raisons d'efficacité, et contrairement aux CSP sur domaines finis, une cohérence partielle d'un NCSP est souvent obtenue avec une précision w [7]. Cette précision évite une convergence lente pour établir la propriété. La définition ci-dessus se généralise en considérant respectivement les intervalles $[\underline{\mathbf{x_i}}, \underline{\mathbf{x_i}} + w]$ et $[\overline{\mathbf{x_i}} - w, \overline{\mathbf{x_i}}]$ dans $P_{\underline{x_i}}$ et $P_{\overline{x_i}}$.

Quand le sous-filtrage est mis en œuvre par la Box-cohérence, au lieu de la 2B-cohérence, on obtient la propriété dite de *Bound consistency* [17].

L'algorithme de filtrage `3B` suit un principe similaire à `CID`, dans lequel `VarCID` est remplacé par un processus de rognage, appelé `VarShaving` dans cet article. En particulier, les deux algorithmes ne sont pas incrémentaux, si bien que la boucle `repeat` extérieure peut relancer le processus de rognage sur toutes les variables, comme le montre l'algorithme `3B`.

algorithm `3B` *(w : stop criterion precision,* `w3B` *: shaving precision, in-out $P = (X, C, B)$: an NCSP, F : subfiltering operator and its parameters)*
  repeat
    for every *variable $xi \in X$* **do**
      `VarShaving` (xi, `w3B`, P, F)
    end
  until `StopCriterion`*(w, P)*
end.

La procédure `VarShaving` réduit les bornes gauche et droite de la variable x_i en essayant de réfuter des intervalles de largeur au moins égale à `w3B` (pourcentage de la largeur initiale de l'intervalle). La proposition suivante permet de mieux comprendre la différence entre les filtrages 3B et CID.

Proposition 1 *Soit $P = (X, C, B)$ un NCSP. Soit F une cohérence partielle.*

Considérons la boîte B' obtenue par `CID` *(lui-même utilisant F), et la boîte B'' obtenue par* `3B` [2]*. Alors, le filtrage* `CID` *est plus fort que le filtrage* `3B`*, c'est-à-dire B' est inclus dans ou égal à B''.*

Cette propriété est basée sur le fait que, à cause de l'opération d'union (`Hull`) dans `VarCID`, la boîte B peut être réduite sur, potentiellement, toutes les dimensions. Avec `VarShaving`, l'effort de réduction porte seulement sur x_i, perdant ainsi les réductions temporaires obtenues par le sous-filtrage F sur l'ensemble des variables.

Dans le cas général cependant, la 3B-cohérence et la CID-cohérence ne sont pas comparables parce que s est le nombre exact d'appels au sous-filtrage F dans `VarCID` alors que $\frac{1}{\texttt{w3B}}$ est une borne supérieure du nombre d'appels à F dans `VarShaving`. D'autre part, la proposition établit que la capacité de filtrage de `CID` est meilleure que celle de `3B`. Les expérimentations confirmeront qu'il est préférable en pratique de faire avec `CID` un travail grossier sur une variable donnée (par exemple en choisissant $s = 4$) et de faire l'union de l'ensemble des déductions que de faire un travail plus fin avec `3B` (en choisissant par exemple $\texttt{w3B} = 5\%$) mais d'oublier les déductions sur les autres variables.

L'algorithme 3BCID

Nous avons vu que `3B` et `CID` suivent le même schéma algorithmique. Cela nous a amenés à concevoir plusieurs versions hybrides dont la plus prometteuse est présentée dans cette section.

`3BCID` gère deux paramètres : un nombre s de tranches pour la part liée à CID et un ratio `w3B` pour la part liée au rognage. Chaque variable x_i est traitée par des processus de rognage et de `VarCID` de la manière suivante.

L'intervalle de x_i est d'abord découpé en $\frac{1}{\texttt{w3B}}$ tranches traitées par rognage. En utilisant un sous-filtrage F, une procédure simple de rognage cherche à réfuter ces tranches par la gauche et par la droite (sans utiliser de processus dichotomique). Soit s_{left} (resp. s_{right}) la tranche de x_i la plus à gauche (resp. la plus à droite) qui n'a pas été réfutée par F, si elle existe. Soit $\mathbf{x'_i}$ l'intervalle restant, c'est-à-dire $\overline{s_{left}} \leq \underline{\mathbf{x'_i}} \leq \overline{\mathbf{x'_i}} \leq \underline{s_{right}}$.

Ensuite, si $\mathbf{x'_i}$ n'est pas vide, il est découpé en s tranches pour être traité par `VarCID` qui se termine

[2] Une hypothèse supplémentaire liée aux nombres flottants est nécessaire en théorie pour permettre une juste comparaison entre algorithmes (sans quoi, aucun point-fixe unique ne peut être atteint et deux appels au même algorithme conduiraient à des résultats différents). On doit donc supposer en outre que `CID` et `3B` gèrent des intervalles irréductibles formés d'un flottant ou de deux flottants consécutifs [7].

en réalisant l'union (Hull) des (au plus) $s+2$ boîtes traitées par le sous-filtrage F : s_{left}, s_{right} et les s tranches entre s_{left} et s_{right}.

On peut montrer trivialement que la cohérence partielle obtenue par 3BCID est plus forte que la 3B(w3B)-cohérence.

Les expérimentations montreront que 3BCID avec $s = 1$ se compare avantageusement à 3B et parfois à CID. Comme 3B, $\frac{1}{\text{w3B}}$ est une borne supérieure du nombre d'appels au sous-filtrage F, mais cette borne est souvent non atteinte, la borne inférieure étant 3 (c'est-à-dire réaliser l'union (Hull) entre les boîtes correspondant à la tranche la plus à gauche, à la tranche la plus à droite, et à l'intervalle entre les deux). De plus, réaliser cette disjonction constructive sur seulement deux ou trois tranches pour chaque variable produit néanmoins un filtrage sur, potentiellement, toutes les dimensions.

Pour conclure, 3BCID avec $s = 1$ peut se voir comme une version améliorée de la 3B où la disjonction constructive produit un effet de filtrage additionnel avec un faible surcoût.

6 Une nouvelle heuristique de bissection basée sur CID

Il existe trois principales heuristiques de bissection (c'est-à-dire pour choisir la prochaine variable à bissecter) pour résoudre les CSP numériques. La stratégie la plus simple, dite "à tour de rôle" (*round-robin*), boucle sur les variables. Une autre heuristique sélectionne la variable avec l'intervalle le plus large. Une troisième, basée sur la fonction *smear* [10], sélectionne une variable x_i impliquée dans les équations dont la dérivée par rapport à x_i est forte.

La stratégie à tour de rôle assure que toutes les variables sont bissectées dans une branche de l'arbre de recherche. Ce point est important car, contrairement à l'arbre de recherche en domaines finis, une variable est généralement découpée (instanciée) plusieurs fois avant d'atteindre la précision souhaitée. La stratégie du plus grand intervalle d'abord conduit également à ne pas toujours sélectionner une même variable. La stratégie basée sur la fonction *smear* peut parfois conduire à bissecter les mêmes variables, si bien qu'un schéma entrelacé avec la stratégie à tour de rôle ou une phase de préconditionnement est parfois nécessaire pour la rendre utilisable en pratique.

Nous introduisons dans cette section une nouvelle heuristique de bissection basée sur CID. Définissons tout d'abord un ensemble de (tailles de) boîtes apprises pendant l'exécution de la procédure VarCID appliquée à une variable x_i donnée :

- Soit OldBox_i la boîte B juste avant l'appel à VarCID sur x_i. Soit NewBox_i la boîte obtenue après l'appel à VarCID sur x_i.
- Soit $B_i^{l'}$ and $B_i^{r'}$ les boîtes gauche et droite calculées par VarCID, après le sous-filtrage par F et avant l'opération d'union (Hull).

Le ratio ratioBis suivant conduit à une nouvelle heuristique "intelligente" de bissection. Le ratio $\text{ratioBis}= \frac{f(Size(B_i^{l'}),Size(B_i^{r'}))}{Size(NewBox)}$ évalue en quelque sorte le filtrage perdu par application de l'opérateur Hull dans VarCID. En effet, $B_i^{l'}$ et $B_i^{r'}$ représentent précisément les boîtes que l'on obtiendrait si l'on effectuait une bissection sur x_i (au lieu de réaliser une opération d'union) immédiatement après l'appel à VarCID ; NewBox est la boîte obtenue par l'opération Hull de CID pour éviter une explosion combinatoire due à un point de choix.

Ainsi, après un appel à LoopCID, le travail de CID permet d'apprendre la prochaine variable à bissecter : *on sélectionne une variable qui produit le plus petit* ratioBis. Quoique non liées à la disjonction constructive, des stratégies similaires ont été appliquées aux CSP finis [4, 13].

Des expérimentations, non reportées dans cet article, ont comparé un grand nombre de variantes de ratioBis avec différentes fonctions f. Elles nous ont conduits à sélectionner la variante où $\text{ratioBis}= \frac{Size(B_i^{l'})+Size(B_i^{r'})}{Size(NewBox)}$.

7 Validation expérimentale

Nous avons mené un grand nombre de comparaisons et de tests sur un échantillon de 20 instances. Ces tests nous ont permis de concevoir et de valider les variantes efficaces de CID présentées dans cet article.

7.1 Benchmarks et outil de résolution par intervalles

Vingt benchmarks sont brièvement présentés dans cette section. Cinq d'entre eux sont des systèmes creux que l'on trouve dans la référence [11] : Hourglass, Tetra, Tangent, Ponts, Mechanism. Ils sont très difficiles à résoudre pour des techniques d'intervalles généralistes, mais l'algorithme IBB peut efficacement exploiter une décomposition préliminaire des systèmes en petits sous-systèmes [11]. Les autres benchmarks sont tirés de la page Web de l'équipe COPRIN ou de la page Web de COCONUT où le lecteur pourra trouver plus de détails [12]. La précision des solutions, c'est-à-dire la largeur au dessous de laquelle un intervalle n'est pas bissecté, est de $1e-08$ pour toutes les instances et de $5e-06$ pour Mechanism. L'opérateur 2B s'avère être le meilleur sous-filtrage local appelé par 3B ou CID sur toutes les instances sauf Yamamura8 pour lequel Box+2B est plus efficace. Toutes les instances ont été sélectionnées parce qu'elle sont résolues en un temps

acceptable par un algorithme standard, ce qui a permis de tester de nombreuses variantes. Aucun benchmark n'a été écarté pour une autre raison !

Les résultats ont été obtenus sur un Pentium IV à 2.66 Ghz en utilisant la bibliothèque de résolution par intervalles en C++ développée par le deuxième auteur. Ce nouveau solveur offre les opérateurs standard comme Box, 2B, Newton sur intervalles [10]. Il offre les heuristiques de bissection à tour de rôle (*round-robin*), du plus grand intervalle d'abord et celle basée sur CID. Quoique récente et en développement, cette bibliothèque semble compétitive avec des résolveurs connus comme RealPaver [5]. Le lecteur trouvera dans [11] une première évaluation sur dix systèmes creux. Pour tous les algorithmes présentés, y compris 3B et 3BCID, un Newton sur intervalles est appelé juste après une bissection si la largeur de l'intervalle le plus large dans la boîte courante est inférieure à $1e-2$.

7.2 Résultats obtenus par CID

Le tableau 1 reporte les résultats obtenus par CID(s, w-hc4, n'), tel que défini à la section 4.

La réduction forte du nombre de bissections requis (souvent plusieurs ordres de grandeur) souligne clairement le pouvoir filtrant de CID. De plus, des gains en temps impressionnants peuvent s'observer sur les benchmarks en haut de la table, en comparaison avec une stratégie de résolution standard utilisant 2B (ou 2B+Box), un Newton sur intervalles et une stratégie de bissection à tour de rôle[3].

CID obtient souvent de bien meilleurs temps de résolution que la stratégie standard sauf pour Bellido, Trigexp2-5 and Caprasse, où la perte de performance est faible. Cependant, il n'est pas raisonnable de proposer à l'utilisateur un opérateur avec trois paramètres à régler manuellement. C'est pourquoi les trois dernières colonnes CID reportent les résultats où seulement 0 ou 1 paramètre est réglé. Les valeurs par défaut (quatrième colonne CID avec $s=4$, w-hc4= 10%, $n'=n$) donnent de très bons résultats : ce CID sans paramètre surpasse la stratégie standard dans 15 des 21 instances. En particulier, sélectionner $s=4$ produit les meilleurs résultats dans 12 des 21 instances.

Les deuxième et troisième colonne CID reportent les très bons résultats obtenus par un algorithme de filtrage possédant exactement le même nombre de paramètres que 2B or Box. Ainsi, puisque CID avec un seul paramètre n' à régler (seconde colonne CID) permet un continuum entre une pure 2B ($n'=0$ revient à un appel à 2B) et CID, une première recommandation serait de proposer cette variante de CID dans un solveur par intervalles.

La seconde recommandation repose sur une considération combinatoire. En un sens, la disjonction constructive sur intervalles peut se voir comme une "bissection en temps polynomial" puisque VarCID effectue une union du travail accompli sur les différentes tranches. Cependant, l'aspect exponentiel de la bissection (classique) devient chronophage seulement quand le nombre de variables devient grand. Cela pourrait expliquer pourquoi CID n'est pas efficace sur Trigexp2-5 et Caprasse qui ont un très petit nombre de variables et ne conduisent donc pas à une explosion combinatoire liée à la bissection. (Cette intuition est confirmée par le meilleur comportement de CID sur la variante - que l'on peut passer à l'échelle - de Trigexp2 à 9 variables.) Ainsi, la deuxième recommandation serait d'utiliser CID sur les systèmes ayant un nombre minimal de variables, par exemple 5 ou 8.

7.3 Comparaison entre CID, 3B et 3BCID

Le tableau 2 reporte les résultats obtenus par CID, 3B et 3BCID (cf. section 5). Tous les résultats ont été obtenus avec un paramètre w-hc4 fixé à 5%.

Pour chaque algorithme, plusieurs essais ont été réalisés avec différentes valeurs des paramètres, et le meilleur résultat est reporté dans la table. Pour 3B, sept valeurs du paramètre w3B ont été essayées : 1%, 2%, 5%, 10%, 15%, 20%, 30%. Pour CID, neuf valeurs des paramètres ont été essayées : cinq valeurs pour le nombre de tranches s (2, 3, 4, 6, 8 ; on fixe $n'=n$), et quatre valeurs pour le paramètre n' (1, $0.5\,n$, $0.75\,n$, $1.2\,n$; on fixe $s=4$). Pour 3BCID, huit combinaisons des paramètres ont été essayées : quatre valeurs pour s (1,2,3,4) combinées avec deux valeurs pour w3B (5%, 10%).
Les principales conclusions déduites de la table 2 sont les suivantes :

- 3BCID et CID surpassent toujours 3B. De plus, 3B est battue par la stratégie standard pour 9 benchmarks du bas de la table.
- 3BCID est compétitif avec CID. 3BCID est meilleur que CID en ce qui concerne 11 benchmarks. CID demeure bien meilleur que 3BCID sur Mechanism. On peut se demander si ce phénomène est lié au grand nombre de variables de cette instance.

La bonne nouvelle est que la meilleure valeur pour s dans 3BCID est souvent $s=1$. Dans seulement quatre cas, la meilleure valeur est $s=2$, mais la valeur $s=1$ produit également de très bons résultats. Cela suggère de proposer 3BCID avec $s=1$ comme alternative à la 3B. En d'autres termes, 3BCID avec $s=1$ peut se voir comme une implémentation prometteuse de la 3B. C'est transparent pour l'utilisateur qui doit seule-

[3] Notons que cette stratégie est inefficace pour Trigexp1 (résolu en 371 secondes) et pour Mechanism (non résolu après un temps limite (TO) de plusieurs heures). Cependant, un temps de résolution raisonnable peut être obtenu avec une variante de 2B qui empile *toutes* les contraintes du NCSP dans la queue de propagation après chaque bissection.

Nom	n	**#s**	w-hc4	2B/Box + Newton	CID $(s, \texttt{w-hc4}, n')$	CID $(4, 10\%, n')$	CID $(4, \texttt{w-hc4}, n)$	CID $(4, 10\%, n)$	s	w-hc4	n'
BroydenTri	32	2	15%	758 2e+07	**0.12** 46	0.28 44	0.19 65	0.45 50	4	80%	40
Hourglass	29	8	5%	24 1e+05	**0.44** 109	0.44 109	0.52 80	0.52 80	4	10%	17
Tetra	30	256	0.02%	401 1e+06	**10.1** 2116	11.6 1558	11.7 1690	14.5 1320	4	30%	20
Tangent	28	128	15%	32 1e+05	**3.7** 692	3.7 692	4.9 447	5.1 450	4	50%	28
Reactors	20	38	5%	156 1e+06	**15.6** 2588	16.4 2803	16.7 2381	17.7 2156	4	15%	18
Trigexp1	30	1	20%	371(3.4) 5025	**0.12** 1	0.15 3	0.14 2	0.14 3	8	2%	30
Discrete25	27	1	0.01%	5.2 1741	**0.62** 2	1.08 3	0.84 12	2.13 99	8	0.5%	35
I5	10	30	2%	692 3e+06	**126** 23105	147 60800	150 20874	157 32309	6	2%	5
Transistor	12	1	10%	179 1e+06	**66** 11008	79.4 31426	91.4 16333	91.4 16333	8	10%	6
Ponts	30	128	5%	10.8 34994	**2.7** 388	2.9 338	2.9 380	3.1 304	4	30%	25
Yamamura8	8	7	1%	13 1032	**7.5** 104	9.5 60	9.5 60	9.5 60	4	10%	4
Design	9	1	10%	395 3e+06	**275** 200272	278 256000	313 76633	313 76633	5	10%	2
D1	12	16	5%	4.1 35670	**1.7** 464	1.7 464	1.7 464	1.7 464	4	10%	12
Mechanism	98	448	0.5%	TO(111) 24538	**43.1** 3419	45.2 3300	46.6 2100	47.8 2420	4	2%	50
Hayes	8	1	0.01%	155 3e+05	**75.8** 1e+05	77 1e+05	111 81750	147 58234	4	80%	2
Kin1	6	16	10%	84 70368	**76.8** 6892	76.8 6892	83.5 4837	87.4 4100	4	10%	3
Eco9	8	16	10%	26 2e+05	**18** 55657	19.4 46902	26.6 10064	26.6 10064	3	10%	1
Bellido	9	8	10%	**80** 7e+05	94.4 1e+05	94.4 1e+05	106 45377	106 45377	4	10%	3
Trigexp2-9	9	1	20%	61.8 3e+05	**50.4** 4887	65.14 14528	62.4 11574	68.4 9541	6	10%	9
Trigexp2-5	5	1	20%	**3.0** 13614	3.8 10221	4.6 4631	6.2 2293	6.6 1887	2	20%	1
Caprasse	4	18	30%	**2.6** 37788	2.73 18176	3.0 12052	4.7 5308	5.1 5624	2	5%	1

TAB. 1 – Comparaison entre [CID + Newton sur intervalles + bissection à tour de rôle (round-robin)] et [une stratégie standard] : 2B/Box + Newton sur intervalles + bissection à tour de rôle. La colonne n indique le nombre de variables. La colonne #s donne le nombre de solutions. La première colonne w-hc4 donne la valeur choisie pour le paramètre w-hc4 utilisé par 2B ou Box. Les trois dernières colonnes s, w-hc4 et n' indiquent les valeurs des paramètres qui ont été réglés pour CID (première colonne CID). La deuxième colonne CID reporte les résultats de CID quand $s = 4$ et w-hc4 $= 10\%$, c'est-à-dire quand seul n' est réglé. La troisième colonne CID reporte les résultats de CID quand $s = 4$ et $n' = n$, c'est-à-dire quand seul w-hc4 est réglé. La quatrième colonne CID reporte les résultats de CID avec les valeurs par défaut pour les trois paramètres : $s = 4$, w-hc4$= 10\%$, $n' = n$. Chaque case contient deux valeurs : le temps CPU en secondes pour calculer toutes les solutions (en haut), et le nombre de bissections requis pour calculer toutes les solutions (en bas). Pour chaque benchmark, le meilleur temps apparaît en gras.

Nom	n	2B/Box	CID	3B	3BCID($s=1$)	3BCID($s=2$)
BroydenTri	32	758	0.23	0.22	**0.18**	0.19
Hourglass	29	24	0.45	0.73	**0.43**	0.50
Tetra	30	401	**13.6**	20.7	17.1	18.8
Tangent	28	32	4.13	8.67	**3.18**	4.13
Reactors	20	156	18.2	24.2	**15.5**	16.9
Trigexp1	30	3.4	**0.10**	0.26	0.12	0.11
Discrete25	27	5.2	1.37	2.19	1.26	**1.13**
I5	10	692	139	144	**115**	123
Transistor	12	179	71.5	77.9	49.3	**46.9**
Ponts	30	10.8	**3.07**	5.75	4.19	4.43
Yamamura8	8	13	**9.0**	9.1	10.3	10.7
Design	9	395	300	403	**228**	256
D1	12	4.1	1.78	2.99	**1.64**	1.76
Mechanism	98	111	**79**	185	176	173
Hayes	8	155	**99**	188	102	110
Kinematics1	6	84	**76.1**	136	76.6	81.4
Eco9	8	26	**19.3**	40.1	27.0	30.3
Bellido	9	**80**	95	143	**93**	102
Trigexp2-9	9	61.8	52.2	74.5	**39.9**	45.1
Caprasse	4	**2.6**	**3.1**	9.38	4.84	5.35

TAB. 2 – Comparaison entre CID, 3B et 3BCID. Les trois premières colonnes rappellent le nom de l'instance, son nombre de variables et le temps CPU en secondes requis pour le résoudre avec la stratégie standard 2B/Box + Newton + bissection à tour de rôle. Les autres colonnes donnent le temps CPU en secondes requis pour résoudre les instances avec CID, 3B et 3BCID. Le meilleur temps CPU apparaît en gras.

ment spécifier le paramètre habituel w3B, la gestion de la disjonction constructive étant cachée dans 3BCID (qui applique des procédures VarCID sur au plus trois tranches puisque $s = 1$).

7.4 Comparaison entre heuristiques de choix de bissection

La table 3 applique les trois heuristiques de bissection au schéma CID (avec $n' = n$ et $s = 4$) + Newton sur intervalles + bissection. Nous soulignons quelques observations.

L'heuristique basée sur CID est meilleure que les deux autres sur 13 des 20 instances, en particulier sur Design. La stratégie du plus large intervalle d'abord est la meilleure sur une seule instance. La stratégie à tour de rôle l'emporte dans 6 des 20 instances. Sur les 7 instances où l'heuristique basée sur CID n'est pas la meilleure, la perte de performance est significative sur Hayes.

L'heuristique de bissection basée sur CID se compare encore mieux à ses compétiteurs quand $s = 6$, la stratégie à tour de rôle étant la meilleure sur seulement 3 instances (tests non reportés dans cet article). Cela peut suggérer qu'un ratio ratioCID est mieux appris pendant l'exécution de VarCID quand le nombre de tranches est plus grand.

8 Conclusion

Ce papier a introduit deux nouveaux opérateurs de filtrage basés sur le principe de disjonction constructive exploité dans les problèmes combinatoires. Les premiers résultats expérimentaux sont très encourageants si bien que nous pensons que CID et 3BCID ont le potentiel pour devenir des opérateurs standard dans les solveurs de contraintes sur intervalles. L'opérateur CID ouvre également la porte à une nouvelle stratégie de bissection utilisant sans surcoût le travail de CID.

Les expérimentations conduisent à des recommandations claires concernant ces nouveaux opérateurs de filtrage. D'abord, CID peut être utilisé avec des valeurs fixées de s et w-hc4, en laissant le soin à l'utilisateur de régler le troisième paramètre n' (c'est-à-dire le nombre de variables qui sont varcidées entre deux bissections). Cela permet à l'utilisateur de sélectionner en quelque sorte un taux de filtrage CID, $n' = 0$ équivalant au pur filtrage 2B/Box. Ensuite, 3BCID avec $s = 1$ est proposé comme alternative prometteuse à l'opérateur 3B.

Plusieurs questions demeurent ouvertes. Il semble que, dû à des considérations combinatoires, CID n'est pas adapté à la résolution de petits problèmes alors qu'il semble prometteur pour les problèmes à grande échelle. Des études expérimentales plus poussées devraient confirmer ou infirmer cette hypothèse. En

Filtrage	CID	CID	CID
Bissection	Round-robin	Largest Int.	CID-based
BroydenTri	0.21	0.18	**0.17**
Hourglass	0.52	0.51	**0.37**
Tetra	**12.1**	28.2	16.4
Tangent	**3.7**	21.7	5.2
Reactors	17.0	13.2	**12.7**
Trigexp1	0.15	0.19	**0.14**
Discrete25	**0.84**	1.49	1.06
I5	**151**	421	179
Transistor	93	**36**	41
Ponts	2.92	5.51	**2.31**
Yamamura8	9.5	6.9	**5.1**
Design	318	334	**178**
D1	**1.72**	2.96	2.50
Mechanism	47	49	**46**
Hayes	**115**	564	318
Kinematics1	83	70	**63**
Eco9	26.7	31.4	**26.1**
Bellido	107	102	**99**
Trigexp2-9	62	55	**53**
Caprasse	5.16	5.43	**5.04**

TAB. 3 – Comparaison sur CID avec trois heuristiques de bissection : à tour de rôle (colonne Round-robin), choix du plus large intervalle d'abord (Largest int.) et la nouvelle heuristique basée sur CID (CID-based).

outre, 3BCID devrait être comparé à l'opérateur weak-3B implémenté dans RealPaver [5][4]. Enfin, la stratégie de bissection basée sur CID mérite une étude plus poussée.

L'un des travaux futurs les plus intéressants est de proposer des variantes *adaptatives* de CID qui permettraient de choisir quelle prochaine variable doit être varcidée ou bissectée, et comment, c'est-à-dire avec quelle force de travail.

Remerciements

Nos plus grands remerciements vont à Olivier Lhomme pour ses commentaires sur ces travaux de recherche. Nous remercions également Bertrand Neveu et les relecteurs anonymes pour leurs remarques pertinentes sur cet article.

Références

[1] R. Barták and R. Erben. A new Algorithm for Singleton Arc Consistency. In *Proc. FLAIRS*, 2004.

[2] C. Bessière and Debruyne R. Optimal and Suboptimal Singleton Arc Consistency Algorithms. In *Proc. IJCAI*, pages 54–59, 2005.

[3] R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.

[4] P.A. Geelen. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *Proc. ECAI'92*, pages 31–35, 1992.

[5] L. Granvilliers and F. Benhamou. RealPaver : An Interval Solver using Constraint Satisfaction Techniques. *ACM Trans. on Mathematical Software*, 32(1), 2006.

[6] C. Lecoutre and S. Cardon. A Greedy Approach to Establish Singleton Arc Consistency. In *Proc. of IJCAI'05*, pages 199–204, 2005.

[7] O. Lhomme. Consistency Tech. for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.

[8] O. Lhomme. Quick Shaving. In *Proc. AAAI*, pages 411–415, 2005.

[9] C. Min Li and Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems. In *Proc. IJCAI*, pages 366–371, 1997.

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

[11] B. Neveu, G. Chabert, and G. Trombettoni. When Interval Analysis helps Interblock Backtracking. In *Proc. CP'06, LNCS 4204*, pages 390–405, 2006.

[12] Web page of COPRIN : www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html
COCONUT benchs : www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html.

[13] P. Refalo. Impact-Based Search Strategies for Constraint Programming. In *Proc. CP'04, LNCS 3258*, pages 557–571, 2004.

[14] J.C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. AAAI*, pages 362–367, 1994.

[15] H. Simonis. Sudoku as a Constraint Problem. In *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, pages 13–27, 2005.

[16] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *CP Workshop IntCP*, 2006.

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

[18] P. Van Hentenryck, V. Saraswat, and Deville Y. Design, Implementation, and Evaluation of the Constraint Language CC(FD). *J. Logic Programming*, 37(1–3) :139–164, 1994.

[4] Une implémentation des deux opérateurs dans un même solveur permettrait une comparaison équitable.