Retour-arrière inter-blocs et résolution par intervalles

Bertrand Neveu et Gilles Trombettoni
Projet COPRIN, CERTIS-INRIA-I3S
2004 route des lucioles
06902 Sophia Antipolis Cedex BP 93
email: neveu,trombe@sophia.inria.fr

Christophe Jermann
LINA, 2 rue de la Houssinière
BP92208, 44322 Nantes Cedex03
email: Christophe.Jermann@lina.univ-nantes.fr

Résumé

Cet article présente une technique, appelée Retour-arrière inter-blocs (en anglais InterBlock Bactracking IBB), qui améliore la résolution par intervalles de systèmes d'équations non linéaires sur les réels quand ces systèmes sont décomposables.

Cette technique, introduite en 1998 par Bliek et al, traite un système d'équations décomposé au préalable en un ensemble de (petits) sous-systèmes $k \times k$, appelés blocs. Une solution est obtenue en combinant les solutions partielles calculées dans les différents blocs. Cette approche semble particulièrement intéressante pour accélérer les techniques de résolution par intervalles.

Dans cet article, nous analysons en détail différentes variantes de IBB, qui diffèrent dans leurs stratégies de retour-arrière et de filtrage. Nous introduisons aussi IBB-GBJ, une nouvelle variante basée sur le *graph-based backjumping* de Dechter.

Une comparaison complète sur huit problèmes nous a permis de mieux comprendre le comportement de IBB. Il montre que les variantes IBB-BT+ et IBB-GBJ sont de bons compromis entre simplicité et performance. De plus, cela montre que limiter le filtrage à l'intérieur des blocs est intéressant. Pour toutes les instances testées, IBB gagne plusieurs ordres de grandeur par rapport à une résolution globale sans décomposition.

Mots-clés: intervalles, décomposition, retour-arrière, systèmes peu denses

1 Introduction

Il existe seulement quelques techniques pour calculer toutes les solutions d'un système d'équations non linéaires sur les réels. Les techniques de calcul formel, comme les bases de Gröbner [Buchberger, 1985] et les méthodes de Ritt-Wu [Wu, 1986] sont souvent très coûteuses en temps et en mémoire et sont limitées à des systèmes d'équations algébriques de petite taille. La méthode par continuation, appelée aussi homotopie [Lahaye, 1934; Durand, 1998], peut donner de très bons résultats. Cependant, éliminer les solutions complexes et prendre en compte les inégalités n'est pas simple. De plus, la méthode doit partir d'un système initial "proche" du système à résoudre. Cela rend difficile une automatisation, particulièrement pour des systèmes non algébriques.

Les techniques par intervalles sont des alternatives prometteuses, quand les domaines des variables sont bornés a priori, ce qui est souvent le cas. Elles prennent aussi facilement en compte les inégalités. Elles ont obtenu de bons résultats dans plusieurs domaines, comme la commande robuste [Jaulin *et al.*, 2001] et la robotique [Merlet, 2002]. Cependant, ces méthodes sont assez lentes et leur efficacité décroît avec la taille du système à résoudre: il est reconnu que des systèmes avec des centaines (parfois des dizaines) de contraintes non linéaires ne peuvent être résolus en pratique.

Dans certaines applications faites de contraintes non linéaires, les systèmes sont peu denses et peuvent être décomposés en sous-systèmes par des techniques équationnelles ou géométriques. La CAO, la reconstruction de scène 3D avec des contraintes géométriques [Wilczkowiak *et al.*, 2003], la chimie moléculaire [Hendrickson, 1992] et la robotique représentent de tels champs d'applications prometteurs. Différentes techniques peuvent être utilisées pour décomposer de tels systèmes en *blocs* $k \times k$. Les décompositions équationnelles travaillent sur le graphe biparti formé par les variables et les équations [Ait-Aoudia *et al.*, 1993; Bliek *et al.*, 1998]. Quand les équations représentent des contraintes géométriques, les décompositions géométriques produisent généralement des blocs plus petits [Hoffmann *et al.*, 1997; Jermann *et al.*, 2000].

L'approche originale, introduite en 1998 [Bliek *et al.*, 1998], appelée dans cet article *Retour-arrière inter-blocs* (en anglais InterBlock Backtracking `IBB`), peut être utilisée après cette phase de décomposition. En suivant l'ordre partiel entre les blocs donné par la décomposition, on peut appliquer un processus de résolution classique pour chaque bloc, traitant ainsi des systèmes de taille réduite. `IBB` combine les solutions partielles obtenues pour construire les solutions globales du problème.

Bien que `IBB` puisse être utilisé avec d'autres types de résolveurs, nous avons intégré des techniques d'intervalles qui sont très générales et efficaces sur des systèmes réduits. Le premier article [Bliek *et al.*, 1998] a présenté les premières versions de `IBB` en incluant plusieurs schémas de retour-arrière et une technique de décomposition équationnelle. Depuis, plusieurs variantes de `IBB` ont été développées sans être présentées en détails et dans [Jermann *et al.*, 2000] nous nous sommes par ailleurs intéressés à des techniques de décomposition géométrique utilisant des algorithmes de flots pour détecter des parties rigides.

Contributions

Cet article détaille les phases de résolution réalisées par IBB avec des techniques par intervalles. Il apporte plusieurs contributions :

- De nombreuses expérimentations ont été réalisées sur un banc d'essais de plus grande taille (entre 30 et 178 équations). Elles ont conduit à une comparaison plus équitable entre les variantes. Ainsi, cela a confirmé que IBB pouvait gagner plusieurs ordres de grandeur en temps de calcul par rapport aux techniques par intervalles appliquées sur le système entier non décomposé. Finalement, cela nous a permis de mieux comprendre les subtilités de l'intégration des techniques d'intervalles dans IBB.
- Une nouvelle version de IBB est présentée, basée sur l'algorithme bien connu en domaines finis GBJ [Dechter, 1990]. Les expérimentations ont montré que IBB-GBJ est un bon compromis entre les versions antérieures.
- Les expérimentations ont également montré qu'il valait mieux limiter le filtrage à l'intérieur de chaque bloc plutôt que de mettre en œuvre un filtrage inter-blocs.

Contenu

La partie 2 pose quelques hypothèses sur les problèmes qui peuvent être traités. La partie 3 rappelle les principes de IBB et de la résolution par intervalles. La partie 4 détaille l'algorithme IBB-GBJ et la stratégie de propagation inter-blocs. La partie 5 présente les expérimentations réalisées sur un échantillon de huit problèmes. Les résultats sont analysés dans la partie 6.

2 Hypothèses

IBB résout un système décomposé d'équations sur les réels. Tout type d'équation peut être traité, algébrique ou non. Notre banc d'essais comprend des équations linéaires et quadratiques. IBB est utilisé pour trouver **toutes** les solutions d'un système de contraintes. Il pourrait être modifié pour l'optimisation globale (choisir la solution minimisant un critère donné) en remplaçant le retour-arrière inter-blocs par un algorithme classique de séparation-évaluation (*Branch and Bound*). Rien n'a été fait dans cette direction pour le moment.

Nous faisons l'hypothèse que les systèmes à résoudre ont un nombre fini de solutions ponctuelles. Ainsi, cela permet à IBB de combiner un ensemble fini de solutions partielles. Cette condition vaut donc ainsi pour chaque sous-système (bloc), qui doit donc être un sous-système carré, c.-à-d qui contient autant d'équations que de variables.

Aucune autre hypothèse ne doit être posée sur la technique de décomposition. Cependant, comme nous utilisons une décomposition structurelle, il ne doit pas y avoir d'équations redondantes. Les inégalités ou les équations supplémentaires utilisées pour réduire le nombre de solutions peuvent être ajoutées facilement pendant la phase de résolution dans le bloc correspondant à leurs variables (comme par exemple, en tant que contraintes "soft" dans Numerica [Van Hentenryck *et al.*, 1997]), mais cette intégration est en dehors du champ de cet article.

Pour traiter certaines redondances dans la phase de décomposition, des décompositions "symboliques" pourraient aussi être envisagées [Bondyfalat *et al.*, 1999]. Ces algorithmes

basés sur des techniques de calcul formel peuvent tenir compte des *coefficients* présents dans les contraintes, et pas seulement des dépendances structurelles.

Remarque

En pratique, les problèmes qui peuvent être décomposés sont souvent sous-contraints et ont plus de variables que d'équations. Cependant, dans les applications existantes, le problème peut être rendu carré (autant de variables que d'équations) en donnant une valeur à un sous-ensemble de variables appelées *paramètres d'entrée*. Les valeurs de ces paramètres peuvent être fournies par l'utilisateur, lues sur une esquisse ou données par un processus préliminaire (par exemple, en reconstruction de scène 3D [Wilczkowiak *et al.*, 2003]).

3 Fondements

Cette partie présente d'abord brièvement la résolution par intervalles. La version la plus simple de IBB est ensuite introduite sur un exemple.

3.1 Techniques de résolution par intervalles

CSP numériques

Un problème de satisfaction de contraintes numériques (NCSP) $P = (V,C,I)$ contient un ensemble de contraintes C et un ensemble de n variables V. Chaque variable $v_i \in V$ peut prendre une valeur réelle dans l'intervalle $d_i \in I$; les bornes de d_i sont des nombres flottants. Résoudre P consiste à affecter des valeurs aux variables de V de telle sorte que les contraintes de C soient satisfaites.

Un produit cartésien d'intervalles en dimension n peut être représenté par un parallélépipède rectangle n-dimensionnel appelé **boîte**. Les algorithmes de résolution vont donc manipuler des boîtes, en appliquant des bissections et des propagations de contraintes qui servent à éliminer des boîtes sans solution. Ils peuvent ainsi isoler des boîtes pouvant contenir des solutions.

Les nombres réels ne pouvant pas être représentés de manière exacte dans les ordinateurs, le processus de résolution s'arrête quand une très petite boîte a été obtenue. Une telle boîte est appelée boîte atomique dans cet article. En théorie, une boîte atomique peut avoir la largeur existant entre deux flottants consécutifs, En pratique, le processus est interrompu quand tous les intervalles contiennent w_1 flottants [1]. Il est important de remarquer qu'une boîte atomique ne contient pas nécessairement une solution. En effet, le processus est semi-déterministe: évaluer une égalité avec l'arithmétique des intervalles peut prouver que la relation n'a pas de solution (quand les boîtes de gauche et de droite ont une intersection vide), mais ne permet pas d'affirmer qu'il existe une solution dans l'intersection.

1. w_1 est un paramètre défini par l'utilisateur. Dans la plupart des implantations, w_1 est une largeur et non un nombre de flottants.

Résolveur utilisé dans IBB

Nous utilisons `IlogSolver` et sa bibliothèque `IlcNum`. `IlcNum` implante la plupart des caractéristiques du système `Numerica` [Van Hentenryck *et al.*, 1997]. Cette bibliothèque utilise plusieurs principes développés dans l'analyse par intervalles et en programmation par contraintes. Le processus de résolution suivi par `IBB` peut être résumé de la manière suivante :

1. *Bissection:* on choisit une variable et on coupe son domaine en deux intervalles (la boîte est coupée selon l'une de ses dimensions). Cela donne deux sous-CSP qui sont traités en séquence et rend le processus de résolution combinatoire.
2. *Filtrage/propagation:* De l'information locale (sur une contrainte) ou plus globale (3B) est utilisée pour réduire la boîte courante ou arrêter une branche détectée sans solution.
3. *Test d'unicité:* un test utilisant des méthodes d'analyse numérique est réalisé sur le système d'équations en prenant en compte les dérivées premières et secondes des équations. Quand ce test réussit sur la boîte courante, il assure qu'il existe une solution unique et qu'un algorithme numérique classique (la méthode de Newton) peut converger vers cette solution.

Ces trois étapes sont réalisées en boucle. Le processus s'arrête dans une branche quand une boîte atomique d'une taille inférieure à w_1 a été obtenue, ou quand le test d'unicité est vérifié sur la boîte courante.

La propagation est réalisée par un algorithme de point fixe de type AC3. Quatre types de filtrage affinent les bornes des intervalles (sans créer de trou : on ne manipule pas d'unions d'intervalles). La `box-consistance` [Van Hentenryck *et al.*, 1997] provient de `IlcNum`, la `2B-consistance` provient de l'implantation initiale des CSP numériques dans `IlogSolver`. Bien qu'algorithmiquement différentes, ce sont des consistances locales qui ne considèrent qu'une contrainte à la fois en réduisant les bornes des variables impliquées. La `3B-consistance` [Lhomme, 1993] utilise la `2B-consistance` comme sous-procédure et un principe de réfutation (rognage) pour réduire les bornes de chaque variable. La `bound-consistance` suit le même principe, mais utilise la `box-consistance` comme sous-procédure. Un paramètre w_2 doit être spécifié pour la `bound` ou la `3B` : la borne d'une variable n'est pas mise à jour si la réduction de l'intervalle est inférieure à w_2. Le paramètre w_1 est aussi utilisé pour éviter un grand nombre de propagations dans le cas de convergence lente de la `2B` ou `Box` : une réduction n'est effectuée que si la portion d'intervalle à enlever est supérieure à w_1.

Le test d'unicité a été implanté dans `IlcNum`. Malheureusement, il ne peut être découplé de la `Box` ou de la `Bound` et il ne peut pas être appelé quand le filtrage est celui de `2B` ou de la `3B` seul. Cela nous empêche d'analyser finement son comportement.

3.2 IBB-BT

`IBB` traite un **Graphe sans circuit** de blocs (**DAG**) produit par une technique de décomposition. Un **bloc** i est un sous-système comprenant équations et variables. Certaines variables de i, appelées **variables d'entrée**, seront remplacées par leurs valeurs lors de la résolution du bloc. Les autres variables sont appelées **variables de sortie**. Un bloc **carré** a autant d'équations que de variables de sortie. Il existe un arc d'un bloc i vers

un bloc j si et seulement si une équation de j contient au moins une variable de sortie de i. Le bloc i est appelé parent de j. Le DAG induit un ordre partiel pour la résolution effectuée par IBB.

Exemple

Nous allons illustrer le principe de IBB en prenant l'exemple de CAO mécanique 2D introduit dans [Bliek *et al.*, 1998] (cf Fig. 1).

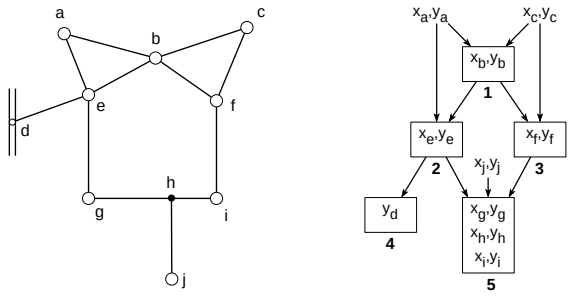

FIG. 1 – *Graphe de blocs : exemple.*

Différents points (cercles blancs) sont reliés par des barres rigides (segments). Les barres imposent une contrainte de distance entre deux points. Le point h (cercle noir) diffère des autres par le fait qu'il est sur la barre $\langle g,i\rangle$. Enfin, le point d est contraint à coulisser sur la droite spécifiée. Le problème est de trouver une configuration des points qui satisfait toutes les contraintes. Une technique de décomposition équationnelle produit le DAG montré sur la figure 1-droite.

Illustration de IBB

Devant respecter l'ordre partiel du DAG, IBB suit donc un des ordres totaux induits, par exemple le bloc 1, puis 2, 3, 4 et 5. Il appelle d'abord le résolveur par intervalles sur le bloc 1 et obtient une première solution pour x_b et y_b (le bloc a deux solutions). Une fois cette solution obtenue, on peut substituer x_b et y_b par leurs valeurs dans les équations des blocs suivants : 2 et 3. Puis, on traite les blocs 2, 3, 4 et 5 de façon similaire.

Quand un bloc n'a pas de solution, il faut revenir en arrière. Un retour-arrière chronologique revient sur le bloc précédent. IBB calcule la solution suivante pour ce bloc et réessaye de résoudre les blocs en aval. Cependant, on peut remarquer que ce retour-arrière chronologique de la version IBB-BT ne prend pas en compte l'ordre partiel du DAG. En effet, supposons que, dans l'exemple ci-dessus, le bloc 5 n'ait pas de solution. Le retour arrière chronologique reviendra sur le bloc 4, en trouvera une solution différente et tentera à nouveau de résoudre le bloc 5. Il est clair que le même échec se reproduira, les équations du bloc 5 ne comprenant pas de variable du bloc 4.

Nous avons expliqué dans [Bliek *et al.*, 1998] que les schémas CBJ et Dynamic Backtracking provenant des CSP en domaines finis ne peuvent pas être directement utilisés. En effet, un échec dans un bloc ne permet pas de discriminer finement le bloc parent qui est la cause de l'échec. Les contraintes sont généralement n-aires et font intervenir plusieurs parents et un échec est donc dû aux valeurs de tous les parents. Un retour

arrière "intelligent" IBB-GPB, basé sur le "partial order backtracking" [McAllester, 1993; Bliek, 1998] avait été présenté. La principale difficulté dans l'implantation de IBB-GPB est de maintenir un ensemble de conflits appelés "nogoods". De plus, toute modification de IBB-GPB, pour ajouter une caractéristique ou une heuristique, comme le filtrage inter-blocs, demande une grande attention.

Nous présentons dans cet article une variante plus simple basée sur le "graph-based backjumping" (GBJ) de Dechter [Dechter, 1990], et nous le comparons à IBB-GPB et à IBB-BT.

Remarque

De part l'utilisation de la résolution par intervalles, une solution d'un bloc ne correspond pas à un ensemble de valeurs réelles, mais à une boîte atomique. Ainsi, le remplacement des variables de sortie des blocs parents reviendrait à introduire de petits intervalles constants de largeur w_1 dans le bloc courant. Mais, le résolveur utilisé ne traite pas les intervalles constants et nous avons dû recourir à une heuristique appelée **heuristique du point milieu** qui remplace un intervalle constant par le nombre flottant situé au milieu de l'intervalle. Cette heuristique a plusieurs conséquences qui sont analysées dans la partie 6.1.

4 Utilisation de la structure du DAG et filtrage inter-blocs

La structure du DAG peut être prise en compte de deux manières :

- *en descendant:* une *condition de recalcul* peut éviter de recalculer inutilement les solutions d'un bloc;
- *en remontant:* quand un bloc n'a pas de solution, on peut revenir sur un bloc parent, et non pas nécessairement sur le bloc précédent.

Les deux paragraphes suivants présentent ces améliorations. Le troisième détaille le filtrage inter-blocs qui peut être ajouté à tous les schémas de retour-arrière. Ceci conduit à plusieurs variantes de IBB qui ont été testées sur notre banc d'essais.

4.1 La condition de recalcul

Cette condition peut être testée dans toutes les variantes de IBB, même IBB-BT. Vérifier cette condition de recalcul n'est pas coûteux et peut apporter des gains importants.

La **condition de recalcul** indique qu'il est inutile de recalculer les solutions d'un bloc si les variables des parents n'ont pas changé de valeur. Dans ce cas, IBB peut réutiliser les solutions calculées la dernière fois que le bloc a été traité. Illustrons ce point sur l'exemple didactique résolu par IBB-BT.

Supposons qu'une première solution ait été calculée pour le bloc 3, et que toutes les solutions calculées dans le bloc 4 aient conduit à un échec. IBB-BT revient alors sur le bloc 3 et la seconde position du point f est calculée. Quand IBB redescend sur le bloc 4, ce bloc devrait normalement être recalculé suite aux modifications de la solution courante

de f. Mais ni x_f ni y_f ne sont présents dans les équations du bloc 4, ainsi les deux solutions du bloc 4 calculées auparavant peuvent être réutilisées.

4.2 IBB-GBJ

Six tableaux sont utilisés dans l'algorithme `IBB-GBJ` présenté sur la figure ci dessous :

- `solutions[i,j]` contient la $j^{\text{ème}}$ solution du bloc i.
- `blocks_back[i]` contient l'ensemble de blocs ancêtres du bloc i. Le bloc le plus récent parmi eux (i.e., celui avec le plus grand numéro) est choisi en cas de retour-arrière.
- `parents[i]` contient l'ensemble des blocs parents du bloc i.
- `assignment[v]` contient la valeur courante de la variable v.
- `save_parents[i]` contient les valeurs des variables des blocs parents de i la dernière fois que i a été résolu. Ce tableau est seulement utilisé quand la condition de recalcul est testée.
- `#sols[i]` contient le nombre de solutions du bloc i.

```
Algorithme IBB_GBJ (#blocks, solutions, parents, save_parents, assignment)

  for i = 1 to #blocks  do
     blocks_back[i] = parents[i]
     sol_index[i] = 0
     #sols[i] = 0
  end_for
  i = 1
  while (i >= 1) do
      if (Parents_changed? (i, parents, save_parents, assignment)) then
           update_save_parents (i, parents, save_parents, assignment)
           sol_index[i] = 0
           #sols[i] = 0
      end_if

      if (sol_index[i] >= #sols[i])    and
           not (next_solution(i, solutions, #sols)))
      then
          i = backjumping (i, blocks_back, sol_index)
      else  /* les solutions [i, sol_index[i] ] sont affectées au bloc i */
          assign_block (i, solutions, sol_index, assignment)
          sol_index[i] = sol_index[i] + 1
          if (i == #blocks)  then  /* solution globale trouvée*/
             store_total_solution (solutions, sol_index, i)
             blocks_back[#blocks] = {1...#blocks-1}
          else
             i = i + 1
          end_if
       end_if
   end_while
```

IBB-GBJ peut trouver toutes les solutions d'un CSP numérique. A partir du DAG, les blocs sont tout d'abord ordonnés dans un ordre total et numérotés de 1 à #blocks. Après une phase d'initialisation, la boucle while correspond à la recherche de solutions, i étant le bloc courant. Le processus se termine quand $i = 0$, ce qui signifie que toutes les solutions ont été trouvées.

La fonction next_solution appelle le résolveur pour calculer la solution suivante du bloc i. Si une solution est trouvée, la fonction retourne *vrai*, et les tableaux solutions et #sols sont mis à jour. Sinon, la fonction retourne *faux*.

Le code correspondant au premier else contient les actions à réaliser quand une solution d'un bloc est choisie. La procédure assign_block modifie le tableau assignment de telle sorte que les valeurs de la solution trouvée sont affectées aux variables du bloc i. Quand une solution totale a été trouvée, la mise à jour de blocks_back est standard et assure la complétude [Dechter, 1990]. Parents_changed? vérifie la condition de recalcul.

Quand un bloc n'a pas de solution, une fonction standard backjumping retourne un niveau j où il est possible de revenir sans perdre de solution. Il est important d'ajouter dans les causes d'échec du bloc j (i.e., blocks_back[j]) celles du bloc i. En effet, ces blocs sont une cause d'erreur possible pour la valeur courante du bloc j (comme dans GBJ).

```
Fonction backjumping (i, in-out blocks_back, in-out sol_index)

 if blocks-back[i] then
     j = more_recent (blocks_back[i])
     blocks_back[j] = blocks_back[j] U blocks_back[i] \ {j}
 else
     j = -1
 end_if

 for k = j+1 to i do
    blocks_back[k] = parents[k]
    sol_index[k] = 0
 end_for

 return j
```

Favoriser la valeur courante

Le principal inconvénient des algorithmes basés sur le "backjumping" est que le travail réalisé dans les blocs entre i et j est perdu. Quand ces blocs sont traités de nouveau, on choisit d'abord la valeur courante d'une variable, au lieu de retraverser le domaine depuis le début. Puisque les domaines sont dynamiques avec IBB (les solutions d'un bloc changent quand de nouvelles valeurs d'entrée lui sont données), cette amélioration ne peut être réalisée que si la condition de recalcul permet de réutiliser les solutions antérieures.

Cette heuristique a été ajoutée à IBB-GBJ [2]. Cependant, probablement à cause de la

2. L' algorithme doit gérer un autre indice en plus de sol_index.

remarque ci-dessus, les gains en performance obtenus par cette heuristique sont faibles et ne sont pas détaillés dans les expérimentations (cf partie 5).

4.3 Filtrage inter-blocs

Contrairement aux caractéristiques du retour-arrière, le filtrage inter-blocs (*ibf*) est spécifique aux techniques de résolution par intervalles. *ibf* peut être incorporé à toute variante de `IBB`.

Pour les CSP en domaines finis, on a généralement observé que, pendant la résolution, un filtrage puissant sur le problème restant est intéressant. C'est pourquoi nous avons décidé d'installer un filtrage inter-blocs dans `IBB`: au lieu de limiter le processus de filtrage (basé sur les consistances `2B`, `3B`, `Box` ou `Bound` dans notre outil) au bloc courant, nous avons étendu le champ du filtrage à toutes les variables.

Plus précisément, avant de résoudre un bloc i, on forme un sous-système de variables et équations extrait des blocs suivants:

1. soit $B = \{i...\#blocks - 1\}$ l'ensemble des blocs non encore résolus,
2. on réduit B aux blocs connexes à i dans le graphe réduit aux blocs de B[3].

Ensuite, la bissection est appliquée seulement sur le bloc i alors que le processus de filtrage est appliqué sur toutes les variables des blocs dans B.

Pour illustrer *ibf*, considérons le `DAG` de l'exemple didactique. Quand le bloc 1 est résolu, tous les blocs sont considérés par *ibf* puisqu'ils sont tous connexes au bloc 1. Alors, toute réduction d'intervalle dans le bloc 1 peut impliquer une réduction pour toute variable du système. Quand le bloc 2 est résolu, une réduction peut avoir une influence sur les blocs 3, 4 ou 5 pour les mêmes raisons. (On notera que le bloc 3 n'est pas en aval du bloc 2.) Quand le bloc 3 est résolu, une réduction ne peut avoir une influence que sur le bloc 5. En effet, après avoir enlevé du graphe les blocs 1 et 2, les blocs 3 et 4 n'appartiennent plus à la même composante connexe. En fait, aucune propagation ne peut atteindre le bloc 4 puisque les variables parents du bloc 4 qui sont dans le bloc 2 ont un intervalle déjà réduit à une largeur d'au plus w_1 et ne peuvent plus être réduits.

Remarque

Il faut faire attention à la façon dont *ibf* est incorporé dans `IBB-GBJ`. En effet, les réductions induites par les blocs précédents doivent être regardées comme des causes d'échec possibles. Cette modification de l'algorithme n'est pas détaillée et nous illustrons juste ce point sur l'exemple didactique. Si aucune solution n'est trouvée dans le bloc 3, `IBB` avec *ibf* doit revenir sur le bloc 2 et non sur le bloc 1. En effet, quand le bloc 2 a été résolu, une réduction pourrait avoir été propagée sur le bloc 3 (à travers 5).

5 Expérimentations

Des expérimentations exhaustives ont été menées sur 8 problèmes composés de contraintes géométriques. Nous avons comparé différentes variantes de `IBB` avec une résolution par intervalles appliquée sur le système entier (appelée *résolution globale* ci-après).

3. L'orientation du `DAG` est oubliée à cette étape, les arcs du `DAG` sont transformées en arêtes, et le filtrage peut s'appliquer sur des blocs frères.

5.1 Banc d'essais

Certains d'entre eux sont des problèmes artificiels, à base surtout de contraintes quadratiques de distance. `Mechanism` et `Tangent` sont issus de [Latham et Middleditch, 1996] et [Bouma *et al.*, 1995]. `Chair` est un assemblage réaliste d'une chaise fait de 178 équations représentant une grande variété de contraintes géométriques: distance, angle, incidence, parallélisme, orthogonalité, etc.

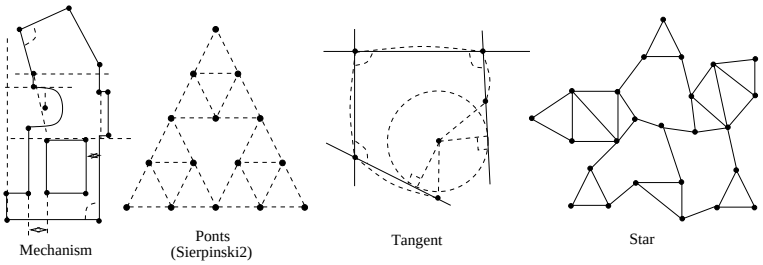

FIG. 2 – *Banc d'essais 2D*

Dim	GCSP	Dec.	Taille	Taille Dec.	t.p	pet.	moy.	gr.
2D	Mechanism	equ.	98	98 = 1x10, 2x4, 27x2, 26x1	1	8	48	448
	Ponts	equ.	30	30 = 1x14, 6x2, 4x1	1	15	96	128
	Sierpinski3	geo.	84	124 = 44x2, 36x1	1	8	96	138
	Tangent	geo.	28	42 = 2x4, 11x2, 12x1	4	16	32	64
	Star	equ.	46	46 = 3x6, 3x4, 8x2	1	4	8	8
3D	Chair	equ.	178	178 = 1x15,1x13,1x9,5x8,3x6, 2x4,14x3,1x2,31x1	6	6	18	36
	Hourglass	geo.	39	39 = 2x4, 3x3, 2x2, 18x1	1	1	2	8
	Tetra	equ.	30	30 = 1x9, 4x3, 1x2, 7x1	1	16	68	256

TAB. 1 – *Détails du banc d'essais : méthode de décomposition (Dec.); nombre d'équations (Taille); Taille des blocs (Taille Dec.)- $N \times K$ signifie N blocs de taille K ; nombre de solutions avec les quatre types de domaines sélectionnés : très petits (largeur = 0.1), petits (1), moyens (10), grands (100).*

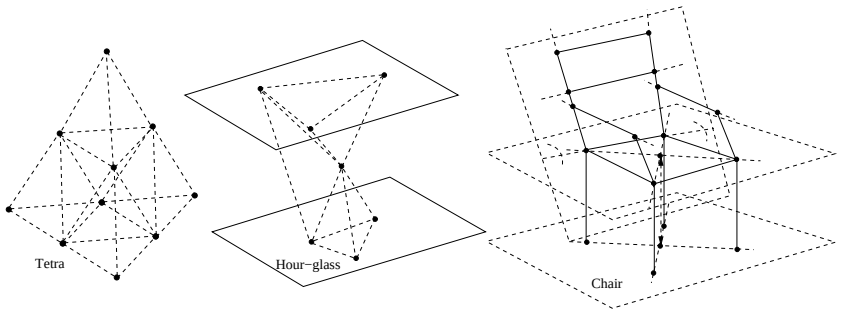

FIG. 3 – *Banc d'essais 3D*

Les domaines ont été choisis autour d'une solution donnée et conduisent à des espaces de recherche radicalement différents. On notera qu'un problème défini avec de grands

domaines revient généralement à mettre $]-\infty,+\infty[$ comme domaine de chaque variable.

`Sierpinski3` est la fractale de `Sierpinski` au niveau 3, c-à-d. 3 `Sierpinski2` mis ensemble. Le système d'équations correspondant aurait environ 2^{40} solutions, on a donc limité les domaines initiaux à des largeurs de 0.1 (très petit), 0.8 (petit), 0.9 (moyen), 1 (grand).

5.2 Choix du filtrage

Nous avons sélectionné les meilleurs algorithmes de filtrage en réalisant des tests sur deux problèmes de taille moyenne. Plusieurs précisions ont été essayées pour les paramètres w_1 et w_2.

Il est apparu clairement (cf TAB. 2) que `2B+Box` et `3B` surpassent les autres filtrages. Tous les tests suivants ont été réalisés avec ces deux techniques.

	w_2	w_1	`2B/3B`	`Box/Bound`	`2B+Box/3B+Bound`
Ponts	0	1e-6	sing	264	**29**
		1e-8	sing	292	**32**
		1e-10	sing	278	**32**
	1e-2	1e-6	116	2078	309
		1e-8	2712	2642	1303
		1e-10	13565	2652	5570
	1e-4	1e-6	84	>54000	523
		1e-8	4413	>54000	5274
Tangent	0	1e-6	sing	547	81
		1e-8	sing	553	82
		1e-10	sing	562	86
	1e-2	1e-6	**26**	265	91
		1e-8	**35**	270	94
		1e-10	**60**	266	93
	1e-4	1e-6	51	2516	369
		1e-8	68	2535	393

TAB. 2 – *Comparaison de différentes consistances partielles. Les meilleurs résultats sont en gras. Un 0 dans la colonne w_2 signifie que* `2B`, `Box`, *ou* `2B+Box` *sont utilisées. Les cases* `sing` *correspondent à des solutions multiples qui conduisent à une explosion combinatoire (cf 6.2).*

5.3 Tests principaux

Les principales conclusions des tests, dont les résultats sont présentés sur le tableau 3, sont les suivantes :

- `IBB` surpasse toujours la résolution globale, ce qui montre l'intérêt d'exploiter la structure du problème. Un, deux ou trois ordres de grandeur peuvent être gagnés en performance. Même avec les très petits domaines, les gains peuvent être significatifs (voir `Sierpinski3`)[4].

4. La résolution globale se compare avantageusement avec `IBB` sur le problème `Star` avec de très petits domaines. Ceci est dû à la plus grande précision requise pour rendre `IBB` complet (voir partie 6). Avec la même précision, la résolution globale met 75 s à trouver les solutions.

- Le filtrage inter-blocs est toujours contre-productif et même parfois très mauvais (cf `Tangent`). Les essais avec la consistance `3B` montrent que la perte de temps due au filtrage inter-blocs est alors réduite.
- L'exploitation de la structure du `DAG` par la condition de recalcul est très bénéfique.

		Très petit		Petit		Moyen		Grand	
		¬IBF	IBF	¬IBF	IBF	¬IBF	IBF	¬IBF	IBF
Chair	Global	XXS		XXS		XXS		XXS	
	BT	3.3	XXS	3.2	XXS	9.4	XXS	12.4	XXS
	BT+	2.4	XXS	2.3	XXS	4.5	XXS	4.7	XXS
	GBJ	2.4	XXS	2.3	XXS	4.5	XXS	4.7	XXS
Mechanism	Global	XXS		XXS		XXS		XXS	
	BT	0.17	14.1	0.6	15.0	2.8	18.7	13.3	32.8
	BT+	0.11	14.1	0.4	13.6	2.6	17.2	13.1	30.6
	GBJ	0.10	14.1	0.4	13.5	2.6	17.3	13.1	30.4
	GPB	0.10	14.2	0.4	13.3	2.7	17.4	13.1	30.5
	3B(GBJ)	0.23	0.68	1.7	2.3	9.7	11	83	88
Ponts	Global	0.73		32		82		110	
	BT	0.16	0.63	2.38	4.2	6.5	10.6	9.1	14.6
	BT+	0.16	0.63	2.36	4.2	6.1	10.2	8.8	14.7
	GBJ	0.17	0.58	2.35	4.1	6.0	10.4	8.7	14.4
	GPB	0.22	0.61	2.37	4.1	6.3	10.4	8.7	14.4
	3B(GBJ)	0.3	0.6	12	15	25	31	49	59
Hour-glass	Global	0.12		1.89		1.47		22.77	
	BT+	0.03	0.88	0.03	1.64	0.06	1.00	0.06	1.21
	GBJ	0.04	0.75	0.03	1.60	0.02	0.83	0.06	1.19
	GPB	0.05	0.73	0.03	1.61	0.05	0.88	0.05	1.15
	3B(GBJ)	0.03	0.3	0.05	0.6	0.05	0.2	0.1	0.4
Sierpinski3	Global	3.1		>54000		>54000		>54000	
	3B(BT)	0.1	1.32	12.3	160	96	788	136	1094
	3B(BT+)	0.1	1.32	12.7	160	67	703	93	928
	3B(GBJ)	0.1	1.32	12	166	61	682	85	916
Tangent	Global	0.5		35		39		46	
	BT+	0.05	1.26	0.11	1.89	0.13	7.63	0.20	8.15
	GBJ	0.07	1.17	0.11	1.89	0.14	7.69	0.19	8.00
	GPB	0.07	1.19	0.10	1.93	0.11	7.69	0.22	8.04
	3B(GBJ)	0.2	0.7	0.2	1.3	0.2	1.3	0.3	1.7
Tetra	Global	2.15		92		197		406	
	BT+	0.14	0.74	1.08	4.00	2.37	7.01	4.73	13.56
	GBJ	0.14	0.67	1.10	3.87	2.30	6.80	4.74	13.20
	GPB	0.16	0.65	1.11	3.90	2.29	6.71	4.72	13.19
Star	Global	8.7		2908		2068		1987	
	BT	9.96	70	40.2	137	81.4	241	80.3	240
	BT+	9.96	70	29.5	99.6	78.1	102	78	102
	GBJ	9.96	70	29.1	99.6	77.9	102	77.9	102
	GPB	9.96	70	29.4	99.6	49.3	102	49	102

TAB. 3 – *Résultats expérimentaux.* `BT+` *est* `IBB-BT` *avec la condition de recalcul. Pour chaque algorithme et chaque taille de domaine, les temps sont donnés avec (IBF) et sans (¬IBF) le filtrage inter-blocs ibf. Tous les temps sont en secondes et ont été obtenus sur un* `PentiumIII 935 Mhz` *sous Linux. Les temps présentés ont été obtenus avec* `2B+Box`*, souvent meilleurs qu'avec* `3B` *sauf pour Sierpinski3. Les lignes* `3B(GBJ)` *montrent les temps avec* `IBB-GBJ` *et* `3B` *quand ils sont compétitifs.*

Les cases dans le tableau 3 contenant XXS correspondent à un échec dans la résolution dû à `IlogSolver` où une taille maximum est dépassée.

Pour affiner nos conclusions, le tableau 4 donne les statistiques obtenues sur le nombre de sauts (backjumps) réalisés par `IBB-GBJ`. On notera qu'aucun saut n'a été observé avec les quatre autres problèmes.

Ces expérimentations montrent un résultat significatif. La plupart des sauts disparaissent quand on utilise un filtrage inter-blocs, ce qui rappelle des résultats similaires obtenus en domaines finis avec MAC-CBJ [Bessière et Régin, 1996]. Cependant, le prix à payer pour ce filtrage inter-blocs est dans ces essais trop important pour être rentable. `Sierpinski3`-Grand montre cette tendance : 2114 des 2118 sauts sont éliminés par le filtrage inter-blocs, mais l'algorithme correspondant est 10 fois plus lent que `IBB-GBJ`!

	Filtrage IB	T.P.	Petit	Moyen	Grand
Ponts	non	0	0	1	0
	oui	0	0	0	0
Mechanism	non	3	4	0	0
	oui	0	0	0	0
Star	non	0	2	6	6
	oui	0	0	0	0
Sierpinski3	non	0	12	829	2118
	oui	0	0	5	4

TAB. 4 – *Nombre de sauts avec et sans filtrage inter-blocs*

6 Discussion

Deux difficultés sont apparues dans l'utilisation des techniques de résolution par intervalles avec `IBB`. Elles sont détaillées ci-dessous.

6.1 Heuristique du point milieu

Cette heuristique (cf partie 3.2) n'est pas satisfaisante puisque des solutions peuvent être perdues, rendant le processus de recherche incomplet. Quand nous l'avons introduite [Bliek *et al.*, 1998], les exemples étaient petits et ce cas ne s'est pas produit. Depuis, il est apparu avec les exemples `Star`, `Sierpinski3` et `Chair`. Le problème a été résolu avec `Star` et `Sierpinski3` en augmentant la précision (c.-à-d, en diminuant w_1). Des modifications ad hoc du système d'équations ont dû être effectuées pour résoudre le problème sur l'exemple `Chair`.

Une solution robuste consisterait à introduire des intervalles constants dans les équations au lieu des points milieux (ce qui n'est actuellement pas possible avec `IlogSolver`). Nous pensons que le surcoût en temps devrait être faible dans le schéma de résolution par filtrage et bissection. Par contre, les tests d'unicité actuels ne pourraient plus être réalisés car ils ont besoin d'un système carré.

6.2 Gérer les solutions multiples

Une autre limite des techniques d'intervalles est aggravée par `IBB`. Les *solutions multiples* se produisent quand plusieurs boîtes atomiques proches sont trouvées : une seule contient une solution et le filtrage n'a pu écarter les autres. Même quand le nombre de solutions multiples est petit, l'effet multiplicatif dû à la combinaison des solutions partielles par `IBB` peut rendre le problème impraticable.

Une solution consiste à augmenter la précision (i.e., réduire w_1), ce qui résout certains cas. Mixer plusieurs techniques de filtrage, comme `2B+Box`, réduit aussi le phénomène.

(Les cases *sing* dans le tableau 2 avec `2B` proviennent de ce phénomène.) Nous avons implanté une méthode pour détecter les solutions multiples en ne gardant que la première des solutions trouvées à une précision epsilon près. Ceci a résolu le problème dans la plupart des cas. Mais quelques cas pathologiques restent, à cause d'une interaction avec l'heuristique du point milieu. Le point milieu du premier intervalle atomique trouvé peut ne pas être une solution et empêcher la résolution des blocs en aval. Il serait sans doute plus robuste de prendre l'union des solutions multiples.

7 Conclusion

Cet article a détaillé le cadre générique du retour arrière inter-blocs `IBB` pour résoudre les CSP numériques. Nous avons implanté trois schémas de retour arrière (chronologique `BT`, basé sur le graphe de blocs `GBJ` et de type *partial order backtracking* `GPB`). Chaque schéma peut incorporer une condition de recalcul qui évite certains appels inutiles au résolveur. Chaque schéma peut utiliser ou non un filtrage inter-blocs.

Des séries de tests expérimentaux ont été réalisées sur un banc d'essais de taille raisonnable et contenant des équations non linéaires. En voici les principaux résultats. D'abord, toutes les variantes de `IBB` peuvent gagner plusieurs ordres de grandeur par rapport à une résolution globale. Ensuite, le choix du schéma de retour-arrière n'est pas apparu crucial. Néanmoins, exploiter la structure du graphe de blocs pendant le retour-arrière peut améliorer la performance sans créer de surcoût. Ceci nous conduit à proposer l'utilisation de la version `IBB-GBJ` présentée dans cet article.

Un autre résultat de cet article est que le filtrage inter-blocs est apparu contre-productif. Cela tend à montrer qu'un filtrage global qui ne tient pas compte de la structure accomplit beaucoup de travail inutile. La prochaine étape de notre travail est la mise en œuvre d'une résolution avec des intervalles constants pour enlever l'heuristique du point milieu et rendre notre implantation plus robuste.

Références

[Ait-Aoudia *et al.*, 1993] S. Ait-Aoudia, R. Jegou, et D. Michelucci. Reduction of constraint systems. Dans *Compugraphic*, 1993.

[Bessière et Régin, 1996] C. Bessière et J.C. Régin. Mac and combined heuristics: two reasons to forsake fc (and cbj?) on hard problems. Dans *Proceedings CP'96*, volume 1118 of *LNCS*, pages 61–75, Cambridge MA,USA, 1996. Springer.

[Bliek *et al.*, 1998] C. Bliek, B. Neveu, et G. Trombettoni. Using graph decomposition for solving continuous csps. Dans *Principles and Practice of Constraint Programming, CP'98*, volume 1520 of *LNCS*, pages 102–116, Pise, Italie, 1998. Springer.

[Bliek, 1998] Christian Bliek. Generalizing partial order and dynamic backtracking. Dans *Fifth National Conference on Artificial Intelligence – AAAI'98*, 1998.

[Bondyfalat *et al.*, 1999] D. Bondyfalat, B. Mourrain, et T. Papadopoulo. An application of automatic theorem proving in computer vision. Dans *2nd International Workshop on Automated Deduction in Geometry, Springer-Verlag*, 1999.

[Bouma *et al.*, 1995] W. Bouma, I. Fudos, C.M. Hoffmann, J. Cai, et R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.

[Buchberger, 1985] B. Buchberger. An algebraic method in ideal theory. Dans *Multidimensional System Theory*, pages 184–232. Reidel Publishing Co., 1985.

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[Durand, 1998] C. Durand. *Symbolic and Numerical Techniques For Constraint Solving.* PhD thesis, Purdue University, 1998.

[Hendrickson, 1992] Bruce Hendrickson. Conditions for unique realizations. *SIAM j Computing*, 21(1):65–84, 1992.

[Hoffmann *et al.*, 1997] C.M. Hoffmann, A. Lomonosov, et M. Sitharam. Finding solvable subsets of constraint graphs. Dans *Principles and Practice of Constraint Programming CP'97*, pages 463–477, 1997.

[Jaulin *et al.*, 2001] L. Jaulin, M. Kieffer, O. Didrit, et E. Walter. *Applied Interval Analysis.* Springer-Verlag, 2001.

[Jermann *et al.*, 2000] C. Jermann, G. Trombettoni, B. Neveu, et M. Rueher. A constraint programming approach for solving rigid geometric systems. Dans *Principles and Practice of Constraint Programming, CP'2000*, volume 1894 of *LNCS*, pages 119–134, Singapour, 2000. Springer.

[Lahaye, 1934] E. Lahaye. Une méthode de résolution d'une catégorie d'équations transcendantes. *Compte-rendu des Séances de L'Académie des Sciences*, 198:1840–1842, 1934.

[Latham et Middleditch, 1996] R.S. Latham et A.E. Middleditch. Connectivity analysis: A tool for processing geometric constraints. *Computer Aided Design*, 28(11):917–928, 1996.

[Lhomme, 1993] O. Lhomme. Consistency techniques for numeric CSPs. Dans *IJCAI'93: Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 232–238, 1993.

[McAllester, 1993] D.A. McAllester. Partial order backtracking. Research Note, Artificial Intelligence Laboratory, MIT, 1993. ftp://ftp.ai.mit.edu/people/dam/dynamic.ps.

[Merlet, 2002] Jean-Pierre Merlet. Optimal design for the micro robot. Dans *in IEEE Int. Conf. on Robotics and Automation*, 2002.

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

[Wilczkowiak *et al.*, 2003] Marta Wilczkowiak, Gilles Trombettoni, Christophe Jermann, Peter Sturm, et Edmond Boyer. Scene Modeling Based on Constraint System Decomposition Techniques. Dans *Proc. International Conference on Computer Vision, ICCV'03*, pages 1004–1010, 2003.

[Wu, 1986] W. Wu. Basic principles of mechanical theorem proving in elementary geometries. *J. Automated Reasoning*, 2:221–254, 1986.