HD_DBT : Hypertree Décomposition pour la résolution des problèmes de satisfaction de contraintes basée sur un Dual Backtracking

Kamal Amroun[1] **Zineb Habbas** [2]

[1] Département d'Informatique, Université de Béjaïa, Algérie ,
[2] Laboratoire L.I.T.A, Université de Metz, France
k_amroun25@yahoo.fr zineb@univ-metz.fr

Résumé

L'hypertree decomposition généralisée est l'approche la plus générale connue dans la littérature pour identifier des sous classes traitables de problèmes de satisfaction de contraintes (CSPs) n-aires représentés à l'aide d'hypergraphes. Seulement, quoi que sa complexité théorique est bornée par la largeur de la décomposition, la méthode proposée pour la résolution du CSP donné sous forme d'hypertree n'est pas efficace en pratique. Dans ce papier, nous proposons la méthode HD_DBT (pour résolution des CSP par un algorithme de type BT sur les tuples guidé par un ordre statique induit par une Hypertree Décomposition) comme une nouvelle approche de résolution des CSPs préalablement décomposés en hypertree généralisée. Nous avons implémenté et expérimenté cette approche. Les résultats de comparaison par rapport à la méthode de résolution des CSPs par hypertree decomposition, connue dans la littérature sont encourageants.

1 Introduction

Les problèmes de satisfaction de contraintes (CSPs) sont au coeur de problèmes fondamentaux de l'intelligence artificielle car le formalisme CSP est très puissant et permet de représenter de nombreux problèmes du monde réel. Un CSP est défini comme un ensemble de contraintes impliquant un ensemble de variables. L'objectif est de trouver une instanciation possible pour chaque variable qui satisfait toutes les contraintes. Le formalisme CSP a été introduit par Montanari en 1974 [15]. Depuis, de nombreuses méthodes ont été proposées pour résoudre les CSPs. La méthode naïve de résolution de CSP est le backtracking dont la complexité théorique est exponentielle en $O(md^n)$ où n est le nombre de variables du problème, m est le nombre de contraintes et d est la taille maximale des domaines des variables. Pour réduire cette complexité, beaucoup de travaux ont été menés en exploitant certaines propriétés des instances et plusieurs approches ont été proposées. Parmi ces méthodes, il y a les méthodes de décomposition structurelles. Les méthodes de décomposition structurelles ont toutes pour but de transformer un CSP en un CSP équivalent (ayant les mêmes solutions) en formant des clusters de variables ou de contraintes dont l'interaction a une structure d'arbre. Chaque méthode de décomposition définit son propre concept de largeur : elle peut être interprétée comme une mesure de cyclicité du graphe ou de l'hypergraphe de contraintes. Parmi ces méthodes, on retient principalement la méthode coupe cycle [4], la méthode hypercutset [8], la méthode BICOMP [7], La méthode de tree-clustering [5]. Dans ce papier nous nous intéressons à la méthode la plus générale connue sous le nom d'hypertree-decomposition généralisée [8]. (On ne considère pas la méthode : fractionnal hypertree decomposition due à Grohe et al [12] dans cette comparaison). L'hypertree decomposition généralisée permet d'obtenir des décompositions de meilleures largeurs. Une méthode de décomposition en hypertree généralisée a pour but de transformer l'hypergraphe associé à un CSP en une structure hyper-arborescente dont les noeuds sont des clusters d'hyper-arêtes et des clusters de variables soumis à certaines contraintes dont la résolution se fait en un temps polynomial en la largeur de cette hyper-arborescence (Hypertreewidth). Cependant, la méthode proposée pour la résolution du CSP préalablement décomposé en hypertree décomposition n'est

pas efficace voir inopérante en pratique. Le problème de cette approche vient du fait que la méthode de résolution résout d'abord les sous problèmes au noeuds avant de résoudre le problème tout entier. Pour cela il faut rechercher au niveau des noeuds toutes les solutions et les sauvegarder puis résoudre le problème tout entier par des opérations de semi-jointure. Cette méthode entraîne inévitablement un problème d'explosion mémoire et se trouve limitée à la résolution de problèmes de petite taille. Pour pallier à ce problème, dans ce papier, nous allons présenter une nouvelle méthode de résolution des CSPs dite HD_DBT (pour résolution des CSP par un algorithme de type BT sur les tuples guidé par un ordre statique induit par une Hypertree Décomposition) qui parcourt l'hypertree decomposition de la racine aux feuilles en recherchant au niveau de chaque noeud un seul tuple solution qui soit compatible avec les tuples déjà calculés dans les noeuds précédents et aussi compatible avec l'ensemble des contraintes de ses noeuds fils. Si aucun tuple n'est possible, l'algorithme effectue un backtrack chronologique sur le noeud précédent dans l'hypertree pour calculer un autre tuple et ainsi de suite jusqu'à ce qu'une solution soit trouvée ou alors remonter jusqu'à la racine, auquel cas le CSP n'a pas de solution si tous les tuples possibles sont explorés. Nous allons présenter dans ce papier cette approche et allons montrer par nos premières expérimentations son intérêt.

Ce papier est organisé comme suit : dans la section 2 nous rappelons la définition formelle d'un CSP, les différentes notions de décompositions structurelles et plus particulièrement la méthode qui généralise toutes les autres méthodes : l'hypertree decomposition généralisée (on ne tient pas compte de la méthode fractional hypertree decomposition de Grohe et al [12]). Dans la section 3, nous rappelons l'approche proposée par Gottlob et al [10] pour la résolution du CSP retourné par la décomposition en hypertree généralisée. Nous présenterons les détails de notre méthode de résolution HD_DBT dans la section 4. Les résultats expérimentaux sont présentés dans la section 5 et enfin la section 6 sera consacrée à la conclusion et aux perspectives.

2 Préliminaires

La notion de problèmes de satisfaction de contraintes (CSPs pour Constraint Satisfaction Problems) a été introduite dans [15] selon la définition 1.

Définition 1 . *Un Problème de Satisfaction de Contraintes est un quadruplet* $P = (X, D, C, R)$, *où* $X = \{x_1, x_2, ..., x_n\}$ *est un ensemble de n variables,* $D = \{d_1, d_2, ..., d_n\}$ *est un ensemble de n domaines finis. Chaque domaine* d_i *est associé à une variable* x_i*. C est un ensemble de m contraintes. Chaque contrainte* C_i *est définie par un ensemble de r variables* $X_i = \{x_{i1}, x_{i2}, ..., x_{ir}\}$*.* $R = \{R_1, R_2, ..., R_m\}$ *est un ensemble de m relations. Chaque relation* R_i *définit l'ensemble des tuples autorisés par la contrainte* C_i*.*

Définition 2 *: Hypergraphe de contraintes*
Un hypergraphe de contraintes est un couple $H =< V, E >$ *où V est l'ensemble des sommets et E est l'ensemble des hyperarêtes. Chaque sommet correspond à une variable du CSP et chaque hyperarête correspond à une contrainte. On note souvent V par var(H) et E par hyperedges(H) ou hyperarêtes(H). Aussi,dans ce papier, si h est une hyperarête,* $var(h)$ *indique l'ensemble des variables de h ; si S est un ensemble d'hyperaretes, alors* $var(S)$ *correspond à l'ensemble des variables apparaissant dans les hyperarêtes de S.*

Un *hypertree* pour un hypergraphe H est un triplet $< T, \chi, \lambda >$ où $< T = (N, E) >$ est un arbre enraciné et χ et λ sont deux fonctions d'étiquetage associant à chaque sommet (noeud) p de N un ensemble de variables χ(p) et un ensemble de contraintes λ(p). On note aussi *sommets(T)* l'ensemble des sommets de T et on note la racine de T par *root(T)*. T_p indique l'ensemble des variables du sous arbre qui a pour racine le noeud p.

Proposition 1. [3] : La classe des CSP dont le graphe de contraintes est acyclique est traitable, et ceci indépendamment des relations des contraintes.

Définition 3 *(* décomposition en hypertree généralisée *[10])*
Une décomposition hypertree généralisée d'un hypergraphe $\mathcal{H} =< V, E >$ *est un hypertree* $HD =< T, \chi, \lambda >$ *qui satisfait les conditions suivantes :*

1. *Pour chaque hyperarete* $h \in E$*, il existe* $p \in vertices(T)$ *telle que* $var(h) \subseteq \chi(p)$*. On dit que p couvre h.*
2. *Pour chaque variable* $v \in V$*, l'ensemble* $\{p \in vertices(T) | v \in \chi(p)\}$ *induit un sous arbre connexe de T.*
3. *Pour chaque sommet* $p \in vertices(T)$*,* $\chi(p) \subseteq var(\lambda(p))$

La figure 1 montre l'exemple d'un hypergraphe et son hypertree décomposition généralisée.

L'hypertree decomposition d'un hypergraphe $\mathcal{H} =< V, E >$, est une hypertree decomposition généralisée $HD =< T, \chi, \lambda >$ qui satisfait la condition

supplémentaire suivante :
Pour chaque sommet $p \in vertices(T)$, $var(\lambda(p)) \bigcap \chi(T_p) \subseteq \chi(p)$.

La largeur d'une hypertree décomposition généralisée $< T, \chi, \lambda >$ est $max_{p \in vertices(T)} |\lambda(p)|$. La largeur d'une décomposition en hypertree (généralisée) $(g)hw(\mathcal{H})$ d'un hypergraphe $\mathcal{H}$ est la largeur minimum de toutes ses décompositions en hypertree (généralisée).

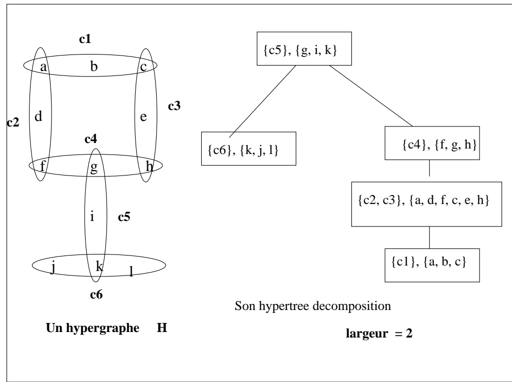

FIG. 1 – Un hypergraphe H et son hypertree decomposition généralisée

Définition 4 *Une hyperarete h est fortement couverte dans un hypertree s'il existe un noeud p tel que $var(h) \subseteq \chi(p)$ et $h \in \lambda(p)$.*

Définition 5 *Une hypertree décomposition $< T, \chi, \lambda >$ d'un hypergraphe $\mathcal{H} =< V, E >$ est* ***complète*** *si chaque hyperarête h de $\mathcal{H} =< V, E >$ est fortement couverte dans $HD =< T, \chi, \lambda >$.*

Calculer une hypertree decomposition généralisée optimale est un problème NP difficile [11]. Pour cette dernière, il existe deux approches de calcul : les méthodes exactes et les heuristiques. Parmi les méthodes exactes, nous citons opt-k-decomp [9] qui permet de calculer une décomposition de largeur optimale bornée par une constante k fixée si une telle décomposition existe. Cependant, ces algorithmes dits exacts ne sont efficaces que pour les problèmes de petites tailles. Pour cela, on a vu ces dernières années le développement des heuristiques pour le calcul de cette décomposition. Dans [6], les auteurs ont proposé les heuristiques BE et DBE et dans [16], Musliu et Schafhauser ont exploré les algorithmes génétiques pour calculer une hypertree decomposition généralisée. Dans [2] nous avons proposé des heuristiques comparables à BE.

3 Résolution des CSPs par la méthode hypertree généralisée

Cette approche nécessite d'abord le calcul d'une décomposition par un algorithme exact ou une heuristique. (En pratique, les algorithmes exacts ne sont pas efficaces). Une fois que la décomposition est obtenue, on la complète de telle sorte que toutes les contraintes figurent dans au moins un noeud de l'hypertree.

Ensuite l'approche proposée par Gottlob et al [10] pour résoudre le CSP obtenu est donnée par l'algorithme 1.

```
Input: Une hypertree decomposition
       HD =< T, χ, λ > associée à un CSP
       donné.
Output: Une solution A
begin
   σ = {n1, n2, ..., nm} un ordre sur les noeuds
   de l'hypertree decomposition où n1 est la
   racine et chaque noeud précède ses fils dans
   l'ordre. ;
   foreach p nœud de l'hypertree do
      Rp = join(λ(p))[χ(p)] ;
   end
   for i = m to 2 do
      begin
         Soit vj le père de vi dans l'ordre ;
         Rj = semijoin(Rj, Ri) ;
      end
   end
   for i = 2 to m do
      Construire une solution A en sélectionnant
      un tuple dans Ri compatible avec tous
      ceux qui le précèdent.
   end
   return A ;
end
```

Algorithm 1: Méthode de résolution proposée par Gottlob

En pratique, cette méthode est très coûteuse aussi bien en temps qu'en espace mémoire. En effet, cette approche est basée sur deux principales opérations qui sont des jointures au niveau des noeuds de l'hypertree et des semi-jointures entre les différents noeuds de l'hypertree. Malgré les différentes heuristiques introduites précédemment dans le rapport de recherche [1], cette approche souffre toujours du problème de l'espace mémoire.

Pour remédier à ce problème d'explosion mémoire et exploiter les atouts des décompositions arborescentes, P. Jégou et C. Terrioux ont proposé dans [14] une

technique intéressante nommée BTD (Résolution d'un CSP par une méthode de type BT guidée par un ordre statique induit par une tree decomposition). BTD hérite à la fois des avantages des méthodes énumératives pour ce qui concerne l'occupation mémoire et des propriétés structurelles du CSP. Dans ce travail, nous allons proposer une autre approche de résolution qui sera la combinaison entre l'approche de base de Gottlob et l'approche de résolution de type retour arrière pour le choix d'un tuple pour un noeud donné. Cette approche est appelée HD_DBT (Hypertree Decomposition versus Dual BackTracking).

4 La méthode HD_DBT

4.1 Description générale

Nous introduisons dans cette section notre approche nommée HD_DBT. HD_DBT consiste à résoudre un CSP donné en passant par la construction de l'hypertree décomposition généralisée associée à son hypergraphe de contraintes. La procédure HD_DBT décrite par l'algorithme2 considère en entrée une hypertree décomposition généralisée et complètée conformément à la définition 5 de l'hypertree décomposition complète et retourne une solution si elle existe.

L'hypertree décomposition est complétée pour s'assurer que pour chaque contrainte c du CSP, il existe un noeud $n =< \lambda_n, \chi_n >$ de l'hypertree $HD =< T,\chi,\lambda >$ tel que les variables de la contrainte c sont contenues dans χ_n et $c \in \lambda_n$. Si ce noeud n'existe pas, on cherche un noeud n' de l'hypertree tel que χ couvre les variables de c puis on crée un noeud n'' fils de n' dont l'ensemble χ est l'ensemble des variables de c et dont le terme λ contient uniquement la contrainte c. Le noeud n' existe forcément parce que c'est l'une des conditions de l'hypertree decomposition généralisée.

La complexité théorique de cet algorithme est en $O(|r|^{w \times Nb_noeuds})$ où $|r|$ est la taille de la plus grande relation, w est l'hypertree-width et Nb_Noeuds est le nombre de noeuds de l'hypertree decomposition.

Pour résoudre le problème représenté sous forme d'hypertree décomposition nous proposons un algorithme de type backtrack. A la différence des algorithmes énumératifs classiques celui-ci instancie un ensemble de variables en une seule étape plutôt que d'instancier variable par variable. Le problème crucial dans la méthode de base de Gottlob est le coût des jointures effectuées aux différents noeuds de l'hypertree décomposition, aussi bien en espace qu'en temps d'exécution. C'est pour celà que nous proposons une approche qui ne calcule pas toutes les solutions au niveau d'un

```
Input: Une hypertree decomposition
       HD =< T,χ,λ > associée à un CSP
       donné.
Output: Une solution A
begin
  CHOIX_RACINE ( HD , racine ) ;
  σ ⟵ ORDRE_INDUIT ( HD , racine ) ;
  nc ⟵ racine ;
  while nc ≠ ∅ do
    begin
      consistant ⟵ FALSE ;
      while ¬ consistant do
        begin
          sol_nc ⟵ resolution (
          λ(nc),χ (nc) ) ;
          if Compatible(sol_nc,
          sol(pere(nc)) then
            begin
              A ⟵ A ∪ {x_i ←
              v_i∀x_i ∈ χ(i)} ;
              consistant ⟵ TRUE ;
            end
        end
      end
      if ¬ Consistant then
        begin
          A ⟵ A − {x_i ← v_i∀x_i ∈ χ(i)}
          ;
          nc ⟵pere(nc) ;
        end
      end
      else
        nc ⟵ [illegible] (nc) ;
      end
    end
  end
  return A ;
end
```

Algorithm 2: Procédure générale HD_DBT

noeud mais ne calcule qu'une seule solution. Si cette solution est consistante avec celle des noeuds déjà résolus on continue sinon on effectue un retour arrière chronologique.

La procédure principale *HD_DBT* considère l'hypertree décomposition obtenue précédemment en entrée et se compose des différentes étapes décrites par l'algorithme 2.

4.2 Description détaillée

Les différentes étapes de *HD_DBT* sont les suivantes :

Etape 1 : La première étape réalisée par la procédure générique **CHOIX_RACINE** est le choix du premier noeud de l'hypertree decomposition à résoudre qu'on appelle racine. Plusieurs heuristiques pour déterminer ce noeud racine sont possibles. Elles peuvent être structurelles ou sémantiques. Nous présenterons certaines heuristiques plus loin.

Etape 2 : La deuxième étape est réalisée par la procédure **ORDRE_INDUIT** qui construit un ordre σ sur les noeuds de l'hypertree décomposition. Cet ordre est obtenu à partir de l'hypertree et de la racine choisie par la procédure **ORDRE_INDUIT** et il découle directement de la propriété de connectivité de l'hypertree decompoisition.

Pour illustrer cette notion d'ordre induit, nous présentons dans la figure 2 deux ordres différents pour l'hypertree decomposition complète associée à l'hypergraphe donnée par la figure 1.

Exemple 1

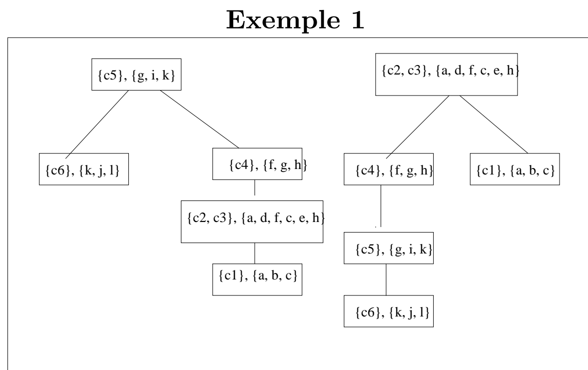

Fig. 2 – Deux ordres différents pour une hypertree décomposition

Etape 3 : Cette étape est la recherche de la solution du CSP par un algorithme énumératif dirigé par l'ordre σ. Pour chaque noeud $\boldsymbol{a} = (\lambda(\boldsymbol{a}), \chi(\boldsymbol{a}))$ de l'ordre σ, *HD_DBT* recherche une solution par la procédure *resolution* . Cette procédure est réalisée par les opérations de jointure et de projection pour trouver une solution (un tuple de jointure). L'algorithme 2 non présenté dans ce papier, pour des raisons de simplicité, est l'algorithme naïf de type BT. Mais cette approche peut se généraliser à tous les autres algorithmes énumératifs tels que FC, Mac, ... Notons toutefois que la connectivité dans un hypertree est liée aux variables et non pas aux contraintes.

4.3 Heuristiques pour le calcul du noeud racine :

Pour améliorer notre approche, nous avons introduit les heuristiques suivantes :

Un noeud dans une hypertree décomposition est composée d'un couple $< \chi\lambda >$. Pour choisir le noeud racine, il est possible de définir des heuristiques en tenant compte de la cardinalité de l'ensemble λ, de celle de l'ensemble χ ou éventuellement des deux à la fois. On distingue deux catégories d'heuristiques : les heursitiques structurelles et les heuristiques sémantiques.

- **Les heuristiques structurelles** : Elle dépendent uniquement des propriétés de la structures de l'hypertree decomposition (taille des clusters, nombre de fils, ...).
  Nous nous proposons à ce titre d'étudier les heuristiques suivantes :

  PGC (Plus Grand Cluster) : La racine correspond au noeud dont la cardinalité de λ est la plus grande

  PPC (Plus Petit Cluster) : La racine correspond au noeud dont la cardinalité de λ est la plus petite

  PNF (Plus Grand nombre de Fils) : La racine correspond au noeud qui possède le plus grand nombre de fils.

  PGS (Plus Grand Séparateur) : La racine correspond au noeud qui partage avec ses fils le plus grand séparateur par rapport aux variables .

- **Les heuristiques sémantiques** : Celles ci dépendent de la nature des données (taille des relations, densité des noeuds, dureté des contraintes, ...).

Nous nous proposons aussi d'étudier les heuristiques suivantes.

NMC (Noeud le moins contraint) : La racine correspond au noeud qui contient le plus grand nombre de solutions attendues. Le nombre de solution attendu est estimé avant l'exécution.

NPC (Noeud le plus contraint) : La racine correspond au noeud qui contient le plus petit nombre de solutions attendues.

NPD (Noeud le plus Dur) : La racine correspond au noeud dont la propriété de dureté est la plus forte. La dureté d'un noeud est égale à $\frac{Nbsol}{NbMax}$ où Nbsol est le nombre de solutions attendu du noeud et NbMax le nombre de tuples maximal (taille du produit cartésien de toutes les relations contenues dans le cluster.

5 Expérimentation et analyse des résultats

Dans cette section nous allons étudier le comportement de notre approche du point de vue expérimental. Les expérimentations ont été menées sur un PC portable HP Compact 6720s, 1,7 GHZ et 2 GO de RAM, sous LINUX Fedora. Les benchmarks sont téléchargés du site http ://www.cril.univ-artois.fr/lecoutre/research/benchmarks.

Pour le calcul des décompositions en hypertree, nous avons exploité l'heuristique BE [6]. Cette heuristique est connue pour donner des décompositions de largeurs proches des largeurs optimales et ce en des temps très courts.

5.1 Comparaison de notre approche à l'approche de Gottlob

Pour montrer l'intérêt de notre approche nous avons commencé par observer son comportement par rapport au comportement de l'algorithme de base proposé par Gottlob et al.

Dans le tableau 1, nous présentons les résultats de cette comparaison en terme de temps d'exécution. Les résultats obtenus par notre approche en temps d'exécution sont meilleurs pour 11 benchmarks sur 12.

Instance	Taille		Temps(s)	
	$\|V\|$	$\|E\|$	HD_DBT	Gottlob
$renault$	101	134	2	3
$series-6-ext$	11	30	0,04	2,18
$series-7-ext$	12	41	0,1	/
$domino-100-100_ext$	100	100	0,12	2,59
$domino-100-200_ext$	100	100	0,30	18,37
$domino-100-300_ext$	100	100	0,4211	60
$hanoi-5-ext$	30	29	0,55	0,88
$hanoi-6-ext$	62	61	120	14
$hanoi-7-ext$	126	124	58	59
$Langford$	8	32	0,20	2,52
$geom-30a-4-ext$	30	81	0,1	0,1
$pigeons-7-ext$	7	21	2	26

Tab. 1 – Comparaison des temps de résolution des deux approches HD_DBT et la méthode de Gottlob.

5.2 Comportement des différentes heuristiques de choix de racine

Dans cette section nous observons l'impact du choix de la racine sur la résolution ainsi que le gain apporté par le filtrage des relations.

Les tableaux 2 et 3 récapitulent les résultats expérimentaux obtenus pour 10 benchmarks en utilisant une implémentation naïve de l'algorithme HD_DBT, sans filtrage et les tableaux 4 et 5 présentent les résultats pour une implémentation avec filtrage des relations . Premièremons nous constatons que sans filtrage cette approche est totalement inefficace. Trois problèmes sur 10 n'ont pu être résolus aprés 20 secondes.

5.2.1 Algorithme naïf et étude des différents ordres

Dans le tableaux 2, nous reportons les résultats obtenus avec les heuristiques portant sur les données (ou les heuristiques sur la sémantique). Il s'agit des heuristiques NPC (Noeud le Plus Contraint), NMC (Noeud) le Moins Contraint) et NPD (Noeud le Plus Dur). Ici les résultats ne sont pas significatifs.

Instance	Taille		Ordre		
	$\|V\|$	$\|E\|$	NPC	NMC	NPD
$renault$	101	134	3,28	3,1 4	3,18
$series-6-ext$	11	30	2,24	1,88	1,59
$series-7-ext$	12	41	2,02	1,5	2,04
$domino-100-100_ext$	100	100	0,21	0,49	0,53
$domino-100-200_ext$	100	100	5,97	7,02	5,65
$domino-100-300_ext$	100	100	12,48	13,01	12,67
$langford-2-4-ext$	8	32	2,83	1,6	0,93
$geom-30a-4-ext$	30	81	>20	>20	>20
$pigeons-7-ext$	7	21	>20	>20	>20
$haystacks-06_ext$	36	96	>20	>20	>20

Tab. 2 – Temps d'exécution pour les différents ordre sans filtrage.

Dans le tableau 3, nous reportons les résultats sur les heuristiques structurelles même si l'heuristique PGC (Plus Grand Nombre de Contraintes) semble être la meilleure.

Instance	Ordre				
	BE	PGC	PPC	PNF	PGS
$renault$	3,40	3,40	3,09	3,06	3,12
$series-6-ext$	1,54	1,55	1,88	1,50	1,55
$series-7-ext$	2,02	2,08	1,73	2,04	1,99
$domino-100-100_ext$	0,20	0,25	0,58	0,5	0,30
$domino-100-200_ext$	5,90	5,85	6,51	5,5	6,5
$domino-100-300_ext$	12,77	12,68	12,57	12,39	12,80
$langford-2-4-ext$	0,54	0,68	1,5	0,50	0,99
$geom-30a-4-ext$	>20	3,09	>20	1,7	>20
$pigeons-7-ext$	>20	>20	>20	>20	>20
$haystacks-06_ext$	>20	>20	>20	>20	>20

TAB. 3 – Temps d'exécution pour les différents ordre sans filtrage.

5.2.2 Algorithme avec filtrage et études des différents ordres

Pour améliorer l'algorithme de base, nous avons introduit le filtrage des relations de tous les noeuds fils de chaque noeud instancié. Les tableaux 4 et 5 montrent le comportement des différentes heuristiques du choix de la racine et l'apport du filtrage. Le filtrage améliore clairement les résultats. Nous observons en effet qu'ici toutes les instances sont résolues.

Le tableau 4 montre que parmi les heuristiques portant sur la sémantique, l'heuristique NPD (Noeud le Plus Dur) est la meilleure. Ceci n'est pas surprenant car elle tient compte de la dureté des contraintes mais aussi de la taille des relations, c'est à dire de l'espace possible des solutions.

Instance	Taille		Ordre		
	$\|V\|$	$\|E\|$	NPC	NMC	NPD
$renault$	101	134	3,03	3,02	3,04
$series-6-ext$	11	30	0,10	0,38	0,43
$series-7-ext$	12	41	0,55	3,48	0,13
$domino-100-100_ext$	100	100	0,12	0,70	0,49
$domino-100-200_ext$	100	100	0,23	2,32	0,76
$domino-100-300_ext$	100	100	0,36	5,14	0,75
$Langford$	8	32	0,14	0,36	0,09
$geom-30a-4-ext$	30	81	7	1,23	0,03
$pigeons-7-ext$	7	21	11	4,23	4,2
$haystacks-06_ext$	36	96	6,96	3,32	3,31

TAB. 4 – Temps d'exécution pour les différents ordre en exploitant le filtrage.

Le tableau 5 montre que parmi les heuristiques statiques, l'heuristique PGC (Plus Grand Cluster) est la meilleure. Ceci signifie qu'il serait intéressant de traiter à la racine le noeud qui contient le maximun de contraintes.

Remarque 1 *Nous avons considéré les heuristiques structurelles et sémantiques séparément. Mais il faudra sans doute les combiner. Les décompositions structurelles se contentent de décomposer les problèmes en exploitant uniquement les propriétés structurelles de leurs représentations en graphes ou hypergraphes. Mais la résolution du problème fait intervenir les relations associées aux contraintes. Cette donnée ajoute une dimension non négligeable à la complexité du problème du point de vue expérimental.*

Instance	Ordre				
	BE	PGC	PPC	PNF	PGS
$renault$	3,38	3,02	4,41	3,03	3,31
$series-6-ext$	0,09	0,07	0,43	0,07	0,12
$series-7-ext$	0,08	3,48	3,18	3,5	0,96
$domino-100-100_ext$	0,125	0,14	0,49	0,13	0,22
$domino-100-200_ext$	0,24	0,23	0,27	0,23	0,28
$domino-100-300_ext$	0,35	0,36	0,69	0,34	0,51
$Langford$	0,03	0,31	0,91	8,73	1,02
$geom-30a-4-ext$	0,03	0,03	5	0,23	1,02
$pigeons-7-ext$	4,34	4,19	3,5	12	4,23
$haystacks-06_ext$	3,33	3,35	3,33	3,22	3,21

TAB. 5 – Temps d'exécution pour les différents ordre en exploitant le filtrage.

Il faut des heuristiques de choix du noeud racine qui tiennent compte du nombre de tuples, du nombre de tuples solutions, de la taille des séparateurs, ...etc.

Pour situer notre approche par rapport à la méthode BTD, nous avons testé les problèmes de la famille renault modofiés traités dans le papier de Philippe Jégou et al [13]. Nous n'avons malheureusement pas pu obtenir de solutions pour cette famille de problèmes pour l'instant. Notre approche est à améliorer probablement par l'exploration des goods et des nogoods, comme nous le soulignons dans la conclusion.

6 Conclusion

Pour remedier à l'inconvénient de la méthode de résolution des CSP par l'hypertree decomposition, nous avons proposé dans ce papier une autre approche basée sur un backtracking sur les tuples. Sachant que le choix du noeud à instancier en priorité est primordial pour le comportement de la méthode, nous avons exploré plusieurs stratégies pour le choix de la racine à instancier en priorité. Les résultats expérimentaux montrent effectivement que le choix judicieux consiste à choisir le noeud le plus dur, c'est à dire le noeud qui minimise le rapport entre le nombre de solutions attendues au niveau du noeud et le nombre de solutions dans le pire des cas. Pour estimer l'intérêt pratique de notre approche, nous avons comparé notre méthode à celle de Gottlob et nos résultats sont encourageants.

Nous avons aussi cherché à comparer notre approche à BTD. Sur le plan théorique ces deux approches sont bien sûr comparables dans le sens où toutes les deux recherchent la solution d'un CSP par un algorithme énumératif guidé par un ordre induit par une décomposition structurelle. BTD explore les décompositions arborescentes alors que notre approche explore les décompositions hyper-arborescentes. BTD a été étendue par ailleurs aux décompositions hyper-arborescentes. la spécifité de notre approche repose sur le fait que les clusters correspondent à un ensemble de contraintes et non pas un ensemble de variables. Du point de vue

expérimental, pour comparer notre approche à BTD, nous avons testé la famille des Benchmarks Renault modifés. Les temps d'exécution ne sont pas aussi bons que ceux obtenus par BTD. Notre approche est à améliorer. Nous allons explorer les notions de goods et de nogoods pour accelérer la recherche de solution au niveau d'un noeud donné. Une de nos perspectives serait aussi d'étudier des heuristiques pour construire des hypertree décompositions basées sur les propriétés structurelles et sémantiques des clusters.

Par ailleurs, la complexité théorique de HD_DBT dépendant du nombre de noeuds de l'hypertree décomposition, il faudra trouver des heuristiques minimisant le nombre de noeuds de l'hypertree décomposition et ceci se fera sans doute au détriment de la largeur de l'hypertree decomposition. Ce qui est en contradiction avec les résultats théoriques sur l'hypertree décomposition.

Références

[1] K. Amroun and Z. Habbas. Résolution des problèmes de satisfaction de contraintes par les techniques de décomposition structurelles. Rapport de recherche, Laboratoire de Recheche en Informatique Théorique et Appliquée de Metz, 2009.

[2] M. Aït-Amokhtar, K Amroun, and Z. Habbas. hypertree decomposition for solving constraint satisfaction problems. In *Proceedings of International conference on Agents and Artificial Intelligence,ICAART 2009*, pages 85–92, Portugal, 2009.

[3] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[4] R. Dechter and J. Pearl. The cycle-cutset method for improving search performance in AI applications. In *Proceedings of the third IEEE on Artificial Intelligence Applications*, pages 224–230, Orlando, 1987.

[5] R. Dechter and J. Pearl. Tree-clustering schemes for constraint-processing. In *Proceedings of the sixth National Conference on Artificial Intelligence (AAAI-88)*, pages 150–154, Saint Paul, MN, 1988.

[6] T. Dermaku, A.and Ganzow, G. Gottlob, B. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decompositions. Technical report, DBAI-R, 2005.

[7] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29 :24–32, 1982.

[8] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. In *Proceedings of IJCAI'99*, pages 394–399, 1999.

[9] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *Proceedings of DEXA'99*, 1999.

[10] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions : A survey. In *Proceedings of MFCS '01*, pages 37–57, 2001.

[11] Georg Gottlob, Zoltan Miklos, and Thoma90(de)-392(la)-392(S