Complexité de Forward Checking et Hiérarchie des Décompositions de CSP Revisitées

Philippe Jégou, Samba Ndojh Ndiaye, Cyril Terrioux

Complexité de Forward Checking et Hiérarchie des Décompositions de CSP Revisitées

Philippe Jégou **Samba Ndojh Ndiaye** **Cyril Terrioux**

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

Résumé

Dans cette contribution, nous proposons une nouvelle évaluation de la complexité de la procédure Forward Checking appliquée à la résolution de CSP n-aires à domaines finis. Généralement, cette évaluation prend en compte la taille des domaines associés aux variables. Ici, nous l'exprimons relativement à la taille des relations de compatibilité associées aux contraintes. Cette nouvelle approche permet ainsi, parfois, de fournir une meilleure borne de complexité théorique. Nous montrons l'intérêt essentiel de cette démarche en revenant sur les résultats proposés dans [10] et qui concernent la hiérarchie des décompositions de CSP (comparaisons des méthodes de décompositions structurelles), et qui constituent un résultat fondamental du domaine. Nous invalidons une partie de ces résultats en démontrant notamment que le concept de décomposition en hyperarbres, situé au sommet de cette hiérarchie, n'est finalement pas meilleur que celui de décomposition arborescente. Ce résultat, essentiellement théorique, s'avère toutefois en adéquation avec les observations expérimentales obtenues ces dernières années au sein de la communauté.

1 Introduction

Il est bien connu que le formalisme des problèmes de satisfaction de contraintes (CSP) et ses généralisations, comme notamment les CSPs Valués, offrent des cadres de formalisation extrêmement puissants, à la fois pour l'expression et la résolution de nombreux problèmes, tant dans le champ de l'intelligence artificielle, que dans d'autres domaines tels que la recherche opérationnelle ou l'optimisation combinatoire. Un CSP peut s'exprimer sous la forme d'un ensemble fini X de variables qui peuvent être affectées dans leurs domaines de valeurs donnés par D, en satisfaisant simultanément un ensemble C de contraintes. Une telle affectation est alors une solution du CSP. Le problème à résoudre consiste généralement à trouver une solution ou bien à énumérer toutes les solutions. Malheureusement, le test d'existence de solution est déjà NP-complet. Pour la résolution de CSP, différentes classes d'algorithmes ont été proposées, qui combinent le backtracking et le filtrage. D'un point de vue pratique, ces algorithmes sont souvent efficaces. C'est le cas notamment, quand un niveau de filtrage adéquat est exploité lors du backtracking. Généralement, ce compromis entre développement d'une arborescence de recherche et filtrage s'appuie sur la notion d'arc-consistance. Cette notion permet de filtrer, pendant la recherche, le domaine des variables non encore affectées. Le bon niveau de filtrage se situe généralement entre FC (Forward Checking [12]), qui constitue la forme la plus restreinte d'arc-consistance (réalisée en une passe et par propagation directe), et MAC [22] qui maintient la consistance d'arc globale sur le sous-problème résultant de la dernière affectation. Alors que ces algorithmes peuvent être réellement efficaces en pratique, leur complexité est en $O(S.m^n)$ où S exprime la taille du CSP en entrée, n le nombre de variables et m la taille maximum des domaines des variables. Cette évaluation est clairement guidée par cette taille des domaines ainsi que par le nombre de variables traitées.

Aussi, différentes approches ont été proposées pour améliorer cette borne de complexité, notamment en s'appuyant sur l'exploitation de propriétés structurelles souvent présentes dans les données. L'intérêt

qu'il y a à exploiter de telles propriétés a été observé, outre dans les CSP [8] ou dans l'optimisation sous contraintes (VCSPs) [23, 5], dans le domaine des bases de données relationnelles [1, 11], de la logique propositionnelle [20, 14, 18], ou des réseaux Bayesiens ou probabilistes [6, 4]. Formellement, les résultats de complexité sont basés sur l'existence de propriétés topologiques au sein d'une donnée. Généralement, elles s'expriment en termes de décomposition arborescente (tree-decomposition [21]) ou de décomposition en hyperarbre (hypertree-decomposition [10]) du réseau de contraintes formalisant la structure. Dans le cas de la décomposition arborescente, la complexité en temps des meilleures approches structurelles est en $O(S.m^{w+1})$, où w est la largeur de la décomposition arborescente considérée. L'intérêt est double. Sur le plan théorique, il réside notamment dans la garantie de vérifier $w < n$. Plus important encore, il est souvent observé en pratique que $w \ll n$. Si l'on considère la décomposition en hyperarbre, la complexité en temps est alors en $O(S.r^h)$, où h est la largeur de la décomposition en hyperarbre utilisée tandis que r exprime la taille maximum des relations (nombre maximum de tuples par table) associées aux contraintes. Un résultat bien connu dans ce cadre, présenté dans [10], tend à montrer que la décomposition en hyperarbre est meilleure que la décomposition arborescente puisqu'il a été établi que $h \leq w$. Plus généralement, [10] présente une comparaison théorique entre méthodes de résolution structurelles de CSP qui est exprimée en termes d'une hiérarchie au sommet de laquelle figure la décomposition en hyperarbre. Ce résultat montre ainsi que cette approche constitue la meilleure méthode de décomposition.

L'intérêt pratique de certaines approches structurelles a été démontré dans quelques travaux récents autour des (V)CSPs [16, 17, 5, 19]. Ces observations empiriques semblent contredire les résultats théoriques. En effet, ils s'appuient essentiellement sur la décomposition arborescente, alors que les méthodes fondées sur la décomposition en hyperarbre n'ont pour le moment pas montré un quelconque intérêt pratique.

Sur la base de ces premiers éléments, nous pensons que ceci est dû au fait que les bornes de complexité de la décomposition en hyperarbre sont malheureusement souvent atteintes, au détriment de l'efficacité pratique. Par exemple, l'approche par décomposition en hyperarbre fait l'hypothèse d'une représentation des relations associées aux contraintes par des tables, ce qui souvent s'avère irréaliste d'un point de vue pratique. De plus, pour s'assurer d'une complexité en $O(S.r^h)$, les méthodes procèdent par le biais de jointures de relations. Dans le cadre de la résolution de CSP, une telle démarche est généralement irréaliste du fait de la taille des relations qui seront engendrées. D'autre part, les méthodes qui ont prouvé leur faisabilité et leur intérêt pratique sont en fait fondées sur des techniques d'affectation de variables, exploitant notamment l'efficacité maintenant bien établie, des algorithmes alliant backtracking et filtrage (comme FC ou MAC), quand bien même elles conduisent à des bornes de complexité du type $O(S.m^{w+1})$, donc *a priori* de moins bonne qualité.

Dans ce papier, nous introduisons une justification théorique à ces observations expérimentales. Dans un premier temps, nous étudions la complexité d'algorithmes tels que FC, mais avec une approche différente de celle classiquement utilisée. En effet, nous présentons une analyse de leur complexité relative à la taille des relations associées aux contraintes plutôt qu'à la taille des domaines. Ceci nous conduit à une borne en $O(S.r^k)$, où k est un paramètre structurel du CSP qui se révèle indépendant de toute décomposition. Ce résultat est ensuite exploité pour montrer que les méthodes s'appuyant sur la décomposition arborescente voient leur complexité en fait ramenée à $O(S.r^h)$. Ce résultat permet d'entrevoir sous un nouveau jour les relations qui existent entre les diverses méthodes de décomposition. Cela nous conduit ainsi naturellement à reconsidérer la fameuse hiérarchie introduite dans [10] en la modifiant. Notons que la preuve du résultat présenté dans [10] demeure vraie. Ici, nous modifions la conséquence du résultat, c'est-à-dire la hiérarchie induite entre la complexité des méthodes (pas la comparaison des paramètres d'(hyper)graphes), car nous modifions l'hypothèse de base qui repose sur la complexité de l'algorithme utilisé pour la résolution des clusters dans les approches basées sur la décomposition arborescente.

Ce papier est organisé comme suit. La prochaine section présente une analyse de la complexité des algorithmes énumératifs "à la FC", fondée sur la taille des relations. La troisième section montre comment cette analyse peut être exploitée de sorte à proposer de nouvelles bornes de complexité pour les méthodes de résolution fondées sur la décomposition arborescente. Cette section présente ensuite une comparaison théorique avec les méthodes s'appuyant sur la décomposition en hyperarbre. Pour conclure, nous présentons quelques questions ouvertes induites par ces résultats.

2 Complexité de Forward Checking Revisitée

2.1 Préliminaires

Un *problème de satisfaction de contraintes à domaines finis* ou *réseau de contraintes fini* (X, D, C, R)

est défini par un ensemble de variables $X = \{x_1, \dots x_n\}$, un ensemble de domaines $D = \{d_1, \dots d_n\}$ (le domaine d_i contient toutes les valeurs possibles pour la variable x_i), un ensemble C de contraintes portant sur les variables. Une contrainte $c_i \in C$ sur un ensemble ordonné de variables, $c_i = (x_{i_1}, \dots x_{i_{a_i}})$ (a_i désigne l'arité de la contrainte c_i), est définie par une relation associée r_i (appartenant à R) qui exprime les combinaisons de valeurs admissibles pour les variables de c_i. Nous noterons par a l'arité maximum des contraintes dans C. Sans manque de généralité, nous supposons que toute variable est concernée par au moins une contrainte. Une solution de (X, D, C, R) est une affectation de chaque variable qui satisfait toutes les contraintes. La structure d'un CSP peut être représentée par l'hypergraphe (X, C), appelé *hypergraphe de contraintes*. Si chaque contrainte ne porte que sur deux variables, (X, C) est alors un graphe, appelé *graphe de contraintes*.

Dans cet article, nous supposerons que les relations peuvent être représentées par des tables, comme dans le cas de la théorie des bases de données relationnelles (on supposera de plus qu'elles ne sont pas vides). Soit $\mathcal{A}$ l'affectation d'un ensemble de variables $Y = \{x_1, \dots x_k\}$. $\mathcal{A}$ peut être considéré comme un tuple $\mathcal{A} = (v_1, \dots v_k)$. La *projection* de $\mathcal{A}$ sur un sous-ensemble Y' de Y, notée $\mathcal{A}[Y']$, est la restriction de $\mathcal{A}$ aux variables de Y'. La projection de la relation r_i sur le sous-ensemble Y' de c_i est l'ensemble de tuples $r_i[Y'] = \{t[Y'] | t \in r_i\}$. La jointure de relations sera notée $\bowtie$, et la jointure de $\mathcal{A}$ avec une relation r_i est $\mathcal{A} \bowtie r_i = \{t | t \text{ est un tuple sur } Y \cup c_i \text{ et } t[Y] = \mathcal{A} \text{ et } t[c_i] \in r_i\}$.

Si les relations sont représentées par des tables, nous noterons S la majoration de la taille d'un CSP avec $S = n.m + a.r.|C|$ où $r = max\{|r_i| : r_i \in R\}$.

L'approche naturelle pour la résolution de CSP est basée sur la procédure classique appelée *Backtracking (BT)*. La complexité de cet algorithme est $O(a.r.|C|.m^n)$ puisque le nombre potentiel de nœuds développés pendant la recherche est m^n, qu'il y a au plus $|C|$ tests de consistance pour un nœud donné et que le coût d'un test $\mathcal{A}[c_i] \in r_i$ est calculable en $O(a.r)$. On peut ramener cette complexité à $O(S.m^n)$. Généralement, cet algorithme n'est pas utilisé car il est inefficace en pratique.

2.2 Complexité exprimée via la taille des domaines

L'approche la plus classique pour améliorer BT s'appuie sur la notion de filtrage. Après une affectation $\mathcal{A}$ sur Y où x_k est la dernière variable affectée, un filtrage sera réalisé sur les domaines des variables futures. Ce filtrage supprime les valeurs qui ne sont pas compatibles avec la dernière affectation réalisée (celle de x_k). Le premier algorithme proposé avec un tel filtrage a été introduit par [12] et est appelé Forward Checking (FC). Il a été initialement défini sur les CSP binaires.

De nombreuses extensions et généralisations de FC ont été proposées, soit pour résoudre des CSP n-aires, soit pour exploiter des filtres plus puissants [12, 22, 2]. Dans cet article, nous considérons l'une de ces extensions appelée nFC2 [2] et dont le niveau de filtrage semble réaliser un bon compromis, au regard des résultats expérimentaux obtenus sur les CSP n-aires.

Considérons une affectation courante $\mathcal{A}$ qui satisfait toutes les contraintes incluses dans Y. Après l'affectation de la dernière variable x_k, nFC2 applique un filtrage par arc-consistance en une passe sur chaque contrainte de $C_{c,f}$ où $C_{c,f}$ est l'ensemble des contraintes portant à la fois sur la dernière variable affectée et sur au moins une variable future (formellement, $C_{c,f} = \{c_j \in C | x_k \in c_j \text{ et } c_j \not\subset Y\}$). Il faut noter que le filtrage résultant dépend de l'ordre dans lequel les contraintes seront prises en compte. Afin d'exprimer plus facilement les domaines filtrés, nous définissons ici nFC2$_m$ de sorte à ce que cette procédure corresponde au filtrage minimal réalisé par nFC2 pour l'ensemble des ordres potentiels pour la prise en compte des contraintes. Il s'avère que cela ne change rien pour ce qui concerne la complexité du filtrage. Ainsi, dans nFC2$_m$, le domaine de chaque variable future est filtré indépendamment de celui des autres variables futures. Le domaine filtré d'une variable future x_i est l'ensemble des valeurs $d_i^{\mathcal{A}} = \{v_i \in d_i^{\mathcal{A}[Y \setminus \{x_k\}]} | \forall c_j \in C_{c,f} \text{ t.q. } x_i \in c_j, \exists t \in r_j, t \in \prod_{x_l \in c_j} d_l^{\mathcal{A}[Y \setminus \{x_k\}]} \text{ et } t[x_i] = v_i\}$ et pour le cas d'une variable x_j déjà affectée, ce sera $d_j^{\mathcal{A}} = \{\mathcal{A}[x_j]\}$. Initialement, $d_i^{\emptyset} = \{v_i \in d_i | \forall c_j \in C, x_i \in c_j, \exists t \in r_j, t[x_i] = v_i\}$ (i.e. nous considérons seulement les valeurs v_i qui figurent dans les relations portant sur x_i). Ainsi, étant donnée une affectation $\mathcal{A}$, le filtrage causé par nFC2$_m$ (i.e. l'ensemble des couples (x_i, v_i) tels que la valeur v_i a été supprimée du domaine de x_i) est nécessairement inclus dans celui occasionné par nFC2 (quel que soit l'ordre de prise en compte des contraintes).

Comme pour nFC2, si le domaine d'une variable future devient vide, alors nFC2$_m$ réalise un backtrack ; sinon, la recherche se poursuit avec l'affectation de l'une des variables futures. On peut facilement démontrer que nFC2$_m$ est correct et complet (la preuve est la même que celle proposée dans [2] pour nFC2). Au niveau de la complexité en temps, la taille de l'espace de recherche est bornée par m^n. L'analyse présentée dans [2] indique par ailleurs que le coût local à chaque nœud, i.e. le nombre de tests de consistance réalisés, est $O(|C_{c,f}|.(a-1).m^{a-1})$. Notons qu'il est facile d'exprimer cette complexité en prenant en compte

la taille maximum r des relations r_i, pour avoir finalement $O(|C_{c,f}|.a.r)$ car $(a-1).m^{a-1}$ exprime en fait la taille potentielle de la relation (si l'on enlève une variable), valeur qui peut en fait, pour le cas d'une représentation sous forme de table, être majorée par r . Par ailleurs, si S désigne la taille du CSP, nécessairement $|C_{c,f}|.a.r \in O(S)$ et ainsi, le coût de nFC2 et de nFC2$_m$ peut être borné par $O(S.m^n)$. Nous obtenons ainsi formellement :

Théorème 1 *La complexité en temps de nFC2$_m$ pour la résolution d'un CSP (X, D, C, R) est $O(S.m^n)$.*

Nous donnons dans la figure 1 l'exemple de CSP présenté dans [2] (figure 1, page 210) afin d'illustrer les différences entre nFC2 et nFC2$_m$. Le CSP possède 6 variables $\{x, y, z, u, v, w\}$ avec un même domaine $\{a, b, c\}$, et sont sujettes à 3 contraintes $c_1(x, y, z)$, $c_2(u, v, w)$ et $c_3(x, y, w)$.

c_1			c_2			c_3		
x	y	z	u	v	w	x	y	w
a	a	a	a	a	a	a	a	a
a	b	c	a	b	b	a	b	c
a	c	b	c	c	c			

Affect	Algo	Action
(x, a)	nFC2	AC($\{c_3\}$) puis AC($\{c_1\}$)
	nFC2$_m$	AC($\{c_3\}$) et AC($\{c_1\}$)
(u, a)	nFC2	AC($\{c_2\}$)
	nFC2$_m$	AC($\{c_2\}$)

(x, a)	nFC2	nFC2$_m$
$d_x^{\mathcal{A}}$	a	a
$d_y^{\mathcal{A}}$	a, b	a, b
$d_z^{\mathcal{A}}$	a, c	a, b, c
$d_u^{\mathcal{A}}$	a, c	a, c
$d_v^{\mathcal{A}}$	a, b, c	a, b, c
$d_w^{\mathcal{A}}$	a, c	a, c

(u, a)	nFC2	nFC2$_m$
D(x)	a	a
D(y)	a, b	a, b
D(z)	a, c	a, b, c
D(u)	a	a
D(v)	a	a
D(w)	a	a

FIG. 1 – Filtrage réalisé par nFC2 et nFC2$_m$ sur un problème, après les affectations (x, a) et (u, a).

Les domaines initiaux des variables sont réduits aux valeurs présentes dans les relations. Les algorithmes nFC2 et nFC2$_m$ réalisent la consistance d'arc ("in one pass" [2]) sur le même ensemble de contraintes. Puisque nFC2 prend en compte les filtrages réalisés par arc-consistance sur les contraintes précédentes, l'ordre des contraintes est important. Mais ceci n'est pas le cas pour nFC2$_m$ qui se comporte comme si les filtrages réalisés par arc-consistance étaient accomplis en même temps.

Ainsi, suite à l'affectation (x, a), si nFC2 réalise AC($\{c_1\}$) avant AC($\{c_3\}$), la valeur b dans le domaine de la variable z ne sera pas supprimée, contrairement à ce qui se passe dans l'exemple.

2.3 Complexité exprimée via la taille des relations

Nous analysons maintenant la complexité de nFC2$_m$ en considérant la taille des relations de compatibilité. Le théorème 2 montre que toutes les affectations courantes (nœuds de l'arbre de recherche) développées par nFC2$_m$ vérifient à la fois les contraintes incluses dans l'ensemble des variables de l'affectation considérée, mais aussi celles dont seulement une partie des variables a été affectée. A l'aide de cette propriété, nous pouvons alors assurer que nFC2$_m$ énumérera les affectations dans la limite des jointures des relations correspondantes. En d'autres termes, le développement de l'arbre de recherche se voit circonscrit au périmètre des tuples présents dans les jointures de relations. Ce simple constat va nous permettre de raffiner la complexité. Nous introduisons quelques notations de façon à alléger le formalisme :

- L'ensemble ordonné $Y = (x_1, \ldots, x_k) \subsetneq X$ est noté Y_k.
- $C_Y = \{c_j \in C | c_j \cap Y \neq \emptyset\}$.
- $C_{Y,f} = \{c_j \in C_Y | c_j \not\subset Y\}$.

Théorème 2 *Si $\mathcal{A} = (v_1, \ldots, v_k) \in \bowtie_{c_i \in C_{Y_k}} r_i[c_i \cap Y_k]$ et $\forall j, k+1 \leq j \leq n, d_j^{\mathcal{A}} \neq \emptyset$, alors $\forall v_{k+1} \in d_{k+1}^{\mathcal{A}}, (v_1, \ldots, v_k, v_{k+1}) \in \bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}]$.*

Preuve : Soit $\mathcal{A}' = (v_1, \ldots, v_k, v_{k+1})$ l'extension de l'affectation $\mathcal{A}$ obtenue en donnant à x_{k+1} la valeur $v_{k+1} \in d_{k+1}^{\mathcal{A}}$. Nous voulons prouver que $\mathcal{A}' \in \bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}]$. Nous avons $\bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}] = (\bowtie_{c_i \subset Y_{k+1}} r_i) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f}} r_i[c_i \cap Y_{k+1}]) = (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \setminus Y_k = \{x_{k+1}\}} r_i) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f} \text{ et } x_{k+1} \in c_i} r_i[c_i \cap Y_{k+1}]) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f} \text{ et } x_{k+1} \notin c_i} r_i[c_i \cap Y_k])$.

Par construction de $d_{k+1}^{\mathcal{A}}$, pour chaque contrainte $c_i \in C$ tel que. $x_{k+1} \in c_i$ et $c_i \cap Y \neq \emptyset$, il existe un tuple $t \in r_i$ tel que $t[Y_k \cap c_i] = \mathcal{A}[Y_k \cap c_i]$ et $t[x_{k+1}] = v_{k+1}$, i.e. tel que. $t[c_i \cap Y_{k+1}] = \mathcal{A}'[c_i \cap Y_{k+1}]$. Par conséquent, $\mathcal{A}'[Y_{k+1} \cap (\cup_{c_i | x_{k+1} \in c_i} c_i)] \in (\bowtie_{c_i \setminus Y_k = \{x_{k+1}\}} r_i) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f} \text{ et } x_{k+1} \in c_i} r_i[c_i \cap Y_{k+1}])$. De plus, $\mathcal{A} \in \bowtie_{c_i \in C_{Y_k}} r_i[c_i \cap Y_k] = (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k,f}} r_i[c_i \cap Y_k]) = (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k,f} \text{ et } x_{k+1} \notin c_i} r_i[c_i \cap Y_k]) \bowtie (\bowtie_{c_i \in C_{Y_k,f} \text{ et } x_{k+1} \in c_i} r_i[c_i \cap Y_k])$. Aussi $\mathcal{A}'[Y_k] =$

$\mathcal{A} \in (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k}, f \text{ et } x_{k+1} \notin c_i} r_i[c_i \cap Y_k])$. Par conséquent, $\mathcal{A}' \in \bowtie_{c_i \cap Y_{k+1}} r_i[c_i \cap Y_{k+1}]$. □

Théorème 3 *La complexité en temps de nFC2$_m$ pour la résolution d'un CSP (X, D, C, R) est $O(S.r^{|C|})$.*

Preuve : Pour toute affectation $\mathcal{A}$ sur Y considérée par nFC2$_m$ (l'ordre des variables est supposé statique et induit par Y), nous avons $\mathcal{A} \in \bowtie_{c_i \in C_Y} r_i[c_i \cap Y]$ (d'après le théorème 2). Aussi, comme le nombre de nœuds est égal au nombre d'affectations, il peut être borné, pour une profondeur donnée $|Y|$, par $\prod_{c_i \in C_Y} |r_i|$. De plus, $C_Y \subset C$. Par conséquent, $\prod_{c_i \in C_Y} |r_i| \leq \prod_{c_i \in C} |r_i| \leq r^{|C|}$. Comme la hauteur de l'arbre de recherche est bornée par n, le nombre de nœuds est borné $n.r^{|C|}$. Comme le coût local en chaque nœud est $O(S)$, nFC2$_m$ a une complexité en $O(S.r^{|C|})$. □

Cette complexité peut être raffinée en prenant en compte la notion de recouvrement minimum (minimum cover) de l'ensemble de contraintes. En effet, on voit au travers du théorème précédent, que l'ensemble des affectations partielles développées est circonscrit à l'ensemble des tuples figurant dans la jointure des relations concernées par les variables affectées. De fait, cet ensemble est inclus dans toute jointure portant sur un sous-ensemble de ces relations, en particulier sur un sous-ensemble recouvrant minimalement les variables considérées. Nous posons cela formellement :

Définition 1 *Etant donnés un ensemble fini X et C, une famille de sous-ensembles de X, un recouvrement de X est un sous-ensemble $C' \subset C$ tel que $\cup_{c_i \in C'} c_i = X$. C' est un recouvrement minimum de X s'il n'existe pas de recouvrement C'' tel que $|C''| < |C'|$. La valeur $|C'|$ sera notée $k_{(X,C)}$.*

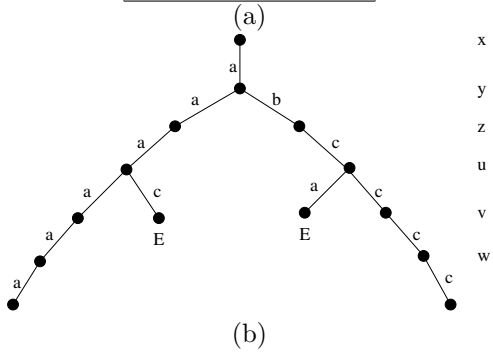
Théorème 4 *La complexité en temps de nFC2$_m$ pour la résolution d'un CSP (X, D, C, R) est $O(S.r^{k_{(X,C)}})$.*

Preuve : Pour toute affectation $\mathcal{A}$ de Y considérée par nFC2$_m$ (l'ordre des variables est encore supposé statique et induit par Y), nous avons $\mathcal{A} \in \bowtie_{c_i \in C_Y} r_i[c_i \cap Y]$ (d'après le théorème 2). De plus, à chaque niveau de l'arbre de recherche, nous avons $\{c_i \in C' | c_i \cap Y \neq \emptyset\} \subset C_Y$ avec C' un recouvrement de X. Par conséquent, $\mathcal{A} \in \bowtie_{c_i \in C' | c_i \cap Y \neq \emptyset} r_i[c_i \cap Y]$. Aussi, à chaque niveau, le nombre de nœuds est borné par $\prod_{c_i \in C' | c_i \cap Y \neq \emptyset} |r_i| \leq \prod_{c_i \in C'} |r_i| \leq r^{|C'|}$. Par conséquent, le nombre total de nœuds est borné par $n.r^{|C'|}$. Aussi, comme le coût local en chaque nœud est $O(S)$, nFC2$_m$ a une complexité en $O(S.r^{|C'|})$. En considérant un recouvrement minimum, cette complexité est alors $O(S.r^{k_{(X,C)}})$. □

Notons que nous avons nécessairement $r^{k_{(X,C)}} \leq n$. Ensuite, nous pouvons observer que nFC2$_m$ exploite donc implicitement un recouvrement minimum de C sans avoir, pour autant à en rechercher un, tâche qui pourrait se révéler extrêmement coûteuse (le calcul de recouvrement minimum est NP-difficile [9]). Enfin, notons que ce résultat peut être étendu à tout autre algorithme qui maintiendrait un filtrage au moins aussi puissant que celui de nFC2$_m$. Cela vaut notamment pour les différentes versions nFCi (pour $i \geq 2$) mais aussi pour M(G)AC.

x	y	z	u	v	w
a	a	a	a	a	a
a	a	a	a	b	b
a	a	a	c	c	c
a	b	c	a	a	a
a	b	c	a	b	b
a	b	c	c	c	c
a	c	b	a	a	a
a	c	b	a	b	b
a	c	b	c	c	c

(a)



(b)

FIG. 2 – Relation $r_1 \bowtie r_2$ (a) et arbre de recherche nFC2$_m$ (b) (E représente les échecs).

Considérons à nouveau l'exemple de la figure 1. L'ensemble $\{c_1, c_2\}$ constitue un recouvrement minimum de ce problème. Nous reportons dans la figure 2 la table associée à la jointure de r_1 avec r_2 ainsi que l'arbre de recherche de nFC2$_m$ sur la base de l'ordre statique d'affectation des variables (x, y, z, u, v, w). On peut constater que chaque tuple obtenu par nFC2$_m$ figure dans la jointure de r_1 et r_2. En d'autres termes, le coût de nFC2$_m$ est inférieur à celui de la jointure $r_1 \bowtie r_2$.

Si l'on généralise cet exemple, en rajoutant un nombre quelconque de variables dans c_1 et c_2, l'évaluation classique de la complexité donnera $O(S.m^n)$, soit une complexité exponentielle alors qu'en fait, ce sera $O(S.r^2)$ en considérant nFC2$_m$ avec la nouvelle évaluation. De plus, le résultat théorique donné par le théorème 4 contredit de façon surprenante les évaluations empiriques. En effet, en général, plus un problème est contraint, plus sa résolution sera facilitée (excepté

pour les problèmes fortement sous-contraints). Aussi, les bornes données par les théorèmes 3 et 4 avec $S.r^{k_{(X,C)}} \leq S.r^{|C|}$ indique que moins le problème est contraint, moins sa complexité est élevée. Mais en observant l'exemple, il s'avère plus efficace de tester également la contrainte c_3 pendant la recherche sur $c_1 \cup c_2$, que de se limiter aux seuls tests de c_1 et c_2, car nous déterminerons alors plus rapidement les points d'échec en exploitant aussi c_3.

3 La hiérarchie des complexités temporelles des méthodes de décomposition revisitée

3.1 Une nouvelle expression de la complexité de la résolution par décomposition arborescente

La décomposition (arborescente) de réseaux de contraintes a été introduite dans [8] avec la méthode Tree-Clustering (TC). TC, ainsi que les autres méthodes basées sur la même approche [7, 16], s'appuient sur la notion de décomposition arborescente de graphes. Néanmoins, pour le cas des CSP n-aires, et donc des hypergraphes de contraintes, cette notion peut être exploitée en considérant le *graphe primal* (ou 2-section) de l'hypergraphe. Soit $H = (X, C)$ un hypergraphe, le graphe primal de H est le graphe $G = (X, A_C)$ où $A_C = \{\{x, y\} \subset X : \exists c_i \in C \ t.q. \ \{x, y\} \subset c_i\}$. Aussi, étant donné un CSP, nous considérons son graphe primal de manière à définir une décomposition arborescente associée au CSP.

Définition 2 *Une décomposition arborescente d'un graphe $G = (X, A_C)$ est une paire (E, T) où $T = (I, F)$ est un arbre de nœuds I et d'arêtes F et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé cluster) E_i est un nœud de T et vérifie :*

(i) $\cup_{i \in I} E_i = X$,

(ii) pour une arête $\{x, y\} \in A_C$, il existe $i \in I$ avec $\{x, y\} \subset E_i$, et

(iii) pour tout $i, j, k \in I$, si k est une chaîne de i à j dans T, alors $E_i \cap E_j \subset E_k$.

La largeur w d'une décomposition arborescente (E, T) est égale à $max_{i \in I}|E_i| - 1$. La largeur d'arbre (tree-width) w^* de G est la largeur minimale pour toutes les décompositions arborescentes de G.

La figure 3 présente un exemple d'hypergraphe de contraintes et l'une de ses décompositions arborescentes possibles parmi celles dont la largeur est minimale.

Initialement, TC a été défini pour les CSP binaires. Néanmoins, des extensions ont été proposées pour le

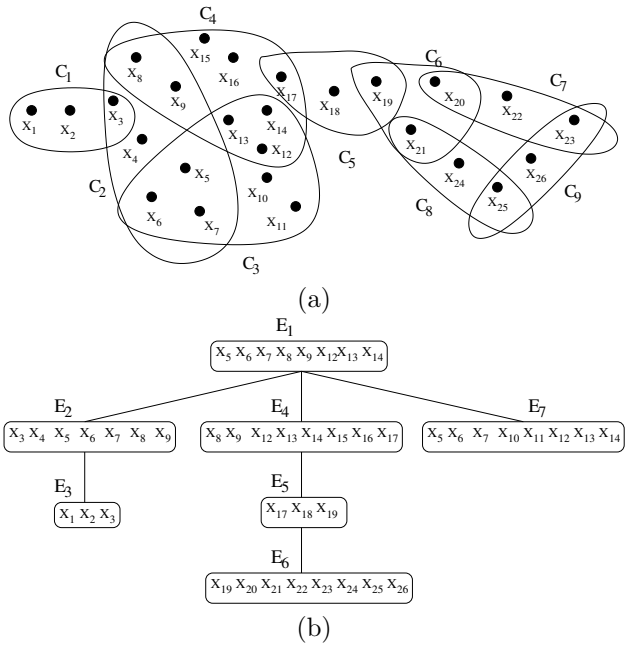


FIG. 3 – Un hypergraphe de contraintes (a) et l'une de ses décompositions arborescentes de largeur minimale (b) ($w = 7$).

cas général (voir [7]). Ici, nous donnons tout d'abord une des généralisations possibles de TC. Etant donnée une décomposition arborescente (E, T) associée à un CSP, le sous-problème associé à un cluster E_i est défini par le même ensemble de variables E_i mais nous considérerons maintenant à la fois les contraintes incluses dans le cluster, mais également celles qui intersectent E_i. Il est à noter que cette question ne se pose pas dans le cas binaire. Formellement, l'ensemble de contraintes pour un cluster E_i est $C_{E_i} = \{c_j \in C : c_j \cap E_i \neq \emptyset\}$. Les relations associées à ces contraintes sont données par $R_{E_i} = \{r_j[c_j \cap E_i] : c_j \in C_{E_i}\}$. Chaque sous-problème (cluster) est alors résolu indépendamment et à l'issue de cette étape, TC résout l'ensemble du CSP comme il le ferait pour tout CSP acyclique. Le coût de la première étape est borné par le coût d'obtention de toutes les solutions des sous-problèmes. Cette complexité a en général (en supposant que l'on dispose d'une décomposition arborescente de largeur w) été évaluée par $O(S.m^{w+1})$, puisque la taille maximum des clusters est $w+1$, en supposant que le coût pour l'énumération des solutions sur les clusters est $O(S.m^{w+1})$. De plus, le nombre maximum de solutions dans chaque cluster est également borné par $O(m^{w+1})$, et ainsi, le coût de la dernière étape est borné par $O(S.m^{w+1})$.

Le théorème 4 nous permet de proposer une autre borne de complexité, sous l'hypothèse parfaitement fondée où nous utiliserions une procédure telle que nFC2$_m$ pour la résolution de chaque cluster. Le coût

de résolution d'un cluster E_i est alors $O(S_i.r^{k_i})$, où S_i est la taille du sous-problème associé à E_i, alors que $k_i = k_{(E_i,C_{E_i})}$ (i.e. le paramètre associé à un recouvrement minimum de E_i). Il est à noter que la taille de l'ensemble des solutions dans E_i est bornée par $O(r^{k_i})$. Ainsi, le coût total de résolution de l'ensemble du CSP décomposé est alors $O(S.r^k)$ où $k = max\{k_i : i \in I\}$. Nous obtenons maintenant une nouvelle borne :

Théorème 5 *La complexité en temps de la résolution par décomposition arborescente d'un CSPs est* $O(S.r^k)$.

Notons que sur le plan pratique, ce résultat pourrait s'avérer utile, permettant de mieux appréhender l'intérêt de TC, dans la mesure où il est souvent observé que $k \ll n$.

3.2 Une nouvelle comparaison entre complexité des décompositions arborescente et hyperarborescente

Dans [10], Gottlob, Leone et Scarcello présentent une comparaison théorique qui prend en compte l'ensemble des méthodes structurelles de résolution de CSP. Il s'agit d'un résultat central dans le domaine dans la mesure où cette comparaison s'exprime par la donnée d'une hiérarchie qui met en évidence les méthodes et leur efficacité de résolution. Plus précisément, ce résultat est exprimé en termes de classes de problèmes que ces méthodes peuvent résoudre en temps polynomial, ce qui ne change finalement rien au propos. Soient D_1 et D_2 deux méthodes de décomposition.

- **Généralisation** : D_2 généralise D_1 si chaque problème traitable (au sens *tractable*, c'est-à-dire traitable en temps polynomial) en utilisant D_1 l'est également en utilisant D_2.
- **Est battue par** : D_1 est battue par D_2 s'il existe une classe de problèmes traitables en utilisant D_2, mais ne l'est pas en utilisant D_1.
- **Généralisation forte** : D_2 généralise fortement D_1 si D_2 généralise D_1 et D_2 bat D_1.
- **Equivalence** : D_1 et D_2 sont équivalentes si D_1 généralise D_2 et D_2 généralise D_1.

Cette hiérarchie, présentée dans la figure 7 permet à [10] de montrer que la décomposition hyperarborescente généralise fortement les autres méthodes, dont notamment Tree-Clustering.

Cette notion de décomposition hyperarborescente peut être assimilée à une généralisation aux hypergraphes de celle de décomposition arborescente.

Définition 3 *Etant donné un hypergraphe* $H = (X, C)$*, un hyperarbre pour l'hypergraphe* H *est un triplet* (T, χ, λ) *où* $T = (N, F)$ *est un arbre enraciné, et* χ *et* λ *sont des fonctions d'étiquetage qui associent à chaque sommet* $p \in N$ *deux ensembles* $\chi(p) \subset X$ *et* $\lambda(p) \subset C$*. Si* $T' = (N', F')$ *est un sous-arbre de* T*, nous définissons* $\chi(T') = \cup_{v \in N'} \chi(v)$*. Nous notons l'ensemble de sommets* N *de* T *par sommets*(T)*, et la racine de* T *par racine*(T)*. De plus, pour chaque* $p \in N$*,* T_p *désigne le sous-arbre de* T *enraciné en* p*.*

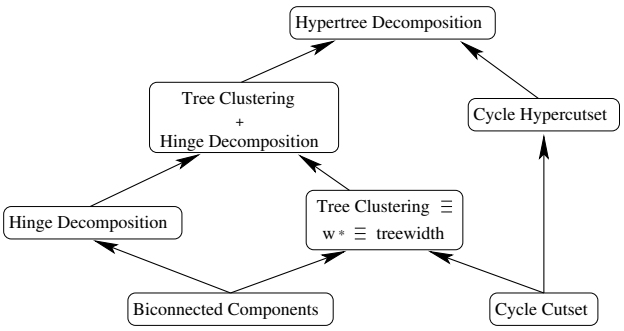


FIG. 4 – La hiérarchie de Gottlob, Leone et Scarcello [10]. Un arc (D_1,D_2) signifie que D_2 généralise fortement D_1.

Définition 4 *Etant donné un hypergraphe* $H = (X, C)$*, une décomposition en hyperarbre, ou décomposition hyperarborescente de* H *est un hyperarbre* $HD = (T, \chi, \lambda)$ *pour* H *qui satisfait les conditions suivantes :*

(i) pour chaque arête $c \in C, \exists p \in sommets(T)$ *tel que* $c \subset \chi(p)$*,*

(ii) pour chaque sommet $x \in X$*, l'ensemble* $\{p \in sommets(T) : x \in \chi(p)\}$ *induit un sous-arbre (connexe) de* T*,*

(iii) pour chaque $p \in sommets(T), \chi(p) \subset \cup_{c \in \lambda(p)} c$*,*

(iv) pour chaque $p \in sommets(T), \cup_{c \in \lambda(p)} c \cap \chi(T_p) \subset \chi(p)$*.*

Une arête $c \in C$ *est fortement couverte dans* HD *s'il existe* $p \in sommets(T)$ *tel que* $c \subset \chi(p)$ *et* $c \in \lambda(p)$*. Une décomposition en hyperarbre* HD *est une décomposition complète de* H *si chaque arête de* H *est fortement couverte dans* HD*. La largeur* h *d'une décomposition en hyperarbre* $HD = (T, \chi, \lambda)$ *est* $max_{p \in sommets(T)} |\lambda(p)|$*. La largeur d'hyperarbre* h^* *de* H *est la largeur minimum parmi toutes ses décompositions en hyperarbre.*

On peut remarquer que les hypergraphes acycliques sont précisément les hypergraphes possédant un hyperarbre de largeur 1. Pour la résolution de CSP, nous considérerons uniquement les décompositions en hyperarbre complètes. La figure 5 présente une décomposition en hyperarbre complète de l'hypergraphe de la figure 3(a).

Une décomposition en hyperarbre associe une décomposition arborescente à un recouvrement par hyperarêtes de chaque cluster. L'objectif est alors de réduire la taille des ensembles recouvrants (le nombre d'arêtes recouvrantes par hyperarête pour chaque cluster). Ainsi, une instance de CSP peut être résolue en $O(S.r^h)$ avec h la largeur d'une décomposition en hyperarbre de son hypergraphe de contraintes. La méthode proposée dans [10], consiste à calculer un CSP acyclique équivalent en résolvant chaque cluster grâce aux jointures des relations associées aux contraintes du recouvrement du cluster. Une fois ce traitement réalisé, la méthode procède en résolvant de façon classique, le CSP acyclique obtenu.

De plus, notons qu'il existe une classe de problèmes pour lesquels la largeur d'hyperarbre est bornée alors que sa largeur d'arbre ne l'est pas. En accord avec ce constat, on peut en déduire, pour la hiérarchie des méthodes structurelles, que la décomposition en hyperarbre généralise fortement Tree-Clustering. Néanmoins, nous montrons que ce résultat est faux sur la base d'une nouvelle analyse de la complexité temporelle du Tree-Clustering exploitant les résultats de la section précédente. En fait, nous montrons que le Tree-Clustering et la résolution par décomposition en hyperarbre sont équivalents au sens de la hiérarchie.

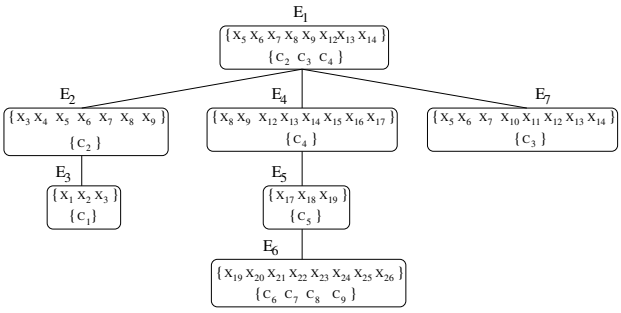


FIG. 5 – Le meilleur hyperarbre induit ($h = 4$) pour la décomposition en arbre optimale de la figure 3 (b).

D'une part, nous prouvons que si un CSP peut être résolu en $O(S.r^h)$ grâce à la décomposition en hyperarbre de son hypergraphe, avec h sa largeur, il peut également l'être avec la même complexité en temps, en utilisant TC sur une décomposition arborescente induite par la décomposition en hyperarbre. Soient $\mathcal{P} = (X, D, C, R)$ un CSP, $H = (X, C)$ son hypergraphe de contraintes et $HD = (T, \chi, \lambda)$ une décomposition en hyperarbre de H dont la largeur est h. (T, χ) vérifie toutes les conditions requises pour constituer une décomposition arborescente, excepté le fait qu'il puisse exister un cluster $\chi(p)$ contenu dans un autre $\chi(p')$, avec $p, p' \in sommets(T)$. Pour calculer une décomposition arborescente (T, χ), il suffit de plonger chaque cluster $\chi(p)$ dans un cluster $\chi(p')$ le contenant. Les arêtes dans T joignant $\chi(p)$ à d'autres nœuds les connecteront à $\chi(p')$. La définition suivante permet de calculer une décomposition arborescente de ce type en s'appuyant sur la notion d'arbre de jointure de l'hypergraphe H [7] qui est un arbre dont les nœuds sont les hyperarêtes de H telles que si un sommet x est présent dans deux nœuds, alors x est dans tous les nœuds sur la chaîne les connectant. En outre, H admet un arbre de jointure si et seulement si H est acyclique [1].

Définition 5 *Etant donnée une décomposition en hyperarbre $HD = (T, \chi, \lambda)$ de $H = (X, C)$, nous définissons $TD_{HD} = (E, T')$ comme suit :*

(i) $E = \{\chi(p), p \in sommets(T) | \nexists p' \in sommets(T), \chi(p) \subsetneq \chi(p')\}$

(ii) $T' = (I', F')$ *est un arbre de jointure de l'hypergraphe* (X, E).

L'hypergraphe (X, E) possède un arbre de jointure car il est acyclique. Nous avons donc :

Propriété 1 *TD_{HD} est une décomposition arborescente.*

Dans la figure 6, nous avons une décomposition en hyperarbre (c) de l'hypergraphe de la figure 3(a) et sa décomposition arborescente induite (b). Ainsi, puisque HD définit un recouvrement des ensembles $\chi(p), \forall p \in sommets(T)$ dont la taille est au plus h, la taille de tout recouvrement minimum d'un cluster quelconque de TD_{HD} vaut également au plus h.

Théorème 6 *$k \leq h$ avec $k = max\{k_{(E_i, C_{E_i})} : i \in I'\}$.*

Corollaire 1 *La complexité en temps de TC pour résoudre $\mathcal{P}$ est $O(S.r^h)$.*

Ce résultat prouve que TC est au moins aussi bon que la résolution par décomposition en hyperarbre.

Pour donner une autre illustration de ce résultat, nous considérons maintenant l'exemple proposé dans [10] appelé $Circle(x, y)$ où x est le nombre d'hyperarêtes et $2y$ le nombre de sommets par hyperarête (la figure 17 dans [10] montre une représentation de $Circle(x, 2)$). Cet exemple illustre la preuve du théorème central (théorème 23) définissant la hiérarchie dans [10]. Il est facile de voir que la largeur d'hyperarbre vaut 2 alors que la largeur d'arbre associée à l'hyperarbre vaut $3y - 1$, c'est-à-dire une largeur non bornée. En effet, la figure 18 de [10] qui présente une décomposition de $Circle(x, y)$ en hyperarbre de largeur 2 indique une décomposition arborescente dont la largeur vaut $3y - 1$. Néanmoins, en appliquant nFC2_m dans TC, le corollaire précédent nous

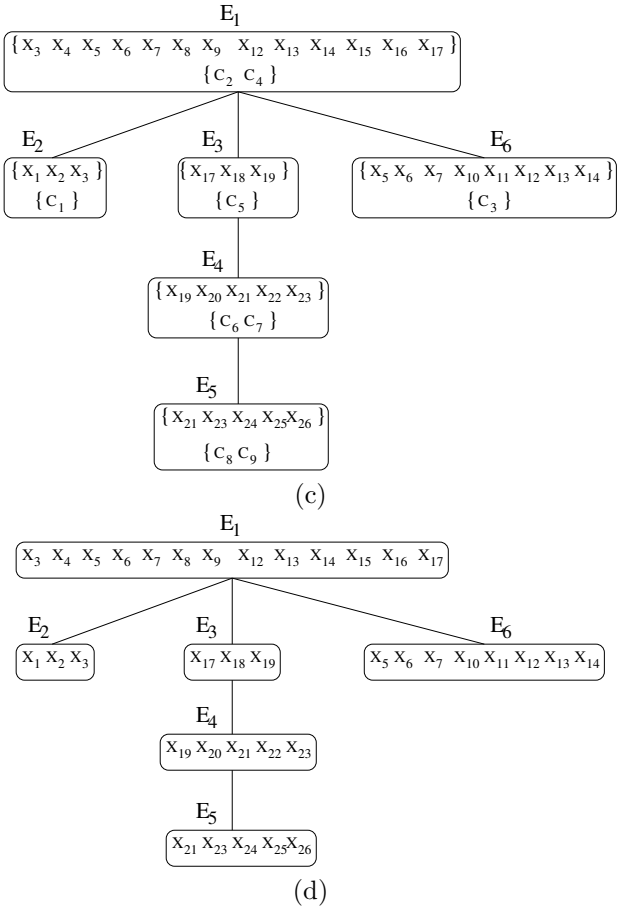


FIG. 6 – (c) Une décomposition en hyperarbre optimale ($h = 2$) de l'hypergraphe de la figure 3(a) et (d) sa décomposition arborescente induite qui ne l'est pas ($w = 12$).

indique que la complexité sera donnée par le nombre de contraintes qui recouvrent les clusters, c'est-à-dire au plus 2, et non par le nombre de variables dans les clusters qui vaut $3y$. Ainsi, la complexité de TC sera $O(S.r^2)$ comme pour HTD et non $O(S.m^{3.y})$ comme on le considère classiquement.

Le même type de raisonnement peut être utilisé pour montrer l'équivalence entre ces deux méthodes. Ainsi, nous prouvons que si un CSP peut être résolu en $O(S.r^k)$ à l'aide de TC sur une décomposition arborescente de son hypergraphe, avec k comme taille maximum pour les recouvrements de clusters, il peut également être résolu avec la même complexité en temps en utilisant une décomposition en hyperarbre induite par la décomposition arborescente. Soit $\mathcal{P} = (X, D, C, R)$ un CSP, $H = (X, C)$ son hypergraphe de contraintes et $TD = (E, T)$ une décomposition arborescente de H dont la taille maximum pour les recouvrements de clusters vaut k. Si nous associons un recouvrement minimum $\lambda(p)$ à chaque cluster $E_p \in TD$, avec $p \in sommets(T)$, nous calculons une décomposition en hyperarbre.

Définition 6 *Etant donnée une décomposition arborescente $TD = (E, T)$ de $H = (X, C)$, nous définissons $HD_{TD} = (T, \chi, \lambda)$ comme suit :*

(i) $E = \chi$

(ii) $\lambda = \{$*recouvrement minimum de* E_p *dans* $\{c \in C | c \cap \chi(T_p) \subset E_p\}, \forall p \in sommets(T)\}$.

Propriété 2 *HD_{TD} est une décomposition en hyperarbre.*

La décomposition en hyperarbre de la figure 5 est induite par la décomposition arborescente de la figure 3(b). Néanmoins, cet hyperarbre n'est pas nécessairement complet. Pour le rendre complet, nous devons rajouter, pour chaque hyperarête c qui n'est pas fortement couverte, un fils à un nœud p tel que $c \subset E_p$. Soit HDc_{TD} la décomposition en hyperarbre complète obtenue.

Théorème 7 *HD_{TD} et HDc_{TD} ont une largeur $h = k$.*

Corollaire 2 *Etant donnée HDc_{TD}, la complexité en temps de la résolution basée sur la décomposition en hyperarbre de $\mathcal{P}$ est $O(S.r^k)$.*

En résumé, nous avons prouvé que les décompositions arborescente ou en hyperarbre sont équivalentes dans la hiérarchie des méthodes structurelles. Aussi, nous proposons dans la figure 7 la correction de cette hiérarchie.

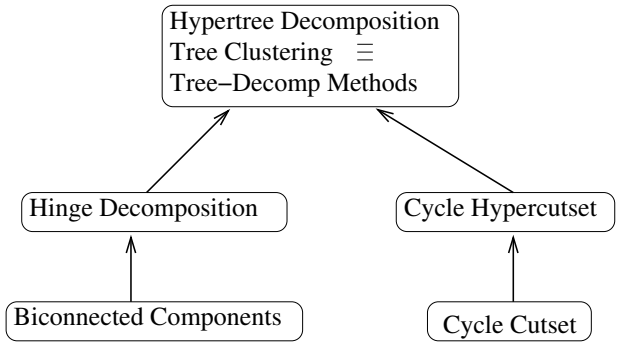


FIG. 7 – La hiérarchie corrigée. Un arc (D_1,D_2) signifie que D_2 généralise fortement D_1.

4 Discussion

Nous avons présenté, dans la section précédente, une correction de la hiérarchie existant entre les méthodes

structurelles, et qui montre que TC est finalement au même niveau que les méthodes fondées sur la décomposition en hyperarbre. L'un des intérêts majeurs de ce résultat réside dans le fait que la complexité en temps de TC est en fait liée à la largeur d'hyperarbre, un paramètre issu de cette décomposition. Bien que TC n'ait pas besoin de s'appuyer sur un recouvrement minimum des clusters de la décomposition arborescente considérée, il capture cependant ses propriétés par le simple usage de techniques de filtrage. En fait, la décomposition par hyperarbre est réellement intéressante car elle met en lumière le manque de précision de paramètres tels que la tree-width, pour estimer la qualité d'une décomposition arborescente pour le cas de la résolution de CSP. La largeur d'hyperarbre donne la meilleure borne de complexité théorique en temps parmi les différents paramètres d'origine structurelle. Cependant, en dépit de son intérêt théorique, la méthode de résolution de CSP par décomposition en hyperarbre n'a jamais fourni de résultats pratiques intéressants pour la résolution de CSPs. Les méthodes basées sur une décomposition arborescente (par exemple BTD [16]) sont significativement plus efficaces en pratique et sont désormais au même niveau sur le plan théorique. Ceci est dû à plusieurs inconvénients de l'approche de résolution par décomposition en hyperarbre.

Le premier porte sur le fait que les relations doivent être définies en extension. Pour certains problèmes, ceci peut être impossible du fait de la quantité d'espace mémoire requis qui s'avère prohibitive.

Pour TC, les relations peuvent être exprimées en intention, ce qui peut conduire à une économie d'espace mémoire considérable. Un autre inconvénient majeur porte sur la jointure de relations, opération qui peut s'avérer extrêmement coûteuse en espace. A l'opposé, pour TC, une stratégie de type backtrack permet de s'affranchir de cette contrainte de place. De plus, il y a de nombreux algorithmes, définis dans les trente dernières années, qui permettent un calcul relativement efficace de décompositions arborescentes [13, 15]. Un tel effort doit également être réalisé au niveau de la décomposition par hyperarbre, dans la mesure où les méthodes qui en proposent sont construites, à la base, sur le même principe que celles calculant des décompositions arborescentes. Elles s'appuient d'abord sur un calcul de décomposition arborescente, puis nécessitent un travail additionnel pour déterminer les recouvrements des clusters. Nous démontrons ici que ce travail est alors inutile dans la mesure où les bornes de complexité en temps d'une résolution par décomposition arborescente seront alors identiques à celles d'une résolution par décomposition en hyperarbre.
Cette nouvelle hiérarchie devra être complétée par l'intégration de la méthode hybride composée de TC et de la décomposition Hinge. Une étude est nécessaire pour situer cette méthode dans la hiérarchie aux vues des nouveaux résultats de complexité présentés dans cet article.
Par ailleurs, de nouvelles questions se présentent. Est-ce qu'une décomposition en hyperarbre optimale induit une décomposition arborescente optimale et réciproquement ? Les figures 3 (b), 5 et 6 apportent un début de réponse. Ils permettent de déduire que ce n'est pas le cas entre deux décompositions optimales quelconques. En effet, la meilleure décomposition en hyperarbre induite (figure 5) de la décomposition arborescente optimale de la figure 3 (b) n'est pas optimale. De même, la décomposition en hyperarbre optimale de la figure 6 (c) induit une décomposition arborescente (figure 6 (d)) qui n'est pas optimale. De ce fait, si ce lien existe, il doit relier une décomposition en hyperarbre particulière parmi toutes celles qui sont optimales et une décomposition arborescente particulière parmi, également, celles qui sont optimales. Ainsi, comment peut-on calculer une bonne décomposition arborescente en termes de taille maximum des recouvrements de clusters sans calculer au préalable une décomposition en hyperarbre de qualité ? Cette question est ouverte et des travaux futurs pourraient peut-être offrir des éléments de réponse.
Ces travaux devraient s'intéresser aux Spread Cut decompositions et plus généralement aux décompositions encadrées [3]. Les décompositions encadrées donnent un cadre générique qui capturent un grand nombre des décompositions existantes notamment les décompositions arborescentes et en hyperarbre. Ce cadre a également permis de définir une nouvelle décomposition : le spread cut. Cette nouvelle décomposition est au même niveau que la décomposition en hyperarbre dans la hiérarchie des décompositions. Par ailleurs, [3] a prouvé qu'il n'existe aucune autre classe de recouvrements encadrés plus puissante que ces deux. Cependant, sur certains graphes les spread cut decompositions peuvent être de meilleure qualité au sens de la largeur des recouvrements.

5 Conclusion

Dans un premier temps, nous avons montré que la complexité d'un algorithme tel que nFC2 (backtracking + filtrage) peut s'exprimer par $O(S.r^k)$, où r est la taille maximum des relations associées aux contraintes, k un paramètre structurel du CSP et S la taille du CSP. Précédemment, cette complexité était essentiellement exprimée par $O(S.m^n)$ avec m la taille des domaines, et n le nombre de variables. Ce résultat nous a permis de modifier la hiérarchie présentant l'efficacité temporelle de résolution de CSP par méthodes

de décompositions introduite dans [10]. En effet, nous pouvons maintenant affirmer que les méthodes de résolution de CSP basées sur la décomposition arborescente sont finalement au même niveau (le sommet) de la hiérarchie que celles fondées sur la décomposition en hyperarbre. Ce résultat fournit une explication théorique plus conforme aux observations expérimentales obtenues ses dernières années dans la communauté. Parmi les poursuites naturelles à ces travaux, l'une des pistes les plus intéressantes pourrait porter sur l'amélioration de l'efficacité pratique des méthodes opérant par décomposition, en réalisant de meilleurs calculs de décomposition arborescente qui intégreraient également la qualité de la décomposition en hyperarbre induite.

Remerciements

Ce travail a été soutenu par un programme blanc de l'ANR (projet STAL-DEC-OPT).

Références

[1] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30 :479–513, 1983.

[2] C. Bessière, P. Meseguer, E. C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141 :205–224, 2002.

[3] D. Cohen, P. Jeavons, and M. Gyssens. A Unified Theory of Structural Tractability for Constraint Satisfaction and Spread Cut Decomposition. In *IJCAI'05*, pages 72–77, 2005.

[4] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126 :5–41, 2001.

[5] S. de Givry, T. Schiex, and G. Verfaillie. Décomposition arborescente et cohérence locale souple dans les CSP pondérés. In *Proceedings of JFPC'06*, 2006.

[6] R. Dechter. Bucket Elimination : A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2) :41–85, 1999.

[7] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

[8] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.

[9] M.R. Garey and D.S. Johnson. Computer and Intractability. In *Freeman*, 1979.

[10] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[11] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.*, 64(3) :579–627, 2002.

[12] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

[13] P. Heggernes. Minimal triangulations of graphs : A survey. *Discrete Mathematics*, 306-3 :297–317, 2006.

[14] J. Huang and A. Darwiche. A Structure-Based Variable Ordering Heuristic for SAT. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1167–1172, 2003.

[15] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, pages 777–781, 2005.

[16] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

[17] P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proc. of ECAI*, pages 196–200, 2004.

[18] W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proceedings of ICTAI*, pages 542–548, 2004.

[19] R. Marinescu and R. Dechter. Dynamic Orderings for AND/OR Branch-and-Bound Search in Graphical Models. In *Proc. of ECAI*, pages 138–142, 2006.

[20] I. Rish and R. Dechter. Resolution versus Search : Two Strategies for SAT. *Journal of Automated Reasoning*, 24 :225–275, 2000.

[21] N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 :309–322, 1986.

[22] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, pages 125–129, 1994.

[23] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of CP*, pages 709–723, 2003.