Sur la génération et l'exploitation de décompositions pour la résolution de réseaux de contraintes

Philippe Jégou, Samba Ndiaye, Cyril Terrioux

Sur la génération et l'exploitation de décompositions arborescentes pour la résolution de réseaux de contraintes

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

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

Résumé

Les méthodes exploitant les décompositions arborescentes pour résoudre des réseaux de contraintes semblent constituer les meilleures approches en termes de complexité théorique en temps. Néanmoins, on peut estimer qu'elles n'ont pas démontré, à ce jour, un véritable intérêt pratique. Aussi, dans cette contribution, nous étudions tout d'abord différentes stratégies d'approximations de décompositions arborescentes optimales, et nous analysons ensuite leur pertinence dans le cadre de la résolution de CSP. Dans une seconde partie, nous étudions le problème du choix de la meilleure stratégie pour le parcours de l'arbre de clusters associé dans le cadre de la résolution du CSP. Les stratégies sont relatives notamment au choix du cluster racine, c'est-à-dire, celui à partir duquel débutera la recherche. Un deuxième aspect concerne l'ordre dans lequel les fils d'un cluster seront visités.

Abstract

Methods exploiting the tree-decomposition notion seem to provide the best approach for solving constraint networks w.r.t. the theoretical time complexity. Nevertheless, they have not shown a real practical interest yet. So, in this paper, we first study several methods for computing an approximate optimal tree-decomposition before assessing their relevance for solving CSPs. Then, we propose and compare several strategies to achieve the best depth-first traversal of the associated cluster tree w.r.t. CSP solving. These strategies concern the choice of the root cluster (i.e. the first visited cluster) and the order according to which we visit the sons of a given cluster.

1 Introduction

Le formalisme CSP (Constraint Satisfaction Problem) offre un cadre puissant pour la représentation et la résolution efficace de nombreux problèmes. Formuler un problème en termes de CSP consiste en la définition d'un ensemble X de variables, qui doivent chacune être affectées dans leur domaine (fini) respectif, tout en satisfaisant un ensemble C de contraintes qui expriment des restrictions sur les différentes affectations possibles. Une solution est une affectation de toutes les variables qui satisfait toutes les contraintes. Le problème d'existence de solution est NP-Complet.

Dans cet article, nous considérons également une extension du modèle CSP tel que défini ci-dessus, et qui permet de prendre en compte, notamment, les CSP "sur-contraints", c'est-à-dire, les CSP ne possédant pas de solution. Dans ce cas, le problème revient à déterminer une affectation des variables qui satisfait de manière optimale les contraintes, selon des critères à définir. Cette extension est généralement connue sous le nom de CSP Valués [11, 5, 24] et conduit à la définition de problèmes qui sont encore plus difficiles que les CSP de base.

La méthode usuelle pour la résolution de CSP est basée sur la recherche de type "backtracking". Pour le rendre efficace en pratique, le backtracking doit exploiter à la fois des techniques de filtrage comme FC ou MAC [23], ainsi que des heuristiques de choix de variables et/ou de valeurs. Ces approches conservent une complexité théorique en temps qui demeure ex-

ponentielle, à savoir $O(e.d^n)$ où n est le nombre de variables, e le nombre de contraintes, et d la taille maximum des domaines. Aussi, de très nombreux travaux ont été développés, afin de fournir des bornes de complexité théorique de meilleure qualité, et qui prendraient en compte certaines caractéristiques des instances. La meilleure borne connue à ce jour, est fournie par la valeur du paramètre généralement noté w, et qui est appelé la "tree-width" du CSP (on pourrait traduire le terme par "largeur arborescente du réseau"). Ce paramètre est relatif à certaines propriétés topologiques du graphe représentant les interactions entre variables via les contraintes. La borne de complexité en temps correspondante sera de la forme $O(n.d^{w+1})$. Pour atteindre cette borne, différentes méthodes ont été proposées dont le Tree-Clustering [9] et ses différentes extensions (on peut consulter [15] qui présente une étude récapitulative sur ces différentes méthodes ainsi que leur comparaison théorique). Ces méthodes sont fondées sur la notion de décomposition arborescente du réseau de contraintes. Ce type d'approche revient à représenter, d'un certain point de vue, le réseau de contraintes par des regroupements de variables (clusters), dont l'agencement est structuré en arbre. La meilleure décomposition, en théorie, conduit à une complexité en temps de $O(n.d^{w+1})$. Selon l'instance, le gain effectif en temps de calcul par rapport à une approche de type backtracking, peut être considérable. Toutefois, la complexité en espace, linéaire pour les méthodes de type backtracking, peut rendre l'approche par décomposition totalement inopérante. Cette complexité peut être réduite à $O(n.s.d^s)$ où s est la taille maximum des intersections entre clusters [8]. Ces résultats théoriques sont valables également pour les CSPs Valués [27]. De nombreux travaux, fondés sur cette démarche, ont été réalisés, mais il s'agit essentiellement de contributions théoriques, au sens où l'intérêt pratique n'est jamais attesté par une validation expérimentale. Hormis, [16], et à un degré moindre [14], il n'existe donc pas de travaux attestant de l'utilité de ces approches.

L'objectif du présent travail est d'explorer cette direction, en proposant des voies visant à l'optimisation pratique de ces méthodes, afin de les rendre effectivement opérationnelles en pratique.

Dans la première partie de cet article, nous étudions à la fois, le problème, NP-difficile, de la recherche d'une décomposition arborescente optimale, et l'intérêt qu'il peut y avoir à le résoudre. Ce problème, souvent appréhendé par le biais de la recherche d'une *triangulation optimale*, a fait l'objet de très nombreux travaux. Il semble que les méthodes produisant des solutions exactes ne constituent toutefois pas les approches les plus intéressantes. En effet, nous montrerons ici qu'il est en général préférable de se limiter au calcul d'approximations de décompositions arborescentes optimales plutôt que de rechercher à tout prix une solution optimale. Ceci, déjà parce que le coût du calcul d'une telle solution peut vite se révéler prohibitif. De plus, et de façon plus surprenante encore, disposer d'une décomposition optimale ne garantit en aucun cas de disposer de la meilleure décomposition au regard de la résolution du CSP qui lui est associé. En effet, la notion d'optimalité telle qu'elle a été considérée dans le cadre de la théorie des graphes, n'intègre que le critère structurel de l'instance traitée, ignorant ainsi une part considérable de la sémantique associée au CSP traité. Par exemple, la meilleure approximation semblerait être celle directement issue de l'algorithme MCS dû à [26], algorithme qui n'est pas réputé pour sa qualité d'approximation de w.

Dans la seconde partie de cette contribution, nous étudions le problème relatif au parcours de la décomposition arborescente lors de la résolution du CSP. Nous essayons de mettre en évidence les meilleures stratégies pour guider la recherche. Par stratégie, nous entendons choix du cluster racine pour débuter la recherche, et choix de l'ordre dans lequel les clusters fils seront considérés lors de la poursuite de l'énumération. Nous avons ainsi exhibé différents critères pour le choix de la racine. Ces critères sont fondés sur la topologie de l'arbre associé à la décomposition. Par exemple, nous montrons que pour résoudre le problème CSP (au sens existence d'une solution), plus la racine est proche du *centre de l'arbre*, plus la résolution sera rapide. Un autre critère a été mis en évidence. Il concerne la difficulté de résolution propre à chaque cluster, et il nous a permis de montrer que plus le cluster racine était difficile à résoudre, plus l'efficacité globale de la résolution était assurée.

Notons que cette étude a été réalisée en utilisant la méthode BTD [16] qui semble constituer l'approche la plus efficace, et sans doute même, la seule méthode raisonnablement opérationnelle pour la résolution de CSP par décomposition arborescente de graphes, du moins à notre connaissance.

Cet article est organisé comme suit. Dans la section 2, nous rappelons les notions de base propres aux méthodes exploitant la décomposition arborescente de graphe. La section 3 analyse les méthodes et algorithmes calculant des décompositions arborescentes, tandis que la section 4 propose et évalue différentes stratégies et donc heuristiques, pour guider la recherche dans l'arbre associé à la décomposition. En outre, cette section présente différents résultats expérimentaux qui montrent l'intérêt de l'emploi de telles heuristiques lors de l'exploration de cet arbre. Dans la dernière section, nous traçons les perspectives qui

s'offrent pour la poursuite de nos travaux.

2 Préliminaires

Un *problème de satisfaction de contraintes* (CSP) peut être défini par un triplet (X, D, C). X est alors un ensemble $\{x_1, \ldots, x_n\}$ de n variables. Chaque variable x_i prend ses valeurs dans un domaine fini associé d_{x_i} figurant dans D. Les affectations des variables doivent satisfaire des contraintes définies dans C. Etant donnée une instance du triplet (X, D, C), le problème CSP consiste à déterminer s'il existe une affectation des variables qui satisfait chaque contrainte. Ce problème est NP-Complet. Dans cet article, et sans manque de généralité, nous considérerons uniquement les contraintes binaires, à savoir les contraintes qui ne portent que sur des couples de variables. Dans un tel cas, la structure d'un CSP peut être représentée par le graphe (X, C), qui est appelé *graphe de contraintes*. Les sommets de ce graphe correspondent alors aux variables de X et les arêtes entre couples de sommets correspondent à l'existence de contraintes entre les couples de variables associées aux sommets.

L'approche de référence pour la résolution de CSP s'appuyant sur la structure du graphe de contraintes est le Tree-Clustering [9]. Cette méthode est basée sur la notion de décomposition arborescente de graphes, notion formellement définie par Robertson et Seymour dans [21].

Définition Etant donné un graphe $G = (X, C)$, une *décomposition arborescente* de G est une paire $(E, \mathcal{T})$ avec $\mathcal{T} = (I, F)$ un arbre et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (ou cluster) E_i est un noeud de $\mathcal{T}$ et vérifie :

1. $\cup_{i \in I} E_i = X$,
2. pour toute arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et
3. pour tout $i, j, k \in I$, si k figure sur un chemin de i à j dans $\mathcal{T}$, alors $E_i \cap E_j \subseteq E_k$

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

La complexité en temps du Tree-Clustering est $O(n.d^{w+1})$. Malheureusement, l'obtention d'une décomposition arborescente optimale, à savoir une décomposition arborescente dont la largeur est w, constitue un problème NP-difficile [3]. Aussi, de nombreux travaux ont été développés afin de résoudre ce problème. Ces travaux exploitent généralement une approche fondée sur la notion de graphe *triangulé* (voir [13] pour une introduction aux graphes triangulés). Un

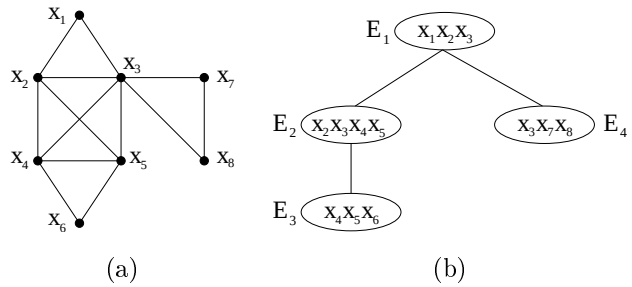

FIG. 1 – Un graphe de contraintes sur 8 variables (a) et l'une de ses décompositions arborescentes (b).

graphe est dit triangulé s'il tolère un *ordre d'élimination parfait*, c'est-à-dire un ordre sur les sommets $\sigma = (x_1, \ldots, x_n)$ tel que le voisinage ultérieur de tout sommet x_i (sommets x_j voisins figurant après x_i dans l'ordre σ) forme une clique.

Le lien entre graphes triangulés et décomposition arborescente est évident. En effet, étant donné un graphe triangulé G, l'ensemble des cliques maximales $E = \{E_1, E_2, \ldots, E_k\}$ de G correspond à la famille de sous-ensembles associée à la décomposition arborescente. Comme tout graphe G n'est pas nécessairement triangulé, une décomposition arborescente de G peut être obtenue par une triangulation de G, c'est-à-dire une opération qui calculera un graphe triangulé G' à partir de G. Plus précisément, nous appelons *triangulation* l'ajout à G d'un ensemble d'arêtes C' tel que $G' = (X, C \cup C')$ soit triangulé. La largeur d'arborescence de G' est alors égale à la taille de la plus grande clique du graphe résultant G' moins un. La largeur d'arborescence (tree-width) de G est alors égale à la plus petite largeur d'arborescence pour toutes les triangulations de G.

Généralement, lors de l'exploitation des décompositions arborescentes pour la résolution de CSP, on considère des approximations de triangulations optimales. Ainsi, la complexité en temps est alors $O(n.d^{w^+ + 1})$ où $w^+ + 1$ est la taille du plus grand cluster (ou clique) et nous avons $w + 1 \leq w^+ + 1 \leq n$. La complexité en espace est alors $O(n.s.d^s)$ où s est la taille maximale des *séparateurs*, c'est-à-dire la taille maximum des intersections entre paires de clusters (on a nécessairement $s \leq w^+$). L'une des conséquences immédiates de l'existence de ces bornes de complexité est que, pour garantir *a priori* de bonnes implémentations, il faut impérativement minimiser les valeurs de w^+ ainsi que de s.

Le graphe de la figure 1(a) est déjà triangulé. La plus grande clique est de taille 4. Par conséquent, la tree-width de ce graphe vaut 3. Dans la partie (b) de cette figure est représenté un arbre dont les noeuds

correspondent aux cliques maximale du graphe triangulé; cet arbre constitue une décomposition arborescente possible pour le graphe (a). Ici nous avons $E_1 = \{X_1, X_2, X_3\}$, $E_2 = \{X_2, X_3, X_4, X_5\}$, $E_3 = \{X_4, X_5, X_6\}$, et $E_4 = \{X_3, X_7, X_8\}$.

Dans la littérature, les meilleurs résultats obtenus en pratique par des approches de ce type l'ont été avec une méthode hybride appelée BTD [16]. De façon surprenante, les résultats présentés d'abord dans [16] puis après dans [17], ont été obtenus sans qu'aucune heuristique liée à la forme de la décomposition ne soit utilisée dans l'implémentation de BTD. Par heuristique, nous entendons d'une part, le calcul ou l'approximation d'une décomposition arborescente qui minimise la valeur du paramètre w, et d'autre part, les stratégies utilisées pour le parcours de la décomposition arborescente, à savoir le choix de la racine (cluster de départ), et l'ordre d'examen des fils durant la recherche. Cet aspect est essentiel car de ces différents choix découleront l'ordre d'affectation des variables, dont on connaît l'importance pour garantir l'efficacité de toute méthode de résolution.

3 Calcul de décompositions

3.1 Critères graphiques

Dans cette section, nous considérons le problème de la recherche de "bonnes" décompositions arborescentes en termes de "bonnes" triangulations. Plusieurs algorithmes et approches ont été proposés pour les triangulations. Dans tous les cas, l'objectif consiste à minimiser soit le nombre d'arêtes ajoutées, soit la taille des cliques dans le graphe triangulé. On peut classer les approches possibles en 4 classes :

1. **Triangulations optimales.** Du fait que le problème traité est NP-difficile, nous ne connaissons à ce jour, aucun algorithme dont la complexité serait polynomiale. Aussi, certains auteurs, comme [25], ont proposés des algorithmes dont la complexité en temps est exponentielle. Plus récemment, [10] ont proposés un algorithme en $O(n^4.(1.9601^n))$. Le problème posé par ce type d'algorithmes est évidemment qu'ils sont d'un intérêt pratique nécessairement limité. Par exemple, l'algorithme présenté dans [10] n'a pas été implémenté car il serait inutile, en l'état actuel, d'en attendre une quelconque utilité pratique [28]. De plus, dans un article récent, [12] ont à nouveau expérimenté l'algorithme de [25] (dans sa version de type "décision", c'est-à-dire calcul de tree-width), et ils ont observé que même pour des petits graphes (50 sommets et 100 arêtes), l'implémentation de Shoikhet et Geiger n'aboutit pas ("did not terminate" [12]). Finalement, et jusqu'à présent, seule l'approche de [12] qui s'appuie sur un algorithme de type *branch and bound*, semble constituer une voie prometteuse pour le calcul de triangulations optimales. Nous l'avons d'ailleurs utilisée dans le cadre de nos expérimentations, mais aucun résultat le concernant ne sera reporté ici, du fait de son inefficacité pratique.
2. **Algorithmes d'approximation.** Ces algorithmes garantissent l'approximation d'un optimum par un facteur constant. Leur complexité est généralement polynomiale en temps (dans la tree-width) [1]. Malheureusement, pour le moment, cette approche semble également irréaliste. En effet, la complexité du dernier algorithme de Amir est $O(n^3.log^4(n).k^5).log(k))$ avec de plus une constante cachée (au sens de la notation "grand O") supérieure à 850 (d'après [6]). De plus, Amir indique dans ses expérimentations (voir [2]) que sur les benchmarks testés, son algorithme prend de 6 minutes à 6 jours, selon l'instance, alors même qu'une heuristique naïve comme *min-degré* (voir plus loin) dont l'implémentation prend une dizaine de minutes, donne sur les mêmes benchmarks respectivement de 1 seconde à 2 minutes, avec de plus des résultats dont la qualité est significativement meilleure (l'approximation de la largeur d'arborescence est de l'ordre de 50 % inférieure) !
3. **Triangulation minimale.** Une triangulation minimale, au sens de l'inclusion, calcule un ensemble C' tel que pour tout sous-ensemble $C" \subset C'$, le graphe $G' = (V, C \cup C")$ n'est pas triangulé. Il faut préciser qu'une triangulation peut être minimale sans être optimale (minimum). L'intérêt de ce type d'approche est relatif notamment à l'existence d'algorithmes de complexité polynomiale. Par exemple [22] (LEX-M) et plus récemment [4] (LB), ont présentés des algorithmes en $O(ne')$ où e' est le nombre d'arêtes figurant dans le graphe après sa triangulation.
4. **Triangulation heuristique.** Ces approches construisent en général un ordre (dynamiquement) et ajoutent des arêtes dans le graphe, de sorte qu'en fin de traitement, l'ordre obtenu soit un ordre d'élimination parfait pour le graphe résultant G'. La complexité de ces méthodes est en général polynomiale (souvent même linéaire) mais elles n'offrent, en contrepartie, aucune garantie de minimalité. Cette approche serait justifiée en pratique [18]. En effet, Kjærulff a observé entre autres, qu'au contraire des résultats attendus, des heuristiques de ce type produisent des triangulations raisonnablement proches de l'optimum. De

Instance	n	e	n-LEX-M		n-LB		n-min-fill		n-MCS	
			w	tps	w	tps	w	tps	w	tps
CELAR06	100	350	11	0,42	11	0,37	11	0,58	11	0,37
CELAR07	200	817	19	4,42	18	3,80	16	6,44	18	4,20
CELAR08	458	1655	20	55,85	19	82,73	16	73,57	19	51,74
CELAR09	340	1130	18	39,96	18	38,89	16	31,43	19	36,36
GRAPH05	100	416	28	1,00	26	0,68	25	1,34	31	0,97
GRAPH06	200	843	58	15,56	53	7,92	54	19,64	58	15,65
GRAPH11	340	1425	106	146,16	90	39,63	91	162,90	104	150,13
GRAPH12	340	1256	99	140,09	85	45,19	85	148,28	96	142,62
GRAPH13	458	1877	146	558,38	120	115,43	126	710,06	131	640,62

TAB. 1 – Tree-width obtenue après triangulation et temps de triangulation (en s) pour des graphes des problèmes "CALMA".

plus, elles sont en général très faciles à implémenter. Plusieurs heuristiques sont généralement considérées, et la complexité en temps de leur implémentation varie de $O(n + e')$ à $O(n(n + e'))$ pour la plus coûteuse (*min-fill*). En voici quelques unes :

- *MCS*. Elle s'appuie sur l'ordre calculé par l'algorithme de [26] qui a pour objet la reconnaissance des graphes triangulés.
- *heuristique min-degré-interieur*. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui possède le plus petit nombre de voisins déjà numérotés.
- *min-degré*. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui possède le plus petit nombre de voisins non numérotés.
- *min-fill*. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui conduira à ajouter un minimum d'arêtes si l'on complète le sous-graphe induit par ses voisins non encore numérotés.

3.2 Évaluation expérimentale des critères graphiques

Jusqu'à présent, les expérimentations présentées dans la littérature semblent indiquer que les approches 1 (recherche d'optimalité) et 2 (approximation) n'ont que très peu d'intérêt pratique car leur temps d'exécution, comme première étape d'une résolution de CSP, est bien trop coûteux par rapport au profit qu'elle pourrait offrir en termes d'approximation de la largeur d'arborescence. Aussi, nous pensons qu'il est préférable de concentrer nos effort sur les approches 3 et 4. Pour évaluer l'intérêt de tels algorithmes, nous avons choisi de les expérimenter sur deux types de benchmarks : des graphes issus de véritables applications, et des graphes générés aléatoirement mais dont on sait qu'il s'agit de sous-graphes de graphes possédant de bonnes propriétés en termes de décomposition arborescente. Il est clair que travailler sur des graphes aléatoires classiques n'a strictement aucun intérêt dans le contexte présent, et ce pour deux raisons. Tout d'abord, les méthodes de résolution de CSP par décomposition n'ont d'intérêt que dans le cadre de problèmes réels qui posséderaient de bonnes propriétés structurelles, ce qui n'est pas le cas des graphes aléatoires classiques. Par ailleurs, il est bien connu que la tree-width de tels graphes est démesurée par rapport à leur densité. Par exemple, pour $n = 50$ avec une densité de 0.25, la tree-width observée dans [12] est de l'ordre de 27.

Dans la table 1, nous présentons des résultats expérimentaux réalisés sur les graphes de l'archive CALMA (problèmes d'allocations de fréquences [7]).[1] Nous avons testé quatre algorithmes de triangulations : *n-LEX-M*, *n-LB*, *n-min-fill* et *n-MCS*. Ces algorithmes sont définis à partir d'une adaptation des heuristiques présentées plus haut (*LEX-M*, *LB*, *min-fill* et *MCS*). Précisément, chaque algorithme *n-X* revient à fixer le choix du premier sommet à considérer puis à utiliser l'heuristique X pour ordonner tous les autres sommets. Afin d'éviter un choix peu judicieux pour le premier sommet, les algorithmes *n-X* vont essayer successivement chacun des n sommets comme premier sommet. Ces résultats indiquent que les meilleures approches sont issues de deux algorithmes : *n-LB* et *n-min-fill*. À titre indicatif, un choix aléatoire du premier sommet conduit à des résultats évidemment moins bons, mais souvent très voisins, avec des temps de calcul de l'ordre du temps reporté dans la table 1 mais bien sûr divisé par n (aucun des temps n'excède alors les 2 secondes).

Notons que les résultats obtenus avec ces triangulations sont généralement meilleurs que ceux obtenus

[1] Toutes les expérimentations sont réalisées sur un PC sous Linux doté d'un Pentium IV 2,4 GHz et de 512 Mo de RAM.

avec l'heuristique MSVS [19], qui est basée sur une approche radicalement différente d'obtention de décompositions arborescente (elle est basée sur les algorithmes de flots).

La table 2 présente des résultats de triangulations obtenues sur des graphes aléatoires générés en utilisant un modèle similaire à celui présenté dans [12]. Dans ce modèle, nous considérons des k-arbres à n sommets, générés aléatoirement (il s'agit ainsi de graphes triangulés dont toutes les cliques sont de taille $k+1$). Etant donné un k-arbre, nous supprimons p % ($p = 20, 40$ ou 60) de ses arêtes. Pour chaque k-arbre partiel ainsi obtenu, nous avons appliqué les quatre algorithmes de triangulations *n-LEX-M*, *n-LB*, *n-min-fill* et *n-MCS*. Nous constatons ici que l'approche la plus intéressante en termes de largeur d'arborescence, est fournie par *n-min-fill*, alors que *n-LB* offre un compromis intéressant entre temps d'exécution et qualité du résultat final.

3.3 Triangulation et résolution

Puisque l'intérêt d'une décomposition, dans le cadre de la résolution de CSP, est relatif à l'efficacité de la dernière étape de traitement, à savoir, la résolution effective du CSP, nous avons complété nos expérimentations sur les quatre triangulations étudiées plus haut, mais en intégrant, après l'étape de triangulation, l'étape de résolution. Pour mener ces expérimentations, nous avons utilisé des CSP aléatoires dont la structure - le graphe - est cependant issue des problèmes de la liste CALMA, donc d'instances de graphes réelles (cf. table 3). De manière surprenante, nous avons observé que la décomposition la plus intéressante est donnée par *MCS*, *n-min-fill* se révélant seulement robuste, plutôt qu'efficace, car terminant dans tous les cas. De plus, quand les décompositions sont modifiées après la triangulation, de sorte à limiter la taille maximale s des séparateurs, l'influence de la triangulation tend à disparaître. Notons que pour des questions d'efficacité pratique, nous avons tout intérêt à réduire la valeur de s, ceci étant possible en regroupant les clusters qui partagent de grandes intersections.

Nous avons également étudié l'effet de la triangulation sur la résolution de CSP structurés aléatoires. Pour ce faire, nous avons repris le modèle introduit dans [16] et qui permet la génération de CSP en utilisant un arbre de cliques aléatoire, dont la taille des cliques et celle des séparateurs sont bornées. Une fois le graphe de contraintes généré, nous avons supprimé, comme pour les k-arbres partiels, un certain pourcentage p d'arêtes ($p = 10, 20, \ldots$). Pour les instances de CSP (problème de décision cf. table 4), la surprise vient du fait que MCS, qui semble constituer *a priori* l'approche la moins intéressante fournit en définitive la meilleure, surpassant même LB, à la fois au regard des critères graphiques, mais surtout en termes de résolution. MCS obtient la meilleure approximation de w tout en limitant d'emblée la valeur de s. Cela lui confère une robustesse remarquable, meilleure que celle offerte par les autres triangulations. Cette tendance est confirmée pour le cas de VCSP (plus précisément Max-CSP) dont le graphe de contraintes est généré dans les mêmes conditions (cf. table 5).

Pour conclure, on peut estimer que le problème de l'obtention d'une "bonne" décomposition, au sens de son utilité pour la résolution de CSP, valués ou non, demeure ouvert. Néanmoins, des éléments nouveaux sont apparus ici, qui peuvent maintenant mieux nous guider dans le traitement de cette question. En effet, dans cette section, plusieurs pistes sont ouvertes.

Tout d'abord, nous avons pu observer que la recherche d'une triangulation optimale n'est pas nécessairement l'objectif le plus pertinent si l'on triangule en vue de résoudre un CSP. Ainsi, des heuristiques exploitables en temps polynomial pourraient s'avérer suffisantes pour calculer des décompositions arborescentes utiles. Ce fait nous semble pratiquement avéré, quand on limite la question à la résolution de CSP au sens existence ou recherche de solution. En effet, il nous est apparu qu'en l'état actuel, le recours à une triangulation coûteuse, qui rechercherait l'optimalité, ne recèle strictement aucun intérêt. Du fait du coût élevé de telles approches, et de la relative efficacité de la résolution ultérieure du CSP, une fois la décomposition obtenue, quand bien même celle-ci serait de qualité moyenne, le gain éventuel engendré ne justifie en aucun cas l'emploi d'algorithmes de triangulation trop gourmands en temps. Par contre, pour le cas des CSP Valués, la question demeure ouverte. Ainsi, nous estimons qu'un effort doit être porté sur l'étude de triangulations de meilleure qualité dans le cas des CSP Valués. La résolution de ceux-ci entraîne souvent un temps de calcul considérable, tel qu'il justifierait vraisemblablement le recours à une phase de pré-traitement conséquente, pour autant que celle-ci se révèle ensuite profitable à la résolution.

Un autre fait observé ici concerne l'importance de la valeur de s. Effectivement, ce paramètre semble constituer un critère extrêmement sensible pour ce qui concerne la résolution ultérieure du CSP. Les tables 3 et 4 fournissent en ce sens, une illustration remarquable puisque des triangulations comme LEX-M ou LB ne permettent pas à la phase de résolution d'être efficace si la valeur de s n'est pas maîtrisée (cf. table 3). Par contre, dès lors que la valeur de ce paramètre est limitée, LEX-M et LB peuvent se situer à un niveau comparable des autres triangulations, mais ce constat n'est pas général (cf. table 4).

n	p	n-LEX-M		n-LB		n-min-fill		n-MCS	
		w	tps	w	tps	w	tps	w	tps
50	20	10,60	0,02	10,20	0,03	10,02	0,14	13,96	0,01
50	40	11,70	0,02	10,28	0,03	10,00	0,11	15,00	0,01
50	60	12,52	0,02	10,72	0,02	9,96	0,06	14,68	0,01
100	20	11,86	0,18	10,24	0,34	10,22	2,15	13,92	0,06
100	40	13,52	0,22	10,44	0,32	10,30	1,47	16,06	0,06
100	60	15,34	0,19	10,30	0,28	10,08	0,79	17,44	0,05
200	20	13,60	1,33	10,66	2,17	10,62	36,99	14,64	0,32
200	40	17,78	2,20	10,68	2,90	10,72	29,61	17,48	0,42
200	60	22,98	1,86	10,66	2,80	10,48	12,09	19,48	0,36

TAB. 2 – Tree-width obtenue après triangulation et temps de triangulation (en s) pour des k-arbres partiels aléatoires.

Instance	d	t	Temps pour s illimité				Temps pour s limité à 10			
			LEX-M	LB	min-fill	MCS	LEX-M	LB	min-fill	MCS
CELAR02	50	1216	2,50	2,51	16,55	2,48	2,50	2,54	15,73	2,57
CELAR03	30	373	1,96	56,68	6,35	1,91	1,71	2,44	1,62	1,32
CELAR06	50	1155	3,17	3,20	3,12	3,15	3,17	3,20	3,12	3,15
CELAR07	25	209	11,91	M	16,14	3,86	4,13	4,01	4,23	3,91
CELAR09	25	209	M	T	16,33	5,73	4,10	4,08	4,02	3,92

TAB. 3 – [CSP] Temps d'exécution (en s) pour la résolution de CSP. D'abord pour une taille s quelconque de séparateur et pour une taille limitée à 10. T et M indiquent qu'il est impossible de résoudre les CSP correspondants, pour des raisons soit de Temps, soit d'encombrement de la Mémoire.

4 Quelle stratégie pour parcourir l'arbre ?

Sans heuristiques de qualité dans le choix des variables (et des valeurs) pendant une résolution de type backtracking telle que FC ou MAC, il serait pratiquement impossible de résoudre quelque CSP que ce soit. Pour les méthodes basées sur la décomposition arborescente de graphes, l'heuristique qui choisirait les variables durant la recherche, correspond à une heuristique ou stratégie de parcours de l'arbre associé à la décomposition. En effet, l'heuristique doit choisir la racine de l'arbre (que nous appellerons "cluster racine"), et cette heuristique doit également ordonner les fils des noeuds de l'arbre, afin de choisir le cluster sur lequel la recherche doit se poursuivre. L'objet de cette section est de proposer et d'étudier de telles heuristiques.

4.1 Choix de la racine

Pour choisir la racine, nous disposons au moins de 2 critères. Tout d'abord, nous pouvons considérer un *critère local* qui a pour objet de d'évaluer la pertinence du choix d'un cluster, indépendamment de ses interactions avec les autres clusters. Par exemple, nous pouvons considérer la taille (qui sera noté **Taille** dans la suite) du cluster comme critère local. Comme autre critère, nous pouvons considérer un *critère global* qui permettra d'opérer un choix de cluster relativement à sa situation dans l'arbre. Pour formaliser ce second critère, nous allons utiliser la notion de *distance* entre couples de sommets x et y dans un graphe G, que nous noterons $dist(x, y)$, et qui correspond à la longueur du plus court chemin entre x et y dans G.

Définition. Soit $G = (X, E)$ un graphe.

1. Un sommet $x \in X$ est appelé **centre (CTR)** de G si x minimise $max\{dist(x, y) : y \in X\}$.
2. Un sommet $x \in X$ est appelé **barycentre (BARY)** de G si x minimise $\Sigma_{y \in X} dist(x, y)$.
3. Un sommet $x \in X$ est dit **périphérique (PERI)** dans G si x maximise $max\{dist(x, y) : y \in X\}$.
4. Un sommet $x \in X$ est dit **fortement périphérique (FPERI)** dans G si x maximise $\Sigma_{y \in X} dist(x, y)$.

Notons que, étant donné un arbre, le problème consistant à déterminer ces différents sommets peut être résolu en $O(n^2)$. Pour cela, il suffit de calculer la matrice associée aux distances entre tous les couples de sommets. Ceci est possible en déterminant la distance entre un sommet x et tous les autres sommets à partir d'une recherche en largeur dans le graphe - un arbre - en débutant à partir de x. Puisque pour les

p	t	LEX-M		LB		min-fill		MCS	
		w	tps	w	tps	w	tps	w	tps
10%	215	18,50	65,00	16,80	4,30	15,97	19,13	14,03	3,98
20%	237	22,00	243,50	20,63	7.25	16,30	9,38	14,00	3,87
30%	257	23,30	72,34	21,07	21,01	17,02	5,83	15,03	4,13
40%	285	24,90	76,55	22,30	72,60	15,33	1,17	15,33	4,97

TAB. 4 – [CSP] Temps d'exécution (en s) et valeur de w pour la classe $(150, 25, 15, t, 5, 15)$ après retrait de $p\%$ arêtes (avec s limité à 5).

p	t	LEX-M		LB		min-fill		MCS	
		w	tps	w	tps	w	tps	w	tps
10%	103	10,70	7,23	9,70	6,49	10,23	14,76	9,00	3,72
20%	110	11,27	28,77	9,60	2,42	11,87	118,43	9,00	2,52
30%	119	13,00	T	9,76	6,24	10,93	28,97	9,00	3,80

TAB. 5 – [VCSP] Temps d'exécution (en s) et valeur de w pour la classe $(75, 15, 10, t, 2, 10)$ après retrait de $p\%$ arêtes (avec s limité à 5).

arbres, le nombre d'arêtes est exactement $n-1$, le coût engendré par cette recherche en largeur sera borné par $O(n)$. Il suffit alors de répéter cette opération à partir de chacun des n sommets de G, d'où la complexité $O(n^2)$.

Si l'arbre que nous considérons est celui associé à une décomposition arborescente, ces définitions doivent être étendues. Une façon naturelle de procéder dans ce cas consiste à utiliser la taille des clusters E_x associés aux noeuds x de l'arbre. Dans les définitions données ci-dessus, pour chaque sommet y considéré, on remplace alors $dist(x, y)$ par sa valeur pondérée, soit précisément $dist(x, y).|E_y|$.

La table 6 présente les résultats obtenus sur des VCSP qui ont été générés en utilisant un arbre de cliques aléatoire, dont la taille des cliques et celle des séparateurs sont bornées. Le modèle est identique à celui présenté dans le détail dans [17].

Classe	Temps d'exécution en secondes			
(n, d, r, t, s, ns)	Taille	BARY	PERI	FPERI
(75,15,10,98,2,10)	25,34	23,06	48,77	70,09
(100,10,15,30,3,15)	40,98	63,42	523,72	454,75
(125,10,15,30,3,20)	72,78	86,03	460,77	448,17
(150,10,15,29,3,25)	73,26	139,36	280,19	278,63

TAB. 6 – [VCSP] Choix d'une racine.

Dans la table 6, on peut observer que les choix de racine "périphérique" ou "fortement périphérique" sont à éviter. Ce constat est renforcé encore quand on voit que l'approche "fortement périphérique" (**FPERI**), est encore plus mauvaise que la simple approche "périphérique" (**PERI**). A l'opposé, deux stratégies, qui ne sont d'ailleurs pas antagonistes, sont à envisager : prendre comme racine, soit le cluster "barycentre", soit le cluster de plus grande taille de l'arbre, ce qui semble d'ailleurs conduire à un choix encore meilleur. Ici, l'intuition semble concorder avec l'observation. En effet, on peut considérer que l'un ou l'autre de ces clusters est choisi de sorte à privilégier en début de résolution, une partie du problème qui sera la plus contrainte possible, soit localement (cf. **Taille**), soit globalement (cf. **BARY**), en ce sens que dans un cas comme dans l'autre, ces choix conduiront à sélectionner un cluster qui, globalement, se situe dans une zone du problème qui est plus contrainte que les autres. On retrouve ici le fameux principe dit du "first fail", qui conduit à privilégier en début de résolution les choix les plus contraints.

4.2 Ordre dans la filiation

Etant donnée une racine, nous avons étudié deux critères pour ordonner le traitement des clusters fils. D'une part, nous avons considéré leur taille : nous pouvons les choisir dans l'ordre décroissant (**PGC**) ou croissant (**PPC**) des tailles. D'autre part, nous avons ordonné les clusters par rapport à la taille du séparateur avec le père, en débutant avec le plus petit et dans l'ordre croissant (**SEP**). Pour les expérimentations, nous avons utilisé le même modèle de CSP aléatoires que celui présenté pour le choix de la racine. Afin de s'assurer de la pertinence des choix réalisés, nous avons également reporté les temps de calcul pour le cas où aucun ordre dans le choix des fils n'est considéré (**NO**).

Pour les résultats expérimentaux, nous n'avons pas

Classe	Temps d'exécution			
(n,d,r,t,s,ns)	NO	PGC	PPC	SEP
(150,25,15,200,5,15)	7,58	7,33	5,98	6,01
(150,30,15,300,5,15)	16,91	16,57	11,69	11,69
(150,30,20,250,5,15)	49,46	49,53	36,83	37,02
(200,25,15,198,5,20)	9,74	9,81	6,53	6,51
(200,25,14,204,5,25)	3,23	3,29	2,55	2,57
(225,25,14,205,5,30)	2,59	2,61	1,91	1,95

TAB. 7 – [CSP] Ordonnancement des fils.

considéré les VCSP, car l'optimisation qu'entraîne ces problèmes impose en pratique le parcours de toute la filiation. Ces résultats (cf. table 7) montrent que la taille des clusters est le critère le plus pertinent, en considérant l'ordre croissant de leur taille. En effet, contrairement au choix de la racine, pour lequel, plus le cluster est gros, plus la résolution sera efficace, dans le choix de la filiation, il y a tout intérêt à privilégier le cluster le plus petit. Ce critère est compatible avec l'ordre croissant appliqué pour la taille des séparateurs.

5 Discussion et Conclusion

Cet article avait pour objet l'étude des heuristiques visant à améliorer la résolution de CSP par les méthodes exploitant les décompositions arborescentes de réseaux de contraintes. Nous avons vu qu'une telle étude, qui était difficilement envisageable auparavant, revêt maintenant un intérêt considérable, notamment pour la résolution de problèmes difficiles recelant des propriétés structurelles. A titre de comparaison, il faut savoir ici, que les instances qui ont servies à nos expérimentations sont inaccessibles à des méthodes de type backtracking (FC ou MAC par exemple).

Dans un premier temps, nous avons fait le point sur les différentes stratégies d'approximation de décompositions arborescentes optimales disponibles. Ces méthodes sont par nature guidées par des critères essentiellement "graphiques" (c'est-à-dire structurels), puisqu'il s'agit de minimiser la largeur arborescente. Or, du fait notamment d'un coût en temps très élevé, celles-ci ne sont généralement pas les plus intéressantes pour résoudre le CSP associé. D'une part, elles conduisent à un prétraitement trop coûteux, et d'autre part, elles ne garantissent pas pour autant, une résolution systématiquement plus efficace. Par ailleurs, l'emploi d'algorithmes à la fois simples à implémenter, et efficaces en temps, qui fonctionnent en appliquant des heuristiques, conduisent à des calculs de décomposition à la fois intéressants au regard des critères graphiques, mais aussi du point de vue de la résolution du CSP. Il existe toutefois un critère purement graphique, qui par contre permet d'améliorer le temps de calcul. Il s'agit de la taille maximale des séparateurs dans la décomposition calculée. En effet, une maîtrise de ce paramètre permet de limiter le temps de calcul lors de la résolution. Il s'avère que ce constat est en contradiction avec la théorie (cf. complexité en temps), puisqu'elle tend à privilégier la minimisation de w et non de s.

Dans un second temps, nous avons étudié le problème du choix de la meilleure stratégie pour le parcours de l'arbre de clusters associé à une décomposition arborescente. Par stratégie, nous entendions tout d'abord choix du cluster racine, c'est-à-dire, du cluster d'où débutera une recherche, puis ordre dans lequel les fils d'un cluster seront visités. La conclusion à laquelle nous sommes arrivés conforte la célèbre heuristique dite du "first fail" ou "choix le plus contraignant". En effet, le choix du cluster le plus grand, ou bien le plus "central" du réseau, nous ont permis d'obtenir les meilleurs résultats. La notion de "centralité" ayant été formalisée ici par la notion de "barycentre d'un arbre" avec pondération des sommets. Concernant l'ordre à appliquer pour le choix des fils, deux heuristiques ont fournis les meilleurs résultats. Soit choisir les clusters fils dans l'ordre croissant de leur taille, soit dans l'ordre croissant des intersections avec le cluster père. Les résultats observés sont alors voisins, et ont permis un gain de plus de 30 % sur les données traitées.

Les gains entrevus ici, déjà conséquents, mais aussi les limites de cette étude, nous incitent à penser que ce travail doit être poursuivi. Deux axes forts doivent être notamment privilégiés. Tout d'abord celui relatif au calcul des décompositions arborescentes. La littérature existant sur ce thème, ainsi que les travaux en cours, abondent. Toutefois, la quasi totalité de ceux-ci n'appréhende pas la problématique sous l'angle qui nous semble le plus pertinent. En effet, à notre connaissance, seules des études intégrant les critères purement graphiques - essentiellement la largeur arborescente obtenue - font l'objet d'attention de la part de la communauté. Or, cet objectif nous semble biaisé, et les expérimentations que nous avons réalisées nous confortent dans cette opinion. En fait, l'effort devrait privilégier l'objectif réel de ces calculs de décompositions, à savoir la phase de résolution qui suivra, but essentiel de ce prétraitement. Le second axe qui doit être développé concerne la nécessité qu'il y a, pour une méthode telle que BTD, à s'affranchir de l'ordre d'énumération induit par la décomposition arborescente. En effet, la validité de la méthode BTD, que ce soit pour le test d'existence ou le calcul d'une solution, ou encore pour l'obtention de la meilleure solution en cas de CSP valué, impose l'application d'un ordre compatible

avec l'ordre partiel induit par un parcours de l'arbre de cliques. Aussi, pour exploiter au mieux la notion d'heuristique pour les choix de variables, il faudrait arriver à s'affranchir de cet ordre compatible, pour tendre à une libéralisation totale de l'ordre. Cette démarche, certes nouvelle, n'est cependant pas isolée. En effet, dans un article récent, Li et Van Beek [20] ont montré que l'exploitation de la structure d'un problème, si elle peut notamment se libérer de l'ordre d'affectation qui en découle, peut conduire à l'élaboration de méthodes extrêmement efficaces, qui comme BTD, allient exploitation de la structure, aux garanties d'efficacité offertes par les techniques structurelles du fait de l'existence de bornes de complexité théorique.

Références

[1] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *UAI'01 : Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 7–15, 2001.

[2] E. Amir. Approximation algorithms for treewidth, 2002. reason.cs.uiuc.edu/eyal/paper.html.

[3] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8 :277–284, 1987.

[4] A. Berry. A Wide-Range Efficient Algorithm for Minimal Triangulation. In *Proceedings of SODA'99 SIAM Conference*, january 1999.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proc. of IJCAI*, pages 624–630, 1995.

[6] V. Bouchitté, D. Kratsch, H. Muller, and I. Todinca. On treewidth approximations. *Discrete Appl. Math.*, 136(2-3) :183–196, 2004.

[7] CALMA. www.zib.de/koster/research/fap.en.html.

[8] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125 :93–118, 2001.

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

[10] F. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *ICALP*, pages 568–580, 2004.

[11] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70, 1992.

[12] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *UAI'04 : Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 201–208, 2004.

[13] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press. New-York, 1980.

[14] G. Gottlob, M. Hutle, and F. Wotawa. Combining hypertree, bicomp and hinge decomposition. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 161–165, 2002.

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

[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] U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark, 1990.

[19] A. M. C. A. Koster, H. L. Bodlaender, and C. P. M. van Hoesel. Treewidth : Computational Experiments. Technical Report 01–38, Berlin, Germany, 2001.

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

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

[22] D. Rose, R. Tarjan, and G. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on computing*, 5 :266–283, 1976.

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

[24] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : hard and easy problems. In *Proc. of IJCAI*, pages 631–637, 1995.

[25] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulation. In *Proc. of AAAI*, pages 185–190, 1997.

[26] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3) :566–579, 1984.

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

[28] I. Todinca. Communication privée. 2005.