Heuristiques dynamiques pour l'exploitation de décompositions arborescentes de (W)CSP

Samba Ndojh Ndiaye

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Av. Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
samba-ndojh.ndiaye@univ-cezanne.fr

Résumé

Les méthodes exploitant les décompositions arborescentes pour résoudre des réseaux de contraintes semblent constituer les meilleures approches en terme de complexité théorique en temps. Néanmoins, les études menées dans le cadre des décompositions arborescentes avaient essentiellement pour but d'optimiser des critères graphiques. Toutefois, nous avons observé que ces critères graphiques n'étaient pas les plus pertinents pour une résolution efficace en pratique. De ce fait, nous menons une étude sur la qualité de ces décompositions et leur exploitation au niveau de l'efficacité pratique des méthodes de résolution structurelles. Et ici nous allons nous focaliser sur l'aspect exploitation.

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. La méthode usuelle pour la résolution de CSP est basée sur la recherche de type "backtracking" (FC, MAC [1]) avec une complexité théorique en temps de $O(e.d^n)$ (notée $O(exp(n))$) où n est le nombre de variables, e le nombre de contraintes, et d la taille maximum des domaines. La meilleure borne connue à ce jour, est fournie par la "tree-width" du CSP (généralement noté w) : $O(exp(n))$. On peut consulter [2] pour une étude récapitulative des différentes méthodes qui permettent d'atteindre cette borne telles que le Tree-Clustering [3] et ses extensions. Ces méthodes sont fondées sur la notion de décomposition arborescente du réseau de contraintes. 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 [4].
Le formalisme CSP a été étendu pour capturer des notions telles que la préférence, la possibilité et également quand il n'existe pas de solution au CSP, pour permettre de produire une affectation qui minimise un certain critère sur la satisfaction des contraintes. Ici, nous nous intéressons au formalisme WCSP qui une extension dans laquelle un poids est associé à chaque tuple de chaque contrainte. Ainsi, le coût d'une affectation est la somme des poids

des tuples qu'elle contient. Résoudre un WCSP consiste donc à trouver une affectation de coût minimum. Ce problème est NP-difficile. La méthode de résolution usuelle est basée sur le branch and bound qui, pour être efficace, utilise des techniques de filtrages et des heuristiques de choix de variables et de valeurs. Néanmoins, sa complexité théorique en temps est en $O(exp(n))$ pour une instance ayant n variables. D'autres méthodes, utilisant la programmation dynamique ([5, 6, 7, 8, 9, 10]), propose de meilleures bornes de complexité. Certaines exploitent la structure du problème ([6, 7, 11, 10]) et donnent une complexité en $O(exp(w+1))$ ([3], [2]).

De nombreux travaux, fondés sur l'utilisation de décompositions arborescentes, ont été réalisés, mais nous allons ici nous focaliser sur la méthode BTD [12] qui semble constituer l'approche la plus efficace pour la résolution de CSP et WCSP par décomposition arborescente de graphes. L'objectif du travail présent est d'améliorer ces méthodes d'un point de vue pratique en proposant des heuristiques pour une meilleure exploitation des décompositions de (W)CSP. Cela doit permettre de trouver une réponse à l'inadéquation entre la complexité théorique de ces méthodes et leur résultats pratiques catastrophiques.

Ce résumé 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 présente des stratégies pour une meilleure exploitation des décompositions. Enfin, la dernière section conclut ce résumé tout en donnant des pistes pour des travaux futurs.

2 Décomposition arborescente de (W)CSP

Dans ce résumé, et sans manque de généralité, nous considérerons uniquement les (W)CSPs binaires, à savoir ceux dont les contraintes ne portent que sur des couples de variables. Dans un tel cas, la structure d'un (W)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.

La méthode BTD (pour Backtracking with Tree-Decomposition) est basée sur la notion de décomposition arborescente de graphes, notion formellement définie par Robertson et Seymour dans [13]. 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 : (i) $\cup_{i \in I} E_i = X$, (ii) pour toute arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et (iii) 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.
BTD s'appuie sur la notion d'ordre compatible pour l'instanciation des variables [12]. La complexité en temps de BTD 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 [14]. Afin de résoudre ce problème, de nombreux travaux exploitent une approche fondée sur la notion de graphe *triangulé* (voir [15] pour une introduction aux graphes triangulés). Un 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

CSP (n, d, w, t, s, n_c, p)	w^+	s	Classe 3		Classe 4			
			B G	E G	B D	A F	B G	E G
$(150, 25, 15, 215, 5, 15, 10)$	13.0	12.2	2.45	5.34	2.75	2.17	2.08	2.65
$(150, 25, 15, 257, 5, 15, 30)$	12.1	11.4	4.97	3.55	1.41	1.05	1.13	1.30
$(250, 20, 20, 129, 5, 20, 30)$	16.8	15.8	6.19	7.84	33.93	4.61	4.41	34.20
$(250, 20, 20, 146, 5, 20, 40)$	15.9	15.2	4.51	17.99	11.38	3.17	3.17	10.63
$(250, 25, 15, 211, 5, 25, 10)$	13.0	12.3	13.37	18.12	5.86	7.71	6.65	6.44
$(250, 25, 15, 253, 5, 25, 30)$	12.3	11.8	5.14	5.26	2.80	3.71	3.52	3.06
$(250, 20, 20, 99, 10, 25, 10)$	17.9	17.0	M	M	66.94	63.15	62.99	66.33
$(500, 20, 15, 123, 5, 50, 10)$	13.0	12.5	7.31	7.54	5.48	4.50	4.41	5.86
$(500, 20, 15, 136, 5, 50, 20)$	12.9	12.1	27.01	15.11	4.86	4.92	3.94	5.24

TAB. 1 – Paramètres w^+ et s de la décomposition arborescente et durée d'exécution (en s) sur des CSP aléatoires structurés avec mdd_{dyn} pour les classes 3 et 4.

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 $O(m.d^{w^+ +1})$ 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.

3 Exploitation des décompositions arborescentes

Dans [16] et [17], nous avons étudié des stratégies pour une meilleure exploitation des décompositions arborescentes. En effet, sans heuristique pertinente dans les choix d'affectation des variables, les algorithmes basés sur le backtracking tels que FC ou MAC sont généralement dans l'impossibilité de résoudre efficacement nombre d'instances. Aussi, pour améliorer les performances d'une méthode telle que BTD, et au-delà, des méthodes à base de décomposition, il est nécessaire de s'appuyer sur des heuristiques performantes. Pour être efficaces, ces heuristiques doivent être dynamiques, c'est-à-dire que les choix opérés doivent être élaborés au gré de la recherche et non au préalable de façon statique et définitive. Nous avons donc défini plusieurs classes d'ordres sur les variables de plus en plus dynamiques mais qui garantissent de bonnes bornes de complexité. Les ordres de la classe 3 permettent un choix de variables totalement dynamique à l'intérieur des clusters et un choix dynamique du prochain cluster à visiter parmi les clusters fils du cluster courant. Ils permettent également de maintenir la borne de complexité de BTD.

De même, nous avons proposé deux nouvelles extensions pour la borne de complexité temporelle qui confère une importance accrue à l'apport et aux bénéfices des heuristiques([18]). En effet, là où la marge qui nous était laissée de choisir la prochaine variable devant être affectée était restreinte à w^+ variables au plus, nous disposons maintenant d'un choix parmi $w^+ + k$ variables, tout en conservant une borne de complexité théorique. Cette démarche permet d'assurer une meilleure maîtrise de l'espace requis en obtenant une complexité en espace de $O(n.s'.d^{s'})$ où $s' \leq s$. Ces extensions sont basées sur la notion de *décompositions arborescentes $k-$recouvrantes orientées* [18]. Suivant qu'on utilise une (Classe 4) ou plusieurs (Classe 5) *décompositions arborescentes $k-$recouvrantes orientées* lors de la résolution, nous disposons de nouvelles bornes de complexité en temps ($O(exp(w + k + 1))$ pour la classe 4 et $O(exp(2(w + k + 1) - s^-))$ pour la classe 5 ou s^- est la taille minimum des séparateurs) qui offrent une liberté accrue aux heuristiques de choix de variables.

Pour calculer des ordres de qualité dans ces différentes classes, nous avons élaboré plusieurs heuristiques. Les critères sur lesquels sont fondées ces heuristiques sont liés aux propriétés topologiques du réseau de contrainte et de sa décomposition, ainsi qu'aux traits

CSP	Classe 3		Classe 4			
	A	*B*	*B*	*A*	*A*	*B*
(n, d, w, t, s, n_c, p)	*F*	*G*	*D*	*C*	*F*	*G*
$(75, 10, 15, 30, 5, 8, 10)$	M	M	18.99	8.69	18.30	16.77
$(75, 10, 15, 30, 5, 8, 20)$	2.88	M	1.67	1.56	1.53	2.32
$(75, 10, 15, 33, 3, 8, 10)$	8.82	5.36	4.31	5.26	4.18	3.24
$(75, 10, 15, 34, 3, 8, 20)$	2.84	2.14	1.61	1.48	1.58	1.40
$(75, 10, 10, 40, 3, 10, 10)$	11.87	1.43	0.81	0.58	0.85	0.52
$(75, 10, 10, 42, 3, 10, 20)$	1.03	0.79	0.42	0.51	0.41	0.38
$(75, 15, 10, 102, 3, 10, 10)$	11.70	4.93	4.73	5.41	4.63	3.13
$(100, 5, 15, 13, 5, 10, 10)$	M	M	9.05	9.60	9.10	11.71

TAB. 2 – Durée d'exécution (en s) sur des WCSP aléatoires structurés avec mdd_{dyn} pour les classes 3 et 4.

sémantiques de l'instance traitée. De même, nous avons défini plusieurs heuristiques de fusion de clusters qui permettent de calculer des *décompositions arborescentes $k-$recouvrantes orientées*. Les tableaux 1 et 2 permettent de comparer la classe 3 et la classe 4 dans les cadres CSP et WCSP. Nous avons utilisé comme critère de fusion de clusters celui basé sur la taille des séparateur (on fusionne deux clusters si la taille de leur séparateur dépasse 5). Malgré l'augmentation de la taille des clusters, la classe 4 obtient les meilleurs résultats grâce à l'apport considérable de l'heuristique dynamique de choix de variables. En outre la maîtrise de s permet de limiter l'espace mémoire requis et ainsi de résoudre les instances CSP de la classe $(250, 20, 20, 99, 10, 25, 10)$ et les instances WCSP des classes $(75, 10, 15, 30, 5, 8, 10)$ et $(100, 5, 15, 13, 5, 10, 10)$ pour lesquelles les ordres de la classe 3 requièrent une trop grande quantité de mémoire (M). Il est tout de même nécessaire de faire un compromis entre la treewidth de la décomposition et la liberté donnée à l'heuristique dynamique de choix de variables et ainsi trouver la bonne valeur de k au delà de laquelle BTD ne tire plus assez profit de la structure du problème.

4 Conclusion

Nous avons présenté ici une étude qui a pour objectif de rendre opérationnelles les méthodes de résolutions structurelles de (W)CSP. En effet, il existe une inadéquation entre les bonnes bornes de complexité de ces méthodes et leurs performances très pauvres en pratiques. L'objectif est donc de leur pourvoir des heuristiques pertinentes pour une meilleure exploitation des décompositions arborescentes, et tout cela d'un point de vue pratique. Nous avons proposé des stratégies pour une meilleure exploitation des décompositions avec des heuristiques de choix de variables plus dynamiques tout en maintenant de bonnes bornes de complexité. Les expérimentations menées ont mis en lumière une amélioration très significative des performances de BTD. Des travaux futurs devraient permettre d'accroître cela avec de nouveaux critères pour fusionner les clusters et ainsi avoir de bonnes valeurs de k.

Références

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

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

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

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

[5] G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proceedings of AAAI*, pages 181–187, 1996.

[6] A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, November 1999.

[7] P. Meseguer and M. Sánchez. Tree-based Russian Doll Search. In *Proceedings of CP Workshop on soft constraint*, 2000.

[8] P. Meseguer and M. Sánchez. Specializing Russian Doll Search. In *Proceedings of CP*, pages 464–478, 2001.

[9] P. Meseguer, M. Sánchez, and G. Verfaillie. Opportunistic Specialization in Russian Doll Search. In *Proceedings of CP*, pages 264–279, 2002.

[10] J. Larrosa and R. Dechter. Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems. *Constraints*, 8(3) :303–326, 2003.

[11] J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-Tree Search with Soft Constraints. In *Proceedings of ECAI*, pages 131–135, 2002.

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

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

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

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

[16] P. Jégou, S. N. Ndiaye, and C. Terrioux. Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre. In *Actes des JFPC'2006*, pages 219–228, 2006.

[17] P. Jégou, S. N. Ndiaye, and C. Terrioux. Dynamic heuristics for branch and bound search on tree-decomposition of Weighted CSPs. In *Proceedings of Workshop Soft 2006*, 2006.

[18] P. Jégou, S. N. Ndiaye, and C. Terrioux. An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of CP*, 2006.