Un nouveau paramètre de graphes pour la résolution de CSP par décomposition *

Philippe Jégou **Cyril Terrioux**
Aix-Marseille Université, LSIS UMR 7296
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, cyril.terrioux}@lsis.org

Résumé

D'un point de vue théorique, les méthodes de décompositions (arborescentes) offrent une bonne approche lorsque la largeur (arborescente) des réseaux de contraintes (CSP) est petite. Dans ce cas, elles ont souvent montré leur intérêt pratique. Ainsi, la littérature, en provenance des Mathématiques ou de l'Intelligence Artificielle, a concentré l'essentiel de ses efforts à minimiser un seul paramètre, la largeur arborescente (ou tree-width). Néanmoins, des études expérimentales ont montré que ce paramètre n'est pas toujours le plus pertinent à considérer pour la résolution de CSP.

En particulier, dans cet article, nous montrons expérimentalement que les algorithmes de décomposition de l'état de l'art produisent des clusters (une décomposition arborescente est un arbre de clusters) ayant plusieurs composantes connexes. Ensuite, nous mettons en évidence que la présence de tels clusters crée un réel problème pour l'efficacité des méthodes de résolution. Pour éviter ce genre de problème, nous présentons ici un nouveau paramètre graphique dans le cadre des CSP, et qui est appelé la *Bag-Connected Tree-Width*, qui ne tient compte que des décompositions arborescentes dont chaque cluster est connexe, et qui sont appelées *Bag-Connected Tree-Decompositions*. Un premier algorithme polynomial calculant ces décompositions est proposé. Enfin, nous montrons expérimentalement que l'utilisation de ces Bag-Connected Tree-Decompositions améliore considérablement la résolution de CSP par les méthodes de décomposition.

*Ce travail est soutenu par l'Agence Nationale de la Recherche dans le cadre du projet TUPLES (ANR-2010-BLAN-0210).

1 Introduction

Les Problèmes de Satisfaction Contraintes (CSP) offrent un moyen assez puissant pour la formulation de problèmes en informatique, et en particulier en Intelligence Artificielle. Formellement, un *Problèmes de Satisfaction Contraintes* est un triplet (X, D, C), où $X = \{x_1, \ldots, x_n\}$ est un ensemble de n variables, $D = (D_{x_1}, \ldots, D_{x_n})$ est un ensemble de domaines finis de valeurs, un par variable, et $C = \{C_1, \ldots, C_e\}$ est un ensemble fini de e contraintes. Chaque contrainte C_i est une paire $(S(C_i), R(C_i))$, où $S(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ définit la *portée* ou *scope* de C_i, et $R(C_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ est sa *relation de compatibilité*. L'*arité* de C_i est notée $|S(C_i)|$. Un CSP est dit *binaire* si toutes ses contraintes sont d'arité 2. La structure d'un réseau de contraintes (autre terme pour appeler un CSP) est représentée par un hypergraphe (qui est un graphe dans le cas binaire), appelé *(hyper)graphe de contraintes*, et dont les sommets correspondent aux variables et les arêtes aux portées des contraintes. Dans cet article, pour une question de simplification, nous n'évoquerons que le cas des CSP binaires mais ce travail peut directement être étendu aux CSP non binaires en s'appuyant sur la *2-section* [2] de l'hypergraphe de contraintes (aussi appelée *graphe primal*), comme cela sera fait dans la partie expérimentale puisque nous y traiterons à la fois des CSP binaires et non binaires. De plus, et sans manque de généralité, nous supposerons que les réseaux considérés sont connexes. Pour simplifier les notations, dans la suite, nous noterons le graphe $(X, \{S(C_1), \ldots S(C_e)\})$ par (X, C). Une affectation sur un sous-ensemble de X sera dite *cohérente* si elle ne viole aucune contrainte. Vérifier si un CSP possède une *solution* (i.e. une affectation cohérente de toutes

les variables) est bien connu pour constituer un problème NP-complet. Aussi, de nombreux travaux ont été développés pour rendre la résolution d'instances très efficace en pratique, cela, en utilisant des formes optimisées de backtracking, des heuristiques, de l'apprentissage de contraintes, des techniques de backtracking non-chronologique, des techniques de filtrage basées sur la propagation de contraintes, etc. La complexité de ces approches demeure bien évidemment exponentielle, au moins en $O(n.d^n)$ où n est le nombre de variables et d la taille maximum des domaines.

Une autre approche de la problématique concerne l'étude des classes polynomiales qui s'appuient sur des propriétés des réseaux de contraintes. Il a notamment été montré que si la structure d'un CSP binaire est acyclique, il peut être résolu en temps linéaire [10]. En exploitant et en généralisant ces résultats théoriques, des méthodes de résolution de CSP ont été définies, comme par exemple le *Tree-Clustering* [7]. Ce type de méthodes s'appuie sur la notion de *décomposition arborescente* (*Tree-Decomposition*) de graphes [19]. Leur avantage fondamental est lié à leur complexité théorique, de l'ordre de d^{w+1} où w est la *largeur arborescente* (*Tree-Width*) du graphe de contraintes. Quand ce graphe possède de « belles » propriétés topologiques, à savoir quant w est « petit », ces méthodes permettent de résoudre des instances de grande taille. C'est le cas par exemple des problèmes bien connus d'allocation de fréquences radio [4]. Notons qu'en pratique, la complexité en temps est plutôt de l'ordre de $d^{w^+ +1}$ où $w^+ \geq w$ est en fait une approximation de la largeur arborescente car le calcul de décompositions arborescentes optimales (i.e. de largeur w) constitue un problème NP-difficile [1].

Toutefois, la mise mise en œuvre pratique de ce type de méthodes, bien qu'elle ait très souvent démontré son intérêt, a également permis de constater que la minimisation du paramètre w^+ n'est pas toujours la plus appropriée. Outre la difficulté du calcul de la valeur optimale de w^+, i.e. w, la manipulation de décompositions optimales ne conduit pas toujours à une résolution des plus efficaces en pratique. Cela a d'ailleurs mené à proposer des méthodes de décomposition de graphes qui rendent la résolution de CSP plus efficace (d'un point de vue pratique), mais pour lesquelles w^+ peut être significativement plus grand que w [14].

Dans cette contribution, nous montrons que l'une des raisons à ce manque d'efficacité dans la résolution de CSP par décomposition peut se trouver dans la nature même des décompositions pour lesquelles w^+ est proche de w. En effet, la minimisation de w^+ peut conduire à des décompositions pour lesquelles certains clusters recèlent plusieurs composantes connexes. Malheureusement, cette absence de connectivité conduit à des méthodes de résolution qui vont dépenser un temps considérable dans la résolution de sous-problèmes relatifs à ces clusters déconnectés, en passant beaucoup de temps à aller d'une composante connexe à une autre. Pour éviter ce problème, nous introduisons dans le cadre des CSP un nouvel invariant de graphe avec un paramètre appelé *Bag-Connected Tree-Width*. Ce paramètre est égal à la largeur minimale pour toutes les décompositions arborescentes pour lesquelles chaque cluster possède une unique composante connexe. Ces décompositions seront appelées *Bag-Connected Tree-Decompositions*. Ce paramètre qui a été introduit très récemment dans [18] [1] est égal à la largeur minimum pour toutes les Bag-Connected Tree-Decompositions. Nous démontrons ici que le calcul de ce paramètre est NP-difficile. Aussi, nous proposons un premier algorithme de complexité polynomiale en temps, précisément $O(n(n+e))$, dont l'objet est d'approximer ce paramètre à l'aide d'un calcul de décomposition associée. Les expérimentations que nous présentons montrent la pertinence de ce paramètre puisque sa prise en compte permet d'améliorer significativement la résolution de CSP par décomposition.

Notons que ce travail s'appuie sur la décomposition arborescente, mais qu'il pourrait très bien être adapté à d'autres méthodes de décompositions (par exemple [12, 13]). En effet, pour la plupart des méthodes de résolution de CSP à base de décomposition, les décompositions sont en fait calculées à l'aide d'algorithmes qui visent à approximer au mieux un paramètre graphique (la largeur) sans prendre en compte ni la connectivité des clusters produits, ni d'ailleurs la phase de résolution qui va suivre. Aussi, les problèmes observés ici avec la décomposition arborescente peuvent également se présenter pour d'autres formes de décompositions.

La partie suivante introduit les principes des méthodes de résolution de CSP par décomposition arborescente. La partie 3 met en évidence certains des problèmes inhérents au calcul de « bonnes » décompositions tandis que la partie 4 introduit la notion de Bag-Connected Tree-Decomposition, tout en proposant un premier algorithme qui permet d'en calculer. Avant de conclure, nous présentons des résultats expérimentaux qui montrent l'intérêt que recèle l'exploitation de cette notion pour la résolution pratique de CSP.

2 Résolution de CSP par décomposition

Le *Tree-Clustering* (noté *TC* [7]) constitue la méthode de référence pour la résolution de CSP binaires

1. Au moment de la soumission, nous ignorions l'existence de cette nouvelle notion essentiellement étudiée d'un point de vue combinatoire car elle n'apparaît actuellement dans aucune publication mais uniquement dans un rapport technique.

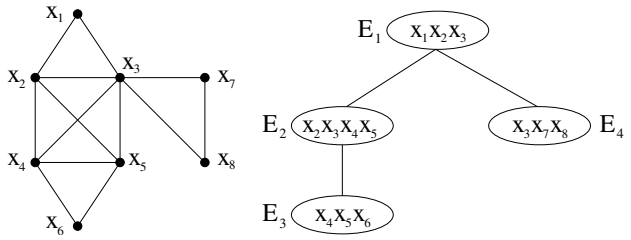


FIGURE 1 – Un graphe de contraintes de 8 variables (a) et une décomposition arborescente optimale (b).

via l'exploitation de la structure de leur graphe de contraintes. Cette méthode est basée sur la notion de décomposition arborescente de graphes [19].

Définition 1 *Étant donné un graphe $G = (X, C)$, une* décomposition arborescente *de G est une paire (E, T) où $T = (I, F)$ est un arbre et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé cluster ou bag au niveau mathématique) E_i est un nœud de T et vérifie :*

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

(ii) pour chaque 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 est un chemin de i vers j dans T, alors $E_i \cap E_j \subseteq E_k$.

Notons que la condition (iii) peut être remplacée par : si un sommet x est tel que $x \in E_i \cap E_j$, alors, tous les nœuds E_k de T qui apparaissent dans l'unique chemin de E_i à E_j contiennent x. La largeur *d'une décomposition arborescente (E, T) est égale à $max_{i \in I}|E_i| - 1$. La* largeur arborescente *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

La figure 2(b) présente un arbre dont les nœuds correspondent aux cliques maximales du graphe présenté dans la figure 2(a). Il s'agit de l'une des décompositions arborescentes de ce graphe. Ainsi, 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 $E_4 = \{x_3, x_7, x_8\}$. La taille maximum des clusters dans cette décomposition arborescente optimale vaut 4 et donc, la largeur arborescente de ce graphe vaut 3.

La première version de TC [7] débute par le calcul d'une décomposition arborescente (qui utilise l'algorithme MCS [22]). Dans la seconde étape, les clusters sont résolus indépendamment, en considérant chaque cluster comme un sous-problème, et donc, en énumérant toutes ses solutions. Après cela, une solution globale au CSP, s'il en existe une, peut alors être efficacement calculée en exploitant la structure arborescente de la décomposition. Les complexités en temps et en espace de cette première version sont en $O(n.w^+.log(d).d^{w^+ + 1})$ où $w^+ + 1$ est la taille du plus grand cluster ($w + 1 \leq w^+ + 1 \leq n$). Notons que cette première approche a été améliorée pour arriver à une complexité en espace en $O(n.s.d^s)$ [6, 5] où s est la taille de la plus grande intersection (*séparateur*) entre deux clusters ($s \leq w^+$). Malheureusement, ce type d'approche qui résout complètement chaque cluster n'est pas efficace dans la pratique. Afin de contourner ce type de problème, la méthode *BTD* (pour *Backtracking on Tree-Decomposition* [16]) a été proposée et a finalement démontré tout son intérêt sur le plan pratique, et figure désormais comme une méthode de référence dans l'état de l'art pour ce type d'approches. Contrairement à TC, BTD n'a pas besoin de résoudre exhaustivement chaque cluster pour démontrer l'existence de solution. Une recherche de type backtrack est réalisée en exploitant un ordre sur les variables, induit par une recherche en profondeur dans la décomposition arborescente considérée. Alors que cette approche a démontré son intérêt pratique, d'un point de vue théorique, dans le pire des cas, ses complexités en temps et en espace sont du même ordre que celles des versions améliorées de TC, précisément $O(n.s^2.e.log(d^s).d^{w^+ + 1})$ pour la complexité en temps, et $O(n.s.d^s)$ pour la complexité en espace. Aussi, pour rendre une méthode structurelle efficace, il faut *a priori* minimiser les valeurs de w^+ et s lors du calcul de la décomposition arborescente. Malheureusement, calculer une décomposition arborescente optimale (i.e. de largeur w) est NP-difficile [1]. Aussi, de très nombreux travaux ont abordé cette question. Ils exploitent souvent une approche algorithmique fondée sur la notion de graphe *triangulé* (se reporter à [11] pour une introduction à cette classe de graphes). Nous pouvons distinguer différentes classes d'approches. Tout d'abord, il y a les méthodes recherchant des décompositions optimales ou bien leur approximation (avec garanties) mais qui n'ont pas démontré à ce jour leur intérêt pratique pour une raison de temps d'exécution extrêmement important au regard de la faible amélioration de la valeur de w^+ qu'elles permettent d'obtenir. Viennent ensuite les méthodes n'offrant aucune garantie en termes d'optimalité (comme celles fondées sur les *triangulations heuristiques*) mais qui s'avèrent les plus utilisées au niveau pratique. Elles opèrent en temps polynomial (entre $O(n + e)$ et $O(n^3)$), sont très simples à implémenter, et leur avantage semble tout à fait justifié. En effet, ces heuristiques semblent produire des triangulations assez proches de l'optimum [17]. En pratique, les méthodes les plus usitées pour calculer des décompositions arborescentes sont basées sur MCS [22] et Min-Fill [20] qui offrent de bonnes approximations de w^+. De plus, dans [14], les expérimentations ont montré que l'efficacité pour la résolution de CSP n'est pas seulement corrélée à la valeur de w^+, mais aussi à celle de s. Néanmoins, à notre connaissance,

ces études se sont essentiellement concentrées sur la valeur de w^+ et à un degré moindre de s, mais pas sur la structure interne des clusters qui semble également fournir un paramètre des plus pertinents. Cette question est développée dans la partie 3 qui montre que les propriétés topologiques des clusters constituent également un paramètre crucial pour la résolution de CSP.

3 Impact des clusters non connexes

L'étude des décompositions arborescentes montre qu'elles recèlent très souvent des clusters constitués de plusieurs composantes connexes. Par exemple, considérons un réseau constitué d'un cycle sans corde (c'est-à-dire sans arête joignant deux sommets non consécutifs dans le cycle) de n sommets (avec $n \geq 4$). Toute décomposition arborescente optimale possède exactement $n-2$ clusters de taille 3, et parmi eux, $n-4$ clusters possèdent 2 composantes connexes.

Ce phénomène est également observé sur des instances réelles, dès lors que l'on considère des décompositions arborescentes de qualité satisfaisante. Par exemple, la célèbre instance RLFAP *Scen-06* que l'on retrouve notamment dans la compétition CSP 2008 [2] est définie sur 200 variables et son réseau admet des décompositions arborescentes de petite largeur et qui peuvent être calculées très facilement (e.g. Min-Fill en trouve une pour laquelle $w^+ = 20$). Malheureusement, une analyse détaillée de ces décompositions arborescentes montre qu'elles possèdent plusieurs clusters non connexes. Plus généralement, il s'avère qu'environ 32% des 7272 instances de la compétition CSP 2008 possèdent des décompositions arborescentes qui recèlent au moins un cluster non connexe quand MCS ou Min-Fill sont utilisées, ce qui est généralement le cas quand on emploie ce type de méthodes pour la résolution de CSP. Parmi ces instances pour lesquelles MCS ou Min-Fill produisent de telles décompositions arborescentes, nous retrouvons notamment la plupart des instances RLFAP ou FAPP qui se trouvent être par ailleurs souvent exploitées comme benchmarks pour ce type de méthodes, à la fois pour les problèmes de décision et pour ceux d'optimisation. De plus, parfois, le pourcentage de clusters non connexes dans ces instances peut être considérable, atteignant jusqu'à 99% et en moyenne aux environs de 35%. Pour les instances FAPP, la moyenne est d'environ 48% pour les décompositions arborescentes produites par Min-Fill, et plus importante encore en utilisant MCS. Cette observation serait encore plus frappante pour les algorithmes qui trouvent des décompositions de largeurs plus petites, comme illustré par l'exemple du cycle sans corde.

La présence de clusters non connexes dans les décompositions arborescentes considérées peut avoir un

2. Voir http ://www.cril.univ-artois.fr/CPAI08.

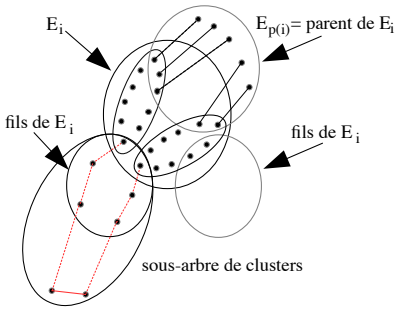


FIGURE 2 – Cluster non connexe au sein d'une décomposition arborescente.

impact extrêmement négatif sur l'efficacité pratique des méthodes de décomposition qui seront alors sanctionnées, lors de la résolution des instances, par un temps de calcul très important et un recours considérable à la mémoire. Pour bien comprendre cela, il faut déjà se rappeler que si un réseau de contraintes n'est pas connexe, cela peut avoir des conséquences importantes sur l'efficacité de sa résolution. Par exemple, si l'une de ses composantes connexes n'a pas de solution, et si la résolution aborde d'abord une composante connexe qui en possède, chacune de ses solutions devra être obtenue avant de prouver l'incohérence du CSP. Pour le cas des méthodes de décomposition, l'existence de clusters non connexes est peut-être encore plus pernicieuse. Dans le cas de TC, considérons un cluster non connexe. D'une part, le phénomène déjà rencontré dans le cas de réseaux non connexes peut se présenter. Mais il est également possible que ce cluster possède des solutions. Toutes ces solutions seront alors calculées et mémorisées avant de traiter un autre cluster. Leur nombre peut être considérable car il correspond au produit du nombre de solutions de chacune de ses composantes connexes. Notons de plus que pour certains benchmarks provenant des instances FAPP, le nombre de composantes connexes dans un cluster peut même être supérieur à 100. Toutefois, de nombreuses solutions locales à ce cluster peuvent être incompatibles globalement, car ces composantes connexes seront en général reliées par des contraintes qui apparaissent dans d'autres clusters. La figure 2 présente un exemple de décomposition pour laquelle deux composantes connexes d'un cluster E_i sont connectées via une séquence de contraintes qui apparaissent dans un sous-problème enraciné dans ce cluster. Ainsi, l'incohérence globale des solutions locales de E_i ne peut être détectée que quand ces clusters auront été résolus, au moment de la recomposition de solutions globales produites par TC lors de sa dernière étape. Cela conduit TC à une consommation considérable de temps et de mémoire, rendant de fait cette approche irréaliste en pratique.

D'autres méthodes ont été proposées de sorte à éviter ce type de phénomène où les clusters sont résolus indépendamment lors d'une première étape. C'est no-

tamment le cas de BTD qui est l'une des approches les plus efficaces basées sur des décompositions. Bien que BTD ait montré son intérêt pratique, le phénomène observé existe toujours, même s'il s'avère généralement atténué. Pour bien comprendre cela, nous devons rappeler que BTD résout une instance en résolvant successivement les sous-problèmes enracinés dans chaque cluster de la décomposition arborescente. Mais contrairement à TC qui calcule d'abord toutes les solutions d'un cluster, lors de l'accès à un cluster, BTD se limite à ne calculer qu'une seule solution. Grossièrement, le sous-problème enraciné dans un cluster E_i correspond à un sous-problème impliquant toutes les variables figurant dans la descendance de E_i dans la décomposition arborescente (voir [16] pour plus de détails). BTD commence une recherche de type backtrack en affectant systématiquement les variables du cluster racine avant d'explorer un cluster fils. Lors de l'exploration d'un nouveau cluster E_i, BTD affecte les variables qui apparaissent dans le cluster E_i exceptées celles figurant dans le séparateur $E_i \cap E_{p(i)}$ [3]. Par exemple, considérons le graphe de contraintes de la figure 2 et sa décomposition arborescente associée. Si nous supposons que E_1 est le cluster racine, BTD essaye d'abord d'affecter de façon cohérente les variables de E_1. Si c'est le cas, BTD poursuit la recherche avec l'un des clusters fils (i.e. E_2 ou E_4). Si BTD choisit d'explorer d'abord E_2, il devra affecter de façon cohérente les variables de $E_2 \backslash (E_1 \cap E_2)$ (i.e x_4 et x_5).

Maintenant et plus généralement, considérons le cas d'un cluster non connexe E_i. Nous avons deux cas :

- si $G[E_i \backslash (E_i \cap E_{p(i)})]$ [4] est déconnecté : BTD doit affecter de façon cohérente les variables qui sont réparties dans plusieurs composantes connexes. Si le sous-problème enraciné en E_i est trivialement cohérent (par exemple, il admet un grand nombre de solutions), BTD va trouver une solution en faisant au plus quelques backtracks et enchaîner directement sur la recherche dans le cluster suivant. Ainsi, dans un tel cas, la non-connexité de E_i n'entraîne pas de problème. En revanche, si ce sous-problème a peu de solutions, voire aucune, nous avons une probabilité significative que BTD passe beaucoup de temps d'une composante connexe de $G[E_i \backslash (E_i \cap E_{p(i)})]$ à une autre lorsqu'il résoudra ce cluster. BTD peut avoir à explorer toutes les affectations cohérentes de chaque composante connexe en intercalant éventuellement les variables des différentes composantes connexes. En effet, si BTD exploite des techniques de filtrage, l'affectation d'une valeur à une variable x de $E_i \backslash (E_i \cap E_{p(i)})$ a une incidence principalement sur les variables de la composante connexe de $G[E_i \backslash (E_i \cap E_{p(i)})]$ qui contient x. En revanche, le filtrage ne modifiera pas ou très peu le domaine d'une variable figurant dans une autre composante connexe. Cela implique que les incohérences seront souvent détectées plus tard et pas nécessairement dans E_i mais dans l'un de ses clusters descendants (comme illustré dans la figure 2). Si c'est le cas, BTD peut nécessiter une quantité considérable de temps et de mémoire (à cause de l'enregistrement de (no-)goods) pour résoudre le sous-problème enraciné en E_i, surtout si les variables ont des domaines de grande taille. Ce phénomène négatif a par exemple été observé empiriquement sur certains benchmarks de la classe FAPP (e.g. l'instance *normalized-fapp05-0350-10*) avec une version de BTD basée sur MAC [21].
- si $G[E_i \backslash (E_i \cap E_{p(i)})]$ est connexe : nécessairement, E_i est un cluster non connexe parce que son séparateur avec son cluster père n'est pas connexe. Comme les variables de ce séparateur sont déjà affectées, la non-connectivité de E_i n'entraîne aucun problème.

Cet impact négatif des clusters non connexes est tout à fait compatible avec les résultats empiriques rapportés dans la littérature. Nous avons constaté que parfois, le pourcentage de clusters non connexes avec Min-Fill diffère sensiblement d'avec celui de MCS, ce qui pourrait notamment expliquer certaines différences d'efficacité observées par différents auteurs. En effet, même si la largeur est identique, les décompositions calculées par Min-Fill offrent de meilleurs résultats pour la résolution que ceux obtenus avec MCS [14] ; et Min-Fill est considérée comme étant la meilleure heuristique de l'état de l'art. Par ailleurs, l'analyse des décompositions arborescentes montre également que la connexion entre composantes connexes de certains clusters est fréquemment observée seulement au niveau des clusters feuilles de la décomposition, ce qui augmente davantage les effets négatifs observés. Pour éviter ce genre de phénomène, nous étudions dans la partie 4 les classes de décompositions arborescentes pour lesquelles tous les clusters devront être connexes.

4 Un nouveau paramètre pour la décomposition de réseaux de contraintes

Les constats présentés plus haut nous conduisent tout naturellement à ne considérer que des décompositions arborescentes pour lesquelles, les clusters seraient tous connexes. Cette notion a très récemment été introduite dans le cadre de la Théorie des Graphes mais n'a semble-t-il pas trouvé, pour le moment,

3. Nous noterons $E_{p(i)}$ le cluster parent du cluster E_i et nous supposerons que $E_i \cap E_{p(i)} = \emptyset$ si E_i est le cluster racine.

4. Pour tout $Y \subseteq X$, $G[Y] = (Y, C_Y)$ est le sous-graphe de $G = (X, C)$ induit par Y où $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

d'écho au sein de cette communauté. Plus précisément, la notion de *Connected Tree-Width* est présentée dans [18] [5] et a fait l'objet, lors de sa définition, d'une étude portant sur ses propriétés combinatoires. Les questions algorithmiques, comme notamment le problème de son calcul en termes de complexité ou bien la proposition d'algorithme l'approximant, n'ont pas été évoquées. Cette contribution fournit en fait un théorème central indiquant une majoration de ce nouveau paramètre en fonction de la largeur arborescente et de la longueur maximum de ses cycles géodésiques (cycles pour lesquels la distance entre toute paire de sommets du cycle est égale à la longueur du plus court chemin empruntant le cycle). Du point de vue terminologique, il nous apparaît préférable d'utiliser les termes Bag-Connected Tree-Decomposition et Bag-Connected Tree-Width qui n'ont à ce jour, et à notre connaissance, pas encore fait l'objet d'une autre définition. Nous présentons donc la notion de Bag-Connected Tree-Decomposition, qui correspond aux décompositions arborescentes telles que chaque cluster E_i est connexe (i.e. $G[E_i]$ est un graphe connexe).

Définition 2 *Étant donné un graphe $G = (X, C)$, une* Bag-Connected Tree-Decomposition *de G est une décomposition arborescente (E, T) de G telle que pour tout $E_i \in E$, le sous-graphe $G[E_i]$ de G induit par E_i est un graphe connexe. La largeur d'une Bag-Connected Tree-Decomposition (E, T) est égale à $max_{i \in I}|E_i| - 1$ et la* Bag-Connected Tree-Width w_c *est la largeur minimale pour toutes les bag-connected tree-decompositions de G.*

Étant donné un graphe $G = (X, C)$ de largeur arborescente w, on a nécessairement $w \leq w_c$. Néanmoins, si G est un graphe triangulé, $w = w_c$. Sinon, par exemple pour les graphes formés d'un cycle de longueur n et sans corde, la Bag-Connected Tree-Width vaut $\lceil \frac{n}{2} \rceil$.

Une question naturelle maintenant est liée au calcul des Bag-Connected Tree-Decompositions optimales, c'est-à-dire de largeur égale à w_c. Nous montrons que ce problème, comme pour le cas des décompositions arborescentes, est NP-difficile (la preuve est donnée en annexe).

Théorème 1 *Calculer une Bag-Connected Tree-Decomposition optimale est NP-difficile.*

Nous avons vu que pour la résolution de CSP, il n'est pas nécessaire de trouver une décomposition arborescente optimale, et c'est même souvent souhaitable. Aussi, nous proposons ici un algorithme appelé *Bag-Connected-TD*, qui calcule, pour un graphe $G = (X, C)$, une Bag-Connected Tree-Decomposition en temps polynomial, bien sûr, sans aucune garantie quant à son optimalité

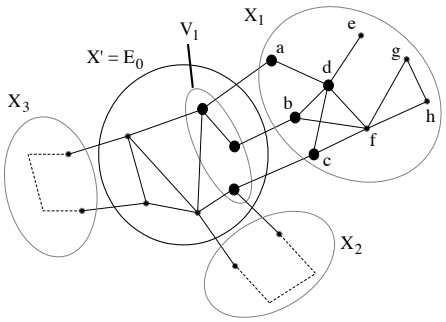


FIGURE 3 – Illustration de *Bag-Connected-TD*

La première étape de l'algorithme calcule un premier cluster, noté E_0, constitué d'un sous-ensemble de sommets connexes. X' notera par la suite l'ensemble des sommets déjà traités. Cet ensemble est initialisé à E_0. Cette première étape peut se faire facilement, en utilisant une méthode heuristique. Dans la suite, nous noterons $X_1, X_2, \ldots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0. Chacun de ces ensembles X_i est inséré dans une file F. Pour chaque élément X_i supprimé de la file F, on notera $V_i \subseteq X$ l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i. On peut noter que V_i (qui peut être connexe ou non) est un séparateur du graphe G puisque la suppression de V_i dans G rend G non connexe (X_i étant déconnecté du reste de G). Un nouveau cluster E_i est alors initialisé par cet ensemble V_i. Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. Nous choisissons un premier sommet $x \in X_i$ qui est connecté à au moins un sommet de E_i (donc un sommet de V_i). Ce sommet est ajouté à E_i. Si $G[E_i]$ est connexe, le processus est arrêté puisque nous sommes sûrs que E_i sera un nouveau cluster connexe. Sinon, on continue, en prenant un autre sommet de X_i.

La figure 3 présente le calcul de E_1, le second cluster (après E_0), lors du premier passage dans la boucle. Après l'ajout des sommets a, b et c, le sous-graphe $G[V_1 \cup \{a, b, c\}]$ n'est pas connexe. Si le prochain sommet atteint est d, il est rajouté à E_1, et donc $E_1 = V_1 \cup \{a, b, c, d\}$ constitue un nouveau cluster connexe, stoppant ainsi la recherche dans $G[V_1 \cup X_1]$.

Quand ce calcul est terminé, nous rajoutons les sommets de E_i à X' et nous calculons $X_{i_1}, \ldots X_{i_{k_i}}$ les composantes connexes du sous-graphe $G[X_i \backslash E_i]$. Chacune est alors rajoutée à la file F. Dans l'exemple de la figure 3, deux composantes connexes seront calculées, $\{e\}$ et $\{f, g, h\}$. Le processus se poursuit tant que la file n'est pas vide. Dans l'exemple, et pour la partie droite du graphe, l'algorithme calculera 3 clusters connexes :

5. À ne pas confondre avec la notion du même nom introduite dans [9] mais qui est différente.

Algorithme 1: *Bag-Connected-TD*

Input : Un graphe $G = (X, C)$
Output : Un ensemble de clusters $E_0, \dots E_m$ d'une bag-connected tree-decomposition de G

1 Choisir un premier cluster connexe E_0 in G
2 $X' \leftarrow E_0$
3 Soient $X_1, \dots X_k$ les composantes connexes de $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \dots X_k\}$
5 **while** $F \neq \emptyset$ **do** `/* calcule un nouveau cluster` E_i `*/`
6 Enlever X_i de F
7 Soit $V_i \subseteq X'$ le voisinage de X_i dans G
8 $E_i \leftarrow V_i$
9 Recherche dans $G[V_i \cup X_i]$ à partir de V_i plus $x \in X_i$. Chaque fois qu'un nouveau sommet x est atteint, il est ajouté à E_i. Le processus s'arrête dès que le sous-graphe $G[E_i]$ est connexe
10 **if** V_i *appartient aux clusters déjà trouvés* **then**
11 Détruire le cluster V_i (parce que $V_i \subsetneq E_i$)
12 $X' \leftarrow X' \cup E_i$
13 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
14 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

$\{d, e\}$, $\{b, c, d, f\}$ et $\{f, g, h\}$.

Notons que la ligne 11 est utile uniquement quand l'algorithme construit un cluster E_i contenant tous les sommets d'un cluster déjà obtenu E_j (i.e. $E_j \subsetneq E_i$). Dans un tel cas, le cluster E_j peut être supprimé car il sera inutile dans la décomposition arborescente.

Nous établissons maintenant la validité de l'algorithme, puis sa complexité temporelle (les preuves sont données en annexe).

Théorème 2 *Bag-Connected-TD calcule les clusters d'une Bag-Connected Tree-Decomposition.*

Théorème 3 *La complexité en temps de l'algorithme Bag-Connected-TD est* $O(n(n+e))$.

D'un point de vue pratique, on peut supposer que le choix du premier cluster E_0 peut être crucial pour la qualité de la décomposition qui est en cours de calcul. De même, le choix de sommet x, sélectionné dans la ligne 9 peut être d'une importance considérable. Pour ces deux choix, des heuristiques peuvent bien sûr être utilisées. Cette question est abordée dans la partie suivante. Cependant, un choix particulier de ces heuristiques permet, sans aucune modification de la complexité, de calculer des décompositions arborescentes optimales pour le cas des graphes triangulés. Supposons que le premier cluster E_0 soit une clique maximale. Cela peut être réalisé efficacement en utilisant une approche gloutonne. Maintenant, pour le choix du sommet x à la ligne 9, nous considérons le sommet qui possède le nombre maximum de voisins dans l'ensemble V_i. Comme dans un graphe triangulé, tous les clusters d'une décomposition arborescente optimale sont des cliques, nécessairement, V_i étant une clique, x sera relié à tous les sommets de V_i et donc, E_i sera une clique. Progressivement, chaque clique maximale sera trouvée et la décomposition arborescente sera optimale. Les lignes 10-11 seront utilisées pour le cas de cliques maximales incluant plus d'un sommet x issu d'une nouvelle composante connexe. Dans tous les cas, l'intérêt pratique de ce type de décomposition est basé à la fois sur l'efficacité de son calcul, mais aussi sur l'apport qu'elle peut avoir pour la résolution de CSP. Cette question est abordée dans la partie suivante.

5 Résultats expérimentaux

Dans cette section, nous comparons l'efficacité de la résolution ainsi que les valeurs des paramètres structurels, selon que l'on utilise des décompositions arborescentes produites par Min-Fill ou bien par l'algorithme *Bag-Connected-TD*. En ce qui concerne le choix du premier cluster dans les calculs de Bag-Connected Tree-Decompositions, nous calculerons de façon gloutonne une clique maximale du réseau de contraintes [6]. Dans l'algorithme *Bag-Connected-TD*, pour choisir le prochain sommet, nous avons considéré six heuristiques dont nous ne présentons ici que les 4 meilleures :

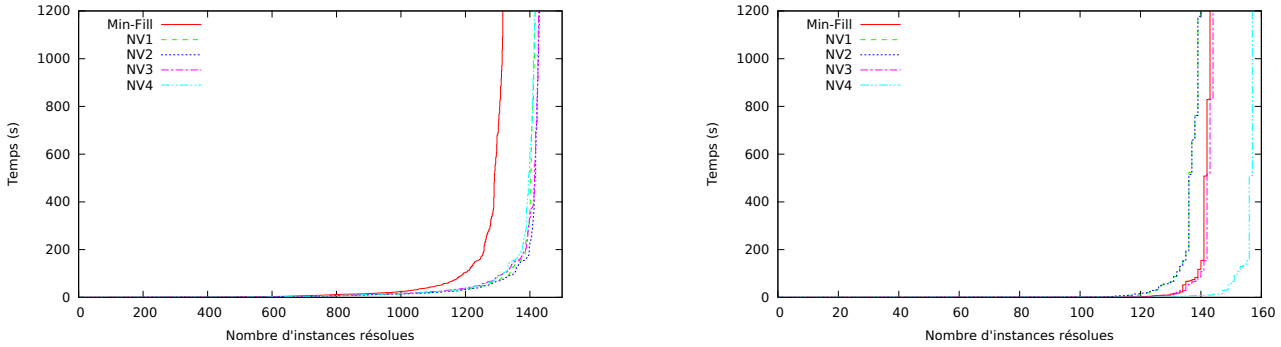
- NV1 : le prochain sommet est un sommet situé dans le voisinage des sommets précédemment choisis.
- NV2 : les sommets sont traités dans l'ordre décroissant des degrés.
- NV3 : les sommets sont traités selon l'ordre dans lequel ils sont visités par un parcours *en largeur d'abord* du graphe à partir des sommets de V_i.
- NV4 : nous choisissons comme prochain sommet celui qui possède le nombre maximum de voisins dans l'ensemble V_i.

La résolution de CSP est réalisée par BTD basée sur MAC, en utilisant l'heuristique de choix de variables *dom/wdeg* [3]. Nous choisissons comme cluster racine celui qui maximise le rapport $\frac{e}{n-1}$. Ce choix donne de meilleurs résultats que ceux introduits dans [15]. Les temps d'exécution de la décomposition pour Min-Fill et *Bag-Connected-TD* sont similaires et sont inclus dans ceux de BTD et ils se révèlent relativement négligeables au regard du temps de résolution. Toutes les implémentations sont écrites en C++. Elles ont été réalisées sur un PC sous Linux doté d'un processeur Intel Pentium IV 3, 2 GHz avec 1 Go de mémoire.

5.1 Instances pour lesquelles Min-Fill produit des clusters non connexes

Dans ce paragraphe, nous comparons, du point de vue de l'efficacité de la résolution, les bag-connected tree-decompositions à celles qui contiennent des clusters non connexes. Pour ce faire, nous considérons

6. Rappelons-nous que nous utilisons les 2-sections pour les instances non binaires.



(a) (b)

FIGURE 4 – Nombre d'instances résolues pour chaque décomposition arborescente considérée pour les benchmarks pour lesquels Min-Fill produit des clusters non connexes (a) et pour les instances pour lesquelles Min-Fill produit une décomposition arborescente avec des clusters connexes (b).

1 597 instances (d'arité quelconque) parmi les 2 310 instances de la compétition CSP 2008 et pour lesquelles Min-Fill produit une décomposition arborescente possédant au moins un cluster non connexe. Sont exclues des résultats, les instances qui n'ont pu être résolues sans dépasser la limite de temps (à savoir 1 200 secondes) ou qui contiennent des contraintes globales (car elles ne sont pas encore mises en œuvre dans notre solveur). Parmi les instances étudiées, nous pouvons notamment trouver des instances issues des familles *rlfap*, *fapp*, *modifiedRenault*, *graphColoring*, *bqwh* ou *travellingSalesman*.

La figure 4 (a) présente le nombre cumulé d'instances résolues pour chaque type de décomposition arborescente. Tout d'abord, nous pouvons observer qu'en utilisant chacune des bag-connected tree-decompositions, BTD résout plus d'instances qu'en utilisant les décompositions arborescentes déconnectées produites par Min-Fill. On notera que cette observation reste vraie si nous utilisons les décompositions produites par les heuristiques non présentées. Le meilleur nombre d'instances résolues est obtenu grâce aux décompositions produites par les heuristiques NV2 et NV3. Ces décompositions permettent de résoudre respectivement 114 et 111 instances de plus qu'avec Min-Fill. Celles à base de NV1 et NV4 sont proches l'une de l'autre. Par ailleurs, pour toute les décompositions, la plupart des instances sont résolues en moins de 60 secondes.

Afin de comparer plus équitablement les temps d'exécution, nous ne considérons maintenant que les instances qui sont résolues par BTD pour toutes les décompositions considérées, y compris Min-Fill. Le temps d'exécution pour résoudre les 1 230 instances en utilisant les décompositions basées sur Min-Fill est de 50 669 secondes alors qu'en utilisant les décompositions connexes basées sur des NV1, cela ne nécessite que 32 372 secondes. Les décompositions basées sur NV2 sont relativement proches de NV1, à savoir 33 202 secondes. Celles qui sont basées sur NV3 et NV4 sont légèrement plus lentes avec respectivement 36 420 et 36 087 secondes. On notera que les deux autres heuristiques (non présentées ici) surpassent également la décomposition Min-Fill.

Si nous nous concentrons sur les 329 instances ayant une structure bien adaptée à la décomposition (i.e. de largeur relativement faible par rapport au nombre de variables), de nouveau, on observe la même tendance, à savoir que BTD exploité sur des bag-connected tree-decompositions est plus performant que BTD avec Min-Fill. Le meilleur temps d'exécution est obtenu par BTD en utilisant NV1 avec 5 698 secondes, tandis que le pire l'est en utilisant Min-Fill avec 13 641 secondes. De plus, BTD utilisant NV2 et NV4 fournit des résultats voisins l'un de l'autre avec respectivement 6 137 et 6 010 secondes tandis que BTD avec NV3 nécessite 8 483 secondes.

Enfin, si l'on compare ces résultats avec ceux obtenus par un algorithme énumératif classique comme MAC, nous pouvons noter que certaines instances résolues par BTD avec certains NVi ne sont pas résolues par MAC et inversement. Nous observons également que MAC se comporte parfois mieux, parfois moins bien que BTD basé sur des décompositions connexes. Toutefois on peut estimer que globalement, les résultats sont similaires. Ceci s'explique par le fait que la plupart des 1 597 instances que nous considérons sont loin d'avoir une structure adaptée à la résolution par décomposition. En revanche, lorsque la structure possède des caractéristiques intéressantes, BTD surclasse clairement MAC. Par exemple, BTD exploitant une décomposition basée sur NV3 ne nécessite que 856 secondes pour résoudre 10 instances sur les 12 de la famille *rlfapScens11* tandis que MAC n'en résout que 8 en 1 595 secondes. Par ailleurs, pour résoudre ces huit instances, BTD ne nécessite que 63 secondes, ce qui correspond à un comportement 25 fois plus rapide que celui de MAC.

5.2 Instances pour lesquelles Min-Fill produit des décompositions arborescentes connexes

Nous abordons ici brièvement le comportement de BTD pour résoudre les instances pour lesquelles Min-Fill produit des bag-connected tree-decompositions. Bien sûr, pour ces instances, Min-Fill et l'algorithme *Bag-Connected-TD* ne produisent pas nécessairement les mêmes décompositions. Nous nous concentrons ici sur les cas les plus pertinents, c'est-à-dire les 191 instances ayant une structure adaptée à une approche de résolution par décomposition.

Comme le montre la figure 4 (b), l'utilisation de BTD basé sur Min-Fill permet de résoudre plus d'instances que BTD basé sur NV1 ou NV2 (143 contre 140 instances), mais un nombre inférieur à BTD basé sur NV3 ou NV4 qui en résolvent respectivement 144 et 157. Si nous concentrons notre étude sur les 132 instances qui sont résolues par BTD pour toutes les décompositions arborescentes considérées, y compris avec Min-Fill, BTD basé sur NV3, NV4 ou Min-Fill conduit aux meilleurs cumuls de temps d'exécution avec respectivement 1 283, 1 298 et 1280 secondes tandis que BTD basé sur NV1 ou NV2 est bien plus lent, avec respectivement 2 226 et 2 265 secondes.

5.3 Comparaison des paramètres structurels

Le tableau 1 présente la valeur des paramètres structurels pour certaines instances qui sont représentatives des tendances générales. Sans surprise, Min-Fill produit des décompositions de largeur plus petite, mais avec un plus grand nombre de clusters, que celles produites par *Bag-Connected-TD*. Cependant, si dans certains cas, la largeur obtenue par *Bag-Connected-TD* est significativement plus grande que celle fournie par Min-Fill (e.g. la largeur produite par NV3 pour l'instance *squares-23-23*), dans d'autres cas, elle reste relativement proche, voire parfois même égale à celle obtenue par Min-Fill. Cela se produit notamment pour l'instance *renault-mod-33_ext* mais aussi pour les instances pour lesquelles Min-Fill produit une bag-connected tree-decomposition (cf. partie (b) du tableau 1). Nous observons également que la qualité de la largeur obtenue grâce à *Bag-Connected-TD* peut varier considérablement selon les instances. Si NV1 présente souvent la meilleure largeur parmi celles calculées par l'algorithme *Bag-Connected-TD*, celle-ci est cependant parfois inférieure pour NV3 ou NV4 (e.g. pour l'instance *mps-red-qnet1*).

Enfin, pour ce qui concerne le paramètre s, les tendances constatées se révèlent similaires à celle observées pour la largeur.

6 Conclusion

Dans cet article, nous avons introduit, dans le cadre de la résolution de CSP, le concept de Bag-Connected Tree-Decomposition. Après avoir montré l'intérêt de cette notion et proposé un premier algorithme polynomial qui calcule de telles décompositions, nous avons démontré expérimentalement la pertinence de cette approche car elle permet d'améliorer considérablement la résolution de CSP basée sur les méthodes de décomposition. En effet, ce type de décomposition permet de résoudre beaucoup plus d'instances et d'améliorer le temps d'exécution, par exemple, d'environ 63% dans le cas des instances pour lesquelles Min-Fill produit des clusters non connexes. En outre, ces décompositions peuvent aussi, pour les benchmarks bien structurés, rendre BTD bien plus performant que MAC, en étant en particulier jusqu'à 25 fois plus rapide (par exemple pour la famille d'instances *rlfapScens11*).

La première extension à ce travail porte sur l'étude de ces décompositions dans le domaine plus général des modèles graphiques en IA. Il s'agit d'exploiter cette notion avec d'autres classes de méthodes telles que l'Hypertree-Decompositions, And/Or Search, ou encore Bucket Elimination. Cette démarche est en effet particulièrement justifiée par le fait que, même si certaines de ces approches sont en théorie basées sur d'autres paramètres (par exemple l'Hypertree-Width), les implémentations efficaces connues s'appuient généralement sur des algorithmes de calcul de décompositions arborescentes (par exemple Min-Fill pour l'Hypertree-Decomposition [8]). Un autre développement qui nous semble prometteur porte sur l'utilisation des Bag-Connected Tree-Decompositions dans le domaine de l'optimisation ou encore des problèmes de comptage de solutions.

Enfin, il serait aussi judicieux de développer le travail balbutiant relatif à une étude théorique de ce nouveau paramètre, d'un point de vue mathématique, et portant par exemple sur les propriétés fondamentales de la Bag-Connected Tree-Width. Le travail préliminaire présenté dans [18] offre un premier élément de réponse mais limité pour le moment à une seule borne supérieure de ce paramètre. Une autre question pourrait porter sur la mise en évidence de classes de graphes pour lesquelles ce paramètre est facile à calculer (i.e. de complexité polynomiale), ou bien alors proche de la largeur arborescente. Ou bien encore, de déterminer des problèmes qui sont difficiles quand la largeur arborescente est bornée par une constante, et qui deviendraient faciles quand la Bag-Connected Tree-Width est bornée par une constante.

TABLE 1 – Valeur des paramètres structurels pour certaines instances pour lesquelles Min-Fill produit des clusters non connexes (a), et pour lesquels Min-Fill produit une bag-connected tree-decomposition (b).

	Instances	n	e	Min-Fill		NV1		NV2		NV3		NV4	
				w^+	s	w_c^+	s	w_c^+	s	w_c^+	s	w_c^+	s
(a)	2-insertions-4-3	149	541	38	34	66	54	95	14	101	66	58	57
	ewddr2-10-by-5-9	50	265	16	15	22	17	21	20	26	23	45	37
	renault-mod-33_ext	111	133	11	11	12	11	14	11	17	15	16	13
	scen7	400	2 865	33	29	90	48	319	9	116	94	81	34
	squares-23-23	1 058	1 268	45	4	45	5	45	5	235	88	45	26
	fapp06-0500-1	500	3 478	221	210	286	284	286	284	314	314	313	248
	js-taillard-15-100-4	225	1 785	86	70	114	102	121	97	129	102	210	197
(b)	mps-red-qnet1	5 380	621	970	773	1 272	1 265	1 272	1 265	978	954	998	974
	anna-9	138	493	12	12	14	14	14	14	16	15	14	13
	haystacks-10	100	459	9	1	9	1	9	1	9	1	9	1
	renault-mod-8_ext	111	126	11	11	11	11	12	11	13	12	11	11
	qwh-15-106-9_ext	225	2 324	99	99	102	102	102	102	103	103	173	168

Références

[1] S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Disc. Math.*, 8 :277–284, 1987.

[2] C. Berge. *Graphs and Hypergraphs*. Elsevier, 1973.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, 2004.

[4] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4 :79–89, 1999.

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

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

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

[8] A. Dermaku, T. Ganzow, G. Gottlob, B. J. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decomposition. In *MICAI*, pages 1–11, 2008.

[9] P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In *LATIN*, pages 479–490, 2006.

[10] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1) :24–32, 1982.

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

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

[13] M. Gyssens, P. Jeavons, and D. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66 :57–89, 1994.

[14] 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.

[15] 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*, pages 741–745, 2006.

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

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

[18] Malte Muller. Connected tree-width. *CoRR*, abs/1211.7353, 2014.

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

[20] D. J. Rose. A graph theoretic study of the numerical solution of sparse positive denite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

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

[22] 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.

7 Annexes

Preuve du Théorème 1 : Nous proposons une réduction polynomiale du problème du calcul d'une décomposition arborescente optimale vers ce problème. Considérons un graphe $G = (X, C)$ de largeur arborescente w, la décomposition arborescente de G associée étant (E, T). Considérons le graphe G' obtenu en ajoutant à G un sommet universel x, i.e. un sommet qui est relié à tous les sommets de G. Notons qu'à partir de (E, T), nous pouvons obtenir une décomposition arborescente pour G' en ajoutant dans chaque cluster $E_i \in E$, le sommet x. Il s'agit d'une Bag-Connected Tree-Decomposition puisque chaque cluster est nécessairement connexe (au moins par des chemins contenant x) et sa largeur vaut $w + 1$. Pour montrer que cet ajout de sommet définit une réduction, il suffit de montrer que w est la largeur d'arbre de G si et seulement si la Bag-Connected Tree-Width w_c de G' est $w + 1$.

1. ($\Rightarrow$) Nous savons qu'au plus, la largeur de la décomposition arborescente considérée de G' est $w + 1$ puisque les clusters de cette décomposition arborescente sont connexes et que sa largeur est $w+1$. Supposons que $w_c \leq w$, et que donc, il existe une Bag-Connected Tree-Decomposition de G' de largeur au plus w. En utilisant la décomposition arborescente de G', on peut définir le même arbre, mais en supprimant le sommet x pour obtenir une décomposition arborescente de G de largeur $w-1$, ce qui contredit l'hypothèse.
2. ($\Leftarrow$) Avec le même type d'arguments que ci-dessus, nous savons que la largeur arborescente w de G est au plus $w_c - 1$. Et par construction, elle ne peut pas être strictement inférieure à $w_c - 1$. Donc, c'est exactement $w_c - 1$.

De plus, la construction de G' est possible en temps linéaire. □

Preuve du Théorème 2 : Il suffit de prouver les lignes 5-14 de l'algorithme. Nous montrons d'abord que l'algorithme s'arrête. À chaque passage dans la boucle, au moins un sommet sera rajouté à l'ensemble X' et ce sommet n'apparaîtra pas plus tard dans un nouvel élément de la file d'attente puisque ces éléments sont définis par les composantes connexes de $G[X_i \backslash E_i]$, un sous-graphe qui contient strictement moins de sommets qu'il n'y en a dans X_i. Ainsi, après un nombre fini d'étapes, l'ensemble $X_i \backslash E_i$ sera un ensemble vide, et donc aucun nouvel ajout à F ne sera possible.

Nous montrons maintenant que l'ensemble des clusters $E_0, E_1, \dots E_m$ induit une bag-connected tree-decomposition. Par construction, chaque nouveau cluster est connexe. Donc, nous devons juste prouver qu'ils induisent une décomposition arborescente. Nous le prouvons par induction sur les clusters ajoutés, en montrant que tous ces clusters ajoutés vont induire une décomposition arborescente du graphe $G[X']$.

Initialement, le premier cluster E_0 induit une décomposition arborescente du graphe $G[E_0] = G[X']$.

Pour l'induction, notre hypothèse est que l'ensemble des clusters déjà ajoutés $E_0, E_1, \dots E_{i-1}$ induit une décomposition arborescente du graphe $G[E_0 \cup E_1 \cup \dots \cup E_{i-1}]$. Considérons maintenant l'ajout de E_i. Nous montrons que par construction, $E_0, E_1, \dots E_{i-1}$ et E_i induit une décomposition arborescente du graphe $G[X']$ en montrant que les trois conditions (i), (ii) et (iii) de la définition des décompositions arborescentes sont satisfaites.

(i) Chaque nouveau sommet ajouté dans X' appartient à E_i

(ii) Chaque nouvelle arête dans $G[X']$ est à l'intérieur du cluster E_i.

(iii) On peut considérer deux cas différents pour un sommet $x \in E_i$, sachant que pour les autres sommets, la propriété est déjà satisfaite par l'hypothèse d'induction :

  (a) $x \in E_i \backslash V_i$: dans ce cas, x n'apparaît pas dans un autre cluster que E_i et donc, la propriété est vérifiée.

  (b) $x \in V_i$: dans ce cas, par hypothèse d'induction, la propriété a déjà été vérifiée.

Finalement, il est facile de voir que si la ligne 11 est appliquée, on obtient bien les clusters d'une décomposition arborescente du graphe $G[X']$. □

Preuve du Théorème 3 : Les lignes 1-4 sont réalisables en temps linéaire, soit $O(n + e)$, puisque le coût de calcul des composantes connexes de $G[X \backslash E_0]$ est borné par $O(n + e)$. Néanmoins, nous pouvons noter que la ligne 1 peut être réalisée par une heuristique éventuellement plus coûteuse, de sorte à obtenir un premier cluster plus pertinent, mais en se limitant à en $O(n(n+e))$ afin de ne pas accroître la complexité globale de l'algorithme. Nous analysons maintenant le coût de la boucle (ligne 5). Tout d'abord, notons qu'il y a nécessairement moins de n insertions dans la file d'attente F car à chaque passage dans la boucle, on est assuré qu'au moins, un nouveau sommet aura été rajouté dans X', et donc supprimé de l'ensemble des sommets n'ayant pas encore été traités. Nous analysons maintenant le coût de chaque traitement associé à l'ajout d'un nouveau cluster, dont nous donnons pour chacun, sa complexité globale.

- Ligne 6 : l'obtention du premier élément X_i de F est bornée par $O(n)$, et donc globalement par $O(n^2)$.

- Ligne 7 : l'obtention du voisinage $V_i \subseteq X'$ de X_i dans G est bornée par $O(n+e)$, et donc globalement par $O(n(n+e))$.
- Ligne 8 : cette étape est réalisable en $O(n)$, soit globalement en $O(n^2)$.
- Ligne 9 : le coût de la recherche dans $G[V_i \cup X_i]$ débutant avec les sommets de V_i et $x \in X_i$ est borné par $O(n+e)$. Puisque la boucle *while* s'exécute au plus n fois, le coût global de la recherche dans ces sous-graphes est borné par $O(n(n+e))$. En outre, pour chaque nouveau sommet x ajouté, la connexité de $G[E_i]$ est testée avec un coût supplémentaire borné par $O(n+e)$. On peut remarquer qu'un tel sommet n'est ajouté au plus qu'une fois, donc globalement, le coût de ce test est borné par $O(n(n+e))$. Donc, le coût de la ligne 9 est globalement borné par $O(n(n+e))$.
- Lignes 10-11 : en utilisant une structure de données adaptée, cette étape peut être réalisée en $O(n)$, soit globalement en $O(n^2)$.
- Ligne 12 : cette étape est réalisable en $O(n)$, soit globalement en $O(n^2)$.
- Ligne 13 : le coût de la recherche des composantes connexes de $G[X_i \backslash E_i]$ est borné par $O(n+e)$. Ainsi, globalement, le coût de cette étape est $O(n(n+e))$.
- Ligne 14 : l'insertion de X_{i_j} dans F est réalisable en $O(n)$, soit globalement en $O(n^2)$ puisqu'il y a moins de n insertions dans F.

Finalement, la complexité temporelle de l'algorithme *Bag-Connected-TD* est $O(n(n+e))$. □