De nouvelles approches pour la décomposition de réseaux de contraintes *

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**

Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296
13397 Marseille Cedex 20 (France)
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Selon la nature des instances de CSP à traiter, les méthodes exploitant la décomposition (arborescente) offrent une approche souvent efficace pour la résolution, le comptage de solutions et l'optimisation. Aussi, une bonne partie des efforts de la communauté s'est focalisée sur l'élaboration d'algorithmes visant à trouver les meilleures décompositions, i.e. celles qui minimisent la largeur arborescente, soit le paramètre central en termes de complexité théorique. Dans ce cadre, l'heuristique $Min\text{-}Fill$ constitue la méthode de référence. Nous introduisons ici deux nouvelles méthodes heuristiques dont le but est d'améliorer $Min\text{-}Fill$. L'une a pour objet de fonctionner sans *triangulation*, ce qui permet d'améliorer les temps de calculs. L'autre vise à améliorer le comportement de $Min\text{-}Fill$ en exploitant mieux la topologie des graphes. L'évaluation expérimentale que nous présentons montre l'intérêt de ces nouvelles approches.

Abstract

Depending on the nature of CSP instances to consider, the (tree-)decomposition methods offer an approach often efficient for the solving, the counting of solutions or the optimization. So, the community has focused a large part of its efforts on the production of algorithms aiming to find the best decompositions, i.e. ones which minimize the *tree-width*, a central parameter in terms of theoretical complexity. In this frame, the heuristic $Min\text{-}Fill$ constitutes the reference method. We introduce here two new heuristics with the aim in view to improve $Min\text{-}Fill$. The first one aims to proceed without *triangulation*, allowing notably an improvement of the runtime. The second intends to improve the behaviour of $Min\text{-}Fill$ by exploiting the topology of graphs. The experimental evaluation we present shows the interest of these new approaches.

*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 de Contraintes (CSP) offrent un cadre puissant pour la formulation de problèmes en Informatique, et en particulier en Intelligence Artificielle. Formellement, un *Problème de Satisfaction de 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, 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* 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. Pour simplifier les notations, dans la suite, nous noterons l'hypergraphe $(X, \{S(c_1), \ldots S(c_e)\})$ par (X, C). Sans manque de généralité, nous supposerons que les réseaux considérés sont connexes. 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 améliorer la résolution en pratique. Bien évidemment, la complexité de ces approches demeure exponentielle, au moins en $O(n.d^n)$ où n est le nombre de variables et d la taille maximum des domaines. Au-delà, étant donnée une instance CSP, d'autres questions peuvent être formulées conduisant à des problèmes plus difficiles

encore comme l'optimisation sous contraintes (NP-difficile) [4] ou le comptage de solutions (#P-complet) [20]. De plus, on peut utiliser certaines techniques développées dans le cadre CSP pour traiter des problèmes de compilation de connaissances qui conduisent également à des problèmes très difficiles en termes de complexité [10].

Afin de contourner la complexité intrinsèque de ces différents problèmes (décision, optimisation, comptage, compilation), d'autres approches sont fondées sur l'exploitation des classes polynomiales structurelles qui s'appuient sur la notion de *décomposition arborescente* de graphes [16]. Leur avantage fondamental est lié à leur complexité théorique, en général de l'ordre de d^{w+1} où w est la *largeur arborescente* du graphe de contraintes [1]. Ainsi, quand la largeur est limitée, ces méthodes permettent de traiter des instances de grande taille. C'est le cas par exemple des problèmes bien connus d'optimisation pour l'allocation de fréquences radio [5]. Notons qu'en pratique, la complexité en temps est plutôt de l'ordre de $d^{w^+ +1}$ où $w^+ \geq w$ est une approximation de la largeur arborescente car le calcul de décompositions optimales (i.e. de largeur w) constitue un problème NP-difficile [1].

Toutefois, la 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 comme nous l'avions montré lors des JFPC 2014 [14] et de CP 2014 [13]. Cela étant, lorsque l'on s'intéresse à des problèmes plus difficiles comme l'optimisation, le comptage ou encore la compilation, la largeur et donc sa minimisation devient alors cruciale. Malheureusement l'obtention d'une largeur optimale semble inaccessible en pratique, sauf pour de très petits graphes (quelques dizaines de sommets tout au plus) [2]. Aussi, en pratique, les décompositions sont généralement calculées par des approches heuristiques pour lesquelles *Min-Fill* [17] constitue à ce jour le meilleur compromis entre temps de calcul et qualité de la décomposition obtenue. Pour autant, cette méthode recèle certains inconvénients. D'une part, elle procède par triangulation, c'est-à-dire, par ajout d'arêtes. Cela peut rendre cette approche extrêmement coûteuse en temps, voire inopérante pour le cas de graphes de grande taille. D'autre part, *Min-Fill* n'exploite pas explicitement les propriétés topologiques du graphe, ce qui de fait pose un problème justement dans le cas du calcul de décompositions et conduit parfois cette heuristique à l'obtention de décompositions de qualité médiocre. La complexité temporelle de *Min-Fill* est $O(n(n + e'))$ où e' est le nombre d'arêtes du graphe après triangulation (on a donc toujours $e' \geq e$).

Dans cette contribution, afin de contourner ces problèmes, nous introduisons de nouvelles approches heuristiques. La première, appelée *Least-TD*, opère sans triangulation en réalisant un calcul basé sur un parcours du graphe qui se limite au calcul de clusters par le biais d'une exploitation de propriétés liées aux séparateurs (un séparateur est un ensemble de sommets dont la suppression engendre la présence de plusieurs composantes connexes dans le graphe) et aux composantes connexes. Elle conduit à un algorithme dont la complexité temporelle est inférieure à celle de *Min-Fill* avec $O(n(n+e))$, attestée d'ailleurs par une meilleure efficacité pratique. De plus, cette approche permet d'améliorer dans de nombreux cas la qualité de la décomposition.

La seconde approche peut être considérée comme mixte au sens où elle utilise les qualités de *Min-Fill* tout en exploitant la topologie du graphe mais en cherchant aussi à éviter les cas de figure où *Least-TD* pose problème. Si elle fournit également des décompositions qui sont en général de meilleure qualité que celles calculées par *Min-Fill*, l'exploitation conjointe de la triangulation et de la topologie du graphe conduit à une complexité plus élevée. En effet, l'algorithme *Min-Fill-MG* qui la met en œuvre a une complexité de l'ordre de $O(n^2(n + e'))$.

La partie suivante introduit les notions de bases associées à la décomposition arborescente et aux méthodes permettant d'en calculer. La partie 3 introduit l'algorithme *Least-TD* alors que la partie 4 présente l'algorithme *Min-Fill-MG*. Avant de conclure, nous présentons des résultats expérimentaux.

2 La décomposition et ses calculs

Historiquement, le *Tree-Clustering* (noté *TC* [8]) est la méthode de référence pour la résolution de CSP binaires via l'exploitation de la structure de leur graphe de contraintes. Elle est basée sur la notion de *décomposition arborescente de graphes* [16].

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 (appelés* clusters*) de X, telle que chaque cluster 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 sur un chemin de i vers j dans T, alors $E_i \cap E_j \subseteq E_k$.

1. Cette notion est exploitée pour les hypergraphes en l'appliquant sur leur *2-section* [3]. La 2-section d'un hypergraphe (X, C) est le graphe (X, C') tel que $C' = \{\{x, y\} | \exists c \in C, \{x, y\} \subseteq c\}$.

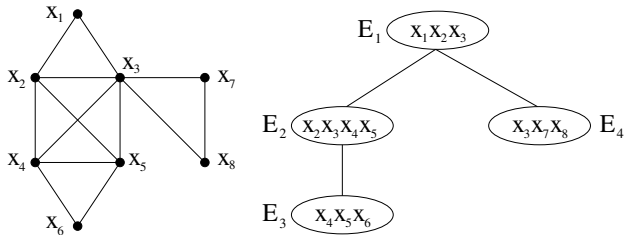

FIGURE 1 – Un graphe de contraintes de 8 variables (a) et une décomposition arborescente optimale (b).

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 *ou* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

La figure 1(b) présente un arbre dont les nœuds correspondent aux cliques maximales du graphe présenté dans la figure 1(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 [8] débute par le calcul d'une décomposition arborescente qui utilise l'algorithme MCS (Maximum Cardinality Search [19]). 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. Le complexité en temps de cette première version est en $O(n.w^+.log(d).d^{w^+ +1})$ et en $O(n.d^{w^+ +1})$ pour l'espace, 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 $O(n.s.d^s)$ [7, 6, 12] où s est la taille de la plus grande intersection (*séparateur*) entre deux clusters ($s \leq w^+$). Aussi, pour rendre ces méthodes structurelles efficaces, il faut *a priori* minimiser les valeurs de w^+ et s lors du calcul de la décomposition arborescente. Ceci est d'autant plus crucial quand on considère, au-delà des problèmes de décision, les problèmes d'optimisation, de comptage ou encore de compilation.

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* (voir [11] pour une introduction à cette classe de graphes), sachant que pour ces graphes, calculer une décomposition arborescente optimale est linéaire. Nous pouvons distinguer différentes classes d'approches. Tout d'abord, il y a les méthodes exactes ou par approximation (avec garantie) 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 heuristiques n'offrant aucune garantie en termes d'optimalité (comme celles fondées sur les *triangulations heuristiques*). Ce sont elles qui sont les plus utilisées en 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 obtenir des triangulations assez proches de l'optimum [15]. C'est le cas notamment de *Min-Fill* [17] qui calcule souvent des décompositions optimales en un temps beaucoup plus court que les approches exactes [18]. Aussi, en pratique, *Min-Fill* constitue l'approche de référence depuis maintenant plusieurs décennies. Nous la présentons dans le détail.

Algorithme 1 : *Min-Fill*

Entrées : Un graphe $G = (X, C)$
Sorties : Un graphe G' triangulation de G et un *peo*
1 Calcul du nombre d'arêtes nécessaires pour la complétion du voisinage de chaque sommet
2 **tant que** $\exists$ *un sommet non numéroté* **faire** `/* choix du prochain sommet dans l'ordre */`
3 Choix d'un sommet v non numéroté nécessitant un minimum d'arêtes pour compléter son voisinage ultérieur
4 Numéroter v
5 Compléter le sous-graphe induit par les voisins de v non numérotés
6 Mettre à jour le nombre d'ajouts d'arêtes nécessaires pour chaque sommet non numéroté

Min-Fill procède en numérotant et donc en ordonnant les sommets de G de 1 à n tout en rajoutant les arêtes nécessaires de sorte que cet ordre soit un *ordre d'élimination parfait* (ou *peo*) pour le graphe résultant G' qui sera triangulé. Dans un tel ordre, les voisins ultérieurs de chaque sommet (c'est-à-dire les voisins apparaissant après ce sommet dans l'ordre) forment une clique. La triangulation va être réalisée sur un graphe G' initialisé à G. À chaque étape, *Min-Fill* choisit parmi les sommets non numérotés, le sommet qui a besoin du minimum d'arêtes à rajouter pour compléter le sous-graphe induit [2] par ses voisins non encore ordonnés. Le processus continue jusqu'à ce que tous les sommets de G soient numérotés.

2. Le sous-graphe $G[Y]$ d'un graphe $G = (X, C)$ induit par $Y \subseteq X$ est le graphe (Y, C_Y) avec $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

La complexité en temps de *Min-Fill* est $O(n(n+e'))$ où e' le nombre d'arêtes du graphe triangulé G'. Si l'heuristique *Min-Fill* s'avère être la plus coûteuse parmi les heuristiques de l'état de l'art, son temps d'exécution est bien meilleur que celui des méthodes exactes, tout en calculant des décompositions assez proches de l'optimum. Cependant, *Min-Fill* souffre de plusieurs défauts. D'une part, elle procède d'une certaine façon à l'aveugle en se limitant à des décomptes d'arêtes sans se préoccuper des propriétés topologiques du graphe traité, du moins, explicitement. D'autre part, l'ajout d'arêtes occasionné par le principe de triangulation génère un coût temporel important. Aussi, dans les sections suivantes, nous introduisons deux nouvelles heuristiques visant à effacer ces défauts.

3 Décomposition sans triangulation

Nous proposons ici un algorithme appelé *Least-TD*, qui calcule, pour un graphe $G = (X, C)$, une décomposition arborescente en temps polynomial, bien sûr sans aucune garantie quant à son optimalité, mais sans recours à la triangulation et en prenant en compte la topologie du graphe. L'objectif est double : obtenir de meilleurs temps de calcul et éviter certains défauts de *Min-Fill* en exploitant d'éventuelles propriétés topologiques.

D'une certaine façon, *Least-TD* s'inspire de l'algorithme *BC-TD* [14, 13] qui décompose également sans triangulation. La première étape de l'algorithme calcule un premier cluster, noté E_0. X' qui notera par la suite l'ensemble des sommets déjà traités est donc initialisé à E_0. Cette première étape peut se faire facilement, en utilisant une méthode heuristique. Notons $X_1, X_2, \dots X_k$ 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 F, on notera V_i l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i. On peut noter que V_i 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). Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. L'étape suivante va consister à déterminer un sommet v de V_i qui possède un minimum de voisins dans X_i. Soit $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$ cet ensemble. On construit alors un nouveau cluster $E_i = N(v, X_i) \cup V_i$.

La figure 2 présente le calcul de E_1, le second cluster (après E_0), lors du premier passage dans la boucle. Parmi les sommets de V_1, celui qui possède un minimum de voisins dans X_1 est le sommet v puisque l'on a $N(v, X_1) = \{a\}$ et que pour les autres sommets, cet ensemble est $N(w, X_1) = N(x, X_1) = \{b, c\}$. Le cluster E_1 aura ainsi pour sommets $V_1 \cup \{a\}$. À partir de là, on va obtenir deux nouvelles composantes connexes : $X_{1_1} = \{d, e, h\}$ et $X_{1_2} = \{b, c, f, g, i, j, k\}$ qui vont alors être ajoutées à la file F. Quand X_{1_1} en sera retiré, on aura $V_i = \{a\}$, un nouveau cluster sera alors construit avec $V_i \cup \{d, e\} = \{a, d, e\}$ et $\{h\}$ sera ajouté à F. Quand X_{1_2} sera retiré de la file, on considérera l'ensemble $V_i = \{a, w, x\}$ pour lequel le sommet choisi sera a qui ne possède qu'un voisin dans X_{1_2} pour former un nouveau cluster $\{a, f, w, x\}$. Deux nouveaux ensembles seront alors insérés dans F : $\{i\}$ et $\{b, c, g, j, k\}$. On remarque qu'ainsi, l'algorithme exploite explicitement la topologie du graphe par le biais de séparateurs et de composantes connexes.

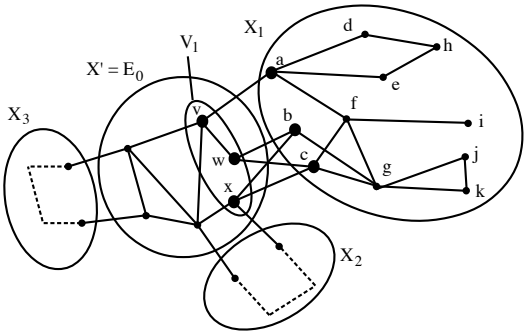

FIGURE 2 – Illustration de *Least-TD*

Algorithme 2 : *Least-TD*

Entrées : Un graphe $G = (X, C)$
Sorties : Un ensemble de clusters $E_0, \dots E_m$ d'une décomposition arborescente de G

1 Choix d'un premier cluster E_0 dans 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 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'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 Déterminer un sommet $v \in V_i$ qui possède un minimum de voisins $N(v, X_i)$ dans X_i
9 $E_i \leftarrow N(v, X_i) \cup V_i$
10 $X' \leftarrow X' \cup N(v, X_i)$
11 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

La file F va être exploitée jusqu'à la suppression de l'ensemble des sommets du graphe. Nous établissons maintenant la validité de l'algorithme, puis sa complexité temporelle.

Théorème 1 *Least-TD calcule les clusters d'une décomposition arborescente.*

Preuve : Il suffit de prouver la correction des lignes 5 à 12 de l'algorithme. Nous montrons d'abord que l'al-

gorithme s'arrête. À chaque passage dans la boucle, puisque $N(v, X_i)$ est ajouté à X', au moins un sommet de X_i 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 bien une décomposition arborescente de G. Nous le prouvons par une induction sur l'ensemble des clusters ajoutés en montrant que tous ces clusters vont induire une décomposition arborescente du graphe $G[X']$. Initialement, le premier cluster E_0 induit une décomposition arborescente du graphe $G[E_0] = G[X']$. L'hypothèse d'induction 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 de $G[X']$ 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. Si $x \in V_i$, la propriété a déjà été vérifiée par hypothèse d'induction. Si $x \in E_i \backslash V_i = N(v, X_i)$, x n'apparaît pas dans un autre cluster que E_i et la propriété est vérifiée.

Finalement, il est facile de voir que l'on obtient bien les clusters d'une décomposition arborescente du graphe $G[X']$, et par extension de G puisqu'en fin de traitement, on a $X' = X$. □

Théorème 2 *La complexité en temps de l'algorithme Least-TD est $O(n(n+e))$.*

Preuve : Les lignes 1-4 sont réalisables en temps linéaire, soit $\mathrm{O}(n+e)$, puisque le coût de calcul des composantes connexes de $G[X \backslash E_0]$ est borné par $\mathrm{O}(n+e)$. Pour le coût de la boucle (ligne 5), notons qu'il y a nécessairement moins de n insertions dans la file 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 $\mathrm{O}(n)$, soit globalement $\mathrm{O}(n^2)$.
- Ligne 7 : l'obtention du voisinage $V_i \subseteq X'$ de X_i dans G est bornée par $\mathrm{O}(n+e)$, et donc globalement par $\mathrm{O}(n(n+e))$.
- Ligne 8 : cette étape est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$. En effet, on peut profiter du traitement de la ligne 7 pour calculer pour tous les sommets de V_i le nombre de sommets voisins que chacun possède dans X_i, cela n'engendrant aucun surcoût au niveau de ce traitement.
- Lignes 9 et 10 : chacune de ces étapes est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$.
- Ligne 11 : le coût de la recherche des composantes connexes de $G[X_i \backslash E_i]$ est borné par $\mathrm{O}(n+e)$. Ainsi le coût de cette étape est $\mathrm{O}(n(n+e))$.
- Ligne 12 : l'insertion de X_{i_j} dans F est réalisable en $\mathrm{O}(n)$, soit globalement en $\mathrm{O}(n^2)$ puisqu'il y a moins de n insertions dans F.

Finalement, la complexité temporelle de l'algorithme *Least-TD* est $\mathrm{O}(n(n+e))$. □

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. Nous pouvons d'ailleurs noter que ce choix peut être réalisé par une heuristique éventuellement plus coûteuse, de sorte à obtenir un premier cluster plus pertinent, mais en se limitant à un coût de l'ordre de $\mathrm{O}(n(n+e))$ afin de ne pas accroître la complexité globale de l'algorithme. De même, le choix du sommet v sélectionné dans la ligne 8, peut être d'une importance considérable sur le plan pratique quand plusieurs sommets sont éligibles. Des heuristiques peuvent bien sûr être utilisées mais cette question ne sera pas abordée dans le cadre de ce travail.

Dans la section suivante, nous proposons un algorithme de triangulation basé à la fois sur *Min-Fill* et sur l'approche de l'algorithme *Least-TD*.

4 Trianguler en exploitant la topologie

4.1 L'effet boule de neige

Outre le fait que *Min-Fill* ne garantit pas une triangulation minimum ni même minimale [15], cette heuristique présente un phénomène bien connu qui est *l'effet boule de neige*. Il est observé au niveau de l'ajout considérable d'arêtes au graphe à trianguler (potentiellement d'ordre quadratique par rapport au nombre

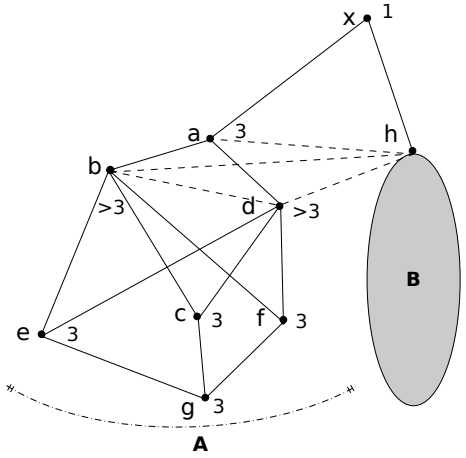

FIGURE 3 – Illustration de *l'effet boule de neige*

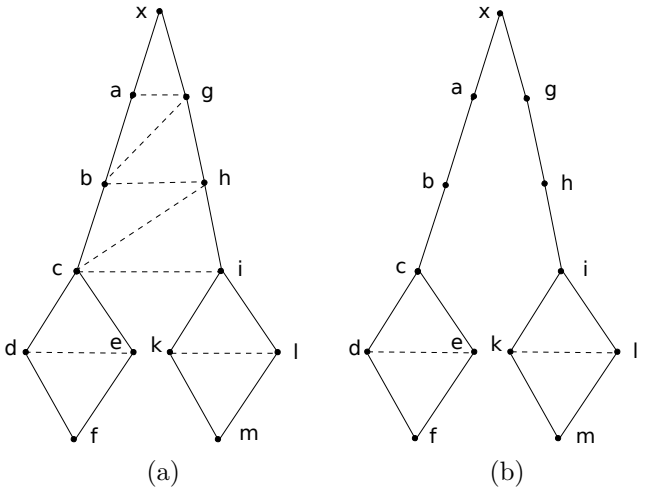

FIGURE 4 – Triangulation ne respectant pas la topologie (a) et une triangulation respectant la topologie (b)

de sommets considérés comme c'est le cas par exemple pour une grille ou *grid graph*). Souvent, cet aspect est dû à un ajout d'arêtes ne respectant pas ce qui serait souhaitable au regard de la topologie du graphe. En effet, la démarche suivie par *Min-Fill* ignore la topologie du graphe et ne tient compte que du nombre d'arêtes nécessaires pour compléter le voisinage d'un sommet. Ainsi, la présence de *l'effet boule de neige* est tout à fait possible avec *Min-Fill*.

On illustre ce phénomène par la figure 3. Les arêtes pleines sont les arêtes du graphe original et les arêtes en pointillés sont les arêtes ajoutées par la triangulation. Ce graphe est constitué de deux parties indépendantes A et B, en considérant le sommet x comme séparateur. Deux parties A et B sont *indépendantes* si aucune arête n'est présente entre un sommet de A et un sommet de B. Ce sont les deux composantes connexes induites par la suppression du sommet x dans le graphe (si l'on suppose qu'aucun sommet de B ne constitue un meilleur choix). La partie B est grisée par souci de simplification. Chaque sommet est annoté par le nombre d'arêtes à ajouter pour compléter son voisinage. Compte tenu de l'heuristique de choix, *Min-Fill* choisit le sommet nécessitant le minimum d'ajout d'arêtes, soit x dans ce cas. En choisissant le sommet x, on ajoute l'arête $\{a, h\}$. Une fois mis à jour, les sommets du graphe auraient toujours besoin du même nombre d'arêtes à rajouter. Désormais, A et B forment une seule composante connexe. Ce phénomène est susceptible de se reproduire si on choisit à l'étape suivante le sommet a, ce qui conduira à rajouter les arêtes $\{b, h\}$ et $\{d, h\}$ en plus de $\{b, d\}$. De plus, au moment de la prise en compte de h, cela pourrait amener à rajouter des arêtes entre les nouveaux voisins de h dans A et les voisins qu'il possède déjà dans B, augmentant ainsi significativement le nombre d'arêtes ajoutées. La rétroaction permettrait donc de renforcer *l'effet de boule de neige*. Une solution à ce problème consiste à traiter indépendamment d'abord les sommets de A ou de B.

La figure 4 montre deux triangulations possibles d'un graphe. Elle montre que la triangulation qui respecte la topologie du graphe ajoute moins d'arêtes que la version qui ne tient pas compte de cette topologie. En effet, *Min-Fill* établit un ordre débutant par x dans la figure 4(a) avec $O = [x; a; g; b; h; f; d; e; c; i; k; l; m]$. Cependant, dans la figure 4(b) *Min-Fill* réussit à ajouter moins d'arêtes en commençant par la partie A avec un ordre $O = [f; d; e; c; b; a; x; g; h; i; k; l; m]$.

Les inconvénients de *l'effet boule de neige* ne se limitent pas à l'ajout excessif d'arêtes qui augmentent le temps de calcul de la triangulation et plus généralement le temps d'obtention de la décomposition arborescente. Il peut y avoir aussi un impact sur la largeur arborescente w de la décomposition calculée. En effet, l'ajout excessif d'arêtes augmente potentiellement la taille des cliques du graphe en cours de triangulation, et par voie de conséquence, la taille des clusters de la décomposition, donc sa largeur.

4.2 Mieux guider Min-Fill

Dans cette partie, nous proposons une version nouvelle de *Min-Fill* appelée *Min-Fill-MG* et qui permet d'exploiter la topologie du graphe à trianguler. Cette approche applique une stratégie assez semblable à celle employée par *Least-TD*. Nous utiliserons d'ailleurs les mêmes notations. En effet, comme *Least-TD*, l'algorithme commence par le choix d'un

Algorithme 3 : *Min-Fill-MG*

Entrées : Un graphe $G = (X, C)$
Sorties : Un ensemble de clusters $E_0, \dots E_m$ d'une décomposition arborescente de G

1 Choix d'un premier cluster E_0 dans 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 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'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 Complétion de V_i dans G
9 $G_i \leftarrow Minfill(G[V_i \cup X_i])$
10 $CM \leftarrow$ Cliques maximales de G_i
11 $E_i \leftarrow$ Clique de CM qui inclut V_i
12 $X' \leftarrow X' \cup (E_i \backslash V_i)$
13 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
14 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

premier cluster noté E_0. X' notera l'ensemble des sommets déjà traités. Cet ensemble est donc initialisé avec le premier cluster E_0. Dans la suite, nous noterons $X_1, X_2, \dots X_k$ 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 en attente de traitement. Ainsi, on tient compte de la topologie du graphe en identifiant les parties indépendantes du graphe du point de vue du cluster qui vient de se former. Pour chaque élément X_i supprimé de la file F, on notera V_i l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i. V_i est donc un ensemble de sommets séparant X_i des autres sommets de X'. La différence avec *Least-TD* se présente au niveau de la construction du nouveau cluster E_i. Considérons le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. Dans un premier temps, l'ensemble V_i est complété en rajoutant dans G les arêtes absentes entre chaque paire de sommets de V_i (ligne 8). L'étape suivante (ligne 9) consiste en une triangulation de $G[V_i \cup X_i]$ utilisant *Min-Fill* et construit un graphe triangulé G_i des sommets de $V_i \cup X_i$. Les cliques maximales de G_i sont ensuite (ligne 10) mémorisées dans l'ensemble CM. Le fait que l'ensemble V_i soit une clique (cf. ligne 8) garantit l'inclusion de V_i dans au moins une des cliques maximales mémorisées dans CM. En fait, cette clique va constituer le nouveau cluster noté E_i qui sera ajouté à l'ensemble des clusters déjà calculés tout en mettant à jour l'ensemble X'. À ce stade, et comme le faisait *Least-TD*, on recalcule les nouvelles composantes connexes issues de la suppression des sommets de E_i et on les ajoute dans la file. Puis, on réitère le processus jusqu'à ce que la file soit vide.

La preuve de la validité de *Min-Fill-MG* est semblable à celle de *Least-TD* (voir le théorème 1), excepté concernant la terminaison. Nous nous limitons donc à prouver l'arrêt de *Min-Fill-MG* :

Théorème 3 *Min-Fill-MG calcule les clusters d'une décomposition arborescente.*

Preuve : Pour prouver que *Min-Fill-MG* se termine, il suffit de montrer qu'à chaque passage dans la boucle de la ligne 5, au moins un sommet de X_i sera ajouté à X'. Comme on choisit à chaque étape une clique maximale issue de la triangulation de $G[V_i \cup X_i] = G_i$ et qui contient V_i, il suffit de démontrer que ce cluster incluera V_i strictement (donc que $V_i \subsetneq E_i$).

D'après le théorème de Dirac [9], tout graphe triangulé possède au moins deux sommets *simpliciaux* (sommets dont tous les voisins forment une clique) qui ne sont pas voisins si le graphe n'est pas complet. Ainsi, on sait qu'il existe deux sommets x et x' dans G_i qui sont simpliciaux. On a alors deux possibilités :

1. x ou $x' \in V_i$. Sans manque de généralité, considérons $x \in V_i$. Puisque $x \in V_i$, on sait que x possède au moins un voisin v dans X_i. Puisque x est simplicial, l'ensemble de ses voisins forme une clique. Or, comme V_i a été complété (cf. ligne 8), x possède comme voisins au moins $(V_i \backslash \{x\}) \cup \{v\}$ qui est une clique de G_i. Ainsi, $V_i \cup \{v\}$ est aussi une clique de G_i. Cette clique est nécessairement contenue dans une clique maximale de G_i et on a ainsi au moins une clique E_i de CM telle que $V_i \subsetneq E_i$.
2. Ni x, ni x' ne sont dans V_i. Nous démontrons le résultat par induction sur la taille de X_i. L'hypothèse est que pour $|X_i| = k \geq 2$ (car ni x, ni x' ne sont dans V_i et si $|X_i| = 1$, on se retrouve dans le cas (1)), il existe une clique de G_i incluant strictement V_i. Pour la base de l'induction, on a $|X_i| = 2$. Dans ce cas, comme $G[X_i]$ est connexe, nécessairement x et x' sont voisins avant triangulation. Par conséquent, G_i forme une clique car on ne peut avoir deux sommets de X_i non voisins et simpliciaux. On suppose maintenant que la proposition est vérifiée pour $|X_i| = k \geq 2$ et on prouve qu'elle est aussi vérifiée pour $|X_i| = k+1$. Considérons x simplicial. Si x inclut V_i dans son voisinage, $V_i \cup \{x\}$ est une clique de G_i et le résultat est vérifié. Si x n'inclut pas V_i dans son voisinage, le sous-graphe de G_i induit par la suppression de x possède k sommets et il est triangulé. Dans ce cas, par hypothèse d'induction, il contient une clique incluant strictement V_i. *A fortiori*, G_i inclut cette clique et le résultat est vérifié. Il faut noter que si x inclut une partie de V_i dans son voisinage, comme x inclut nécessairement au moins un autre sommet v de X_i dans son voisinage (sinon l'hypothèse de connexité de X_i avant triangulation de $G[V_i \cup X_i]$ ne serait pas vérifiée) et que x est simplicial, v sera également voisin de

ces sommets dans le sous-graphe de G_i induit par la suppression de x.

En conséquence, on a bien un élément E_i de CM tel que $V_i \subsetneq E_i$. □

Théorème 4 *La complexité en temps de l'algorithme Min-Fill-MG est* $O(n^2(n+e'))$.

Preuve : 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 $O(n+e)$. 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 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 : $O(n^2)$ pour la même raison que *Least-TD*.
- Ligne 7 : $O(n(n+e'))$ pour la même raison que *Least-TD*, mais par contre en considérant e' comme nombre d'arêtes.
- Ligne 8 : $O(n^3)$ avec une majoration abusive.
- Ligne 9 : cette étape calcule une triangulation en utilisant l'heuristique *Min-Fill* qui a une complexité de $O(n(n+e'))$, soit globalement $O(n^2(n+e'))$.
- Ligne 10 : le calcul des cliques maximales d'un graphe triangulé peut se faire en temps linéaire (voir [11]), soit $O(n+e')$ ici. Ainsi, globalement le coût sera donc en $O(n(n+e'))$.
- Ligne 11 : comme le nombre de clusters est majoré par n, et que le coût du test d'inclusion est de $O(n)$, le coût total de cette étape peut facilement être majoré par $O(n^2)$, soit globalement par $O(n^3)$.
- Pour les lignes 12, 13 et 14 et pour les mêmes raisons que *Least-TD*, on peut majorer respectivement par $O(n^2)$, $O(n(n+e'))$ et $O(n^2)$.

Finalement, la complexité temporelle de l'algorithme *Min-Fill-MG* est de $O(n^2(n+e'))$. □

5 Évaluation expérimentale

Dans cette section, nous comparons la largeur des décompositions arborescentes produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* entre elles, mais aussi à la largeur arborescente quand celle-ci est connue. Pour *Least-TD* et *Min-Fill-MG*, le choix du premier cluster consiste à calculer une clique maximale contenant le sommet de plus grand degré dans le graphe. Les trois méthodes de décomposition sont implémentées en C++ au sein de notre bibliothèque graphique. Les expérimentations ont été réalisées sur des serveurs lames sous Linux Ubuntu 14.04 dotés chacun de deux processeurs Intel Xeon E5-2609 à 2,4 GHz et de 32 Go de mémoire. Elles portent principalement sur 1 668 instances CSP issues de la compétition CSP de 2008 [3] auxquelles s'ajoutent quelques instances issues du dépôt UCI sur les Réseaux de Croyance et quelques instances du problème de coloration de graphes. Pour les deux derniers cas, il s'agit d'instances dont la largeur arborescente est connue [2]. A noter que parmi toutes ces instances, certaines correspondent en fait à des hypergraphes dont les décompositions arborescentes sont obtenues en travaillant sur leur 2-section.

Pour comparer la largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG*, nous avons considéré 38 instances dont la largeur est connue. La table 1 présente les résultats obtenus pour quelques-unes d'entre elles. Le nombre d'instances considéré peut paraître faible, mais il faut se rappeler que déterminer la largeur arborescente d'un graphe quelconque est un problème NP-difficile. Les méthodes exactes ne sont vraiment opérationnelles que sur des graphes de petite taille. Une alternative consiste à procéder à un encadrement de la largeur arborescente par des bornes inférieures et supérieures mais, faute de disposer de bornes de qualité, cette solution n'aboutit que trop rarement à déterminer w.

Nous avons observé que *Min-Fill* et *Min-Fill-MG* trouvent une décomposition optimale pour 35 instances alors que *Least-TD* n'y parvient que pour 24 instances. Dans les cas où la décomposition n'est pas optimale, la largeur obtenue est souvent proche de l'optimum. Ceci illustre bien l'aptitude de ces méthodes heuristiques à calculer des décompositions de qualité en temps raisonnable. En effet, chaque décomposition est ici calculée en moins de 0,12 s.

À présent, nous comparons les différentes méthodes heuristiques de décomposition entre elles. Nous considérons pour cela 1 668 instances CSP issues de la compétition CSP de 2008. La table 2 fournit les largeurs des décompositions obtenues pour une sélection d'instances représentatives des différentes tendances observées. *Least-TD* produit des décompositions ayant une largeur inférieure ou égale à celles de *Min-Fill* pour 1 005 des 1 668 instances. Pour 772 de ces instances, *Least-TD* améliore la largeur des décompositions produites par *Min-Fill*. L'amélioration peut être très significative comme c'est le cas, par exemple, pour l'instance `bqwh-18-141-37_ext` avec une largeur de 54

3. Voir http ://www.cril.univ-artois.fr/CPAI08 pour plus de détails.

	Instances	n	e	w	*Min-Fill*	*Least-TD*	*Min-Fill-MG*
					w^+	w^+	w^+
(a)	alarm.net	37	65	4	4	4	4
	barley.net	48	126	7	7	10	7
	hepar2.net	70	158	6	6	6	6
	water.net	32	123	8	10	11	9
(b)	primes-30-20-3-1	100	98	3	3	3	3
	primes-30-40-2-1	100	82	2	2	3	2
	mps-red-markshare1	230	12 835	79	79	79	79
	mps-red-markshare2-1	330	35 385	149	149	149	149
(c)	david	87	406	13	13	14	13
	miles500	128	1170	22	23	29	23
	myciel3	11	20	5	5	5	5
	myciel4	23	71	10	11	11	11

TABLE 1 – Nombre de sommets et d'arêtes, largeur arborescente (optimum) et largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* pour des instances dont la largeur arborescente w est connue. Ces instances proviennent du dépôt UCI sur les Réseaux de Croyance (a), de la compétition CSP de 2008 (b) et du problème de coloration de graphes (c).

pour *Least-TD* contre 73 pour *Min-Fill*. Concernant *Min-Fill-MG*, les largeurs produites sont strictement meilleures que celles de *Min-Fill* pour 522 instances et de qualité égale pour 749. À nouveau, le gain peut être important. Par exemple, la largeur de la décomposition de l'instance `ii-8e2` est de 149 pour *Min-Fill-MG* contre 179 pour *Min-Fill*. Enfin, *Least-TD* obtient des largeurs strictement inférieures à celles de *Min-Fill-MG* pour 733 instances et égales pour 254 instances.

Concernant le temps d'exécution, *Least-TD* permet de calculer des décompositions plus rapidement qu'avec *Min-Fill*. En effet, pour 85 % (respectivement 98 %) des instances, la décomposition est calculée en moins d'une seconde (resp. une minute) par *Least-TD* contre 83 % (resp. 95 %) pour *Min-Fill*. Par contre, *Min-Fill-MG* requiert un temps d'exécution plus long que les deux premières méthodes avec 57 % (resp. 76 %). Ces résultats étaient prévisibles au regard de la complexité des algorithmes mais aussi dans la mesure où l'implémentation actuelle de *Min-Fill-MG* est assez rudimentaire.

6 Conclusion

Dans cet article, nous avons introduit deux nouvelles heuristiques *Least-TD* et *Min-Fill-MG* pour le calcul de décompositions arborescentes. L'objectif était d'améliorer *Min-Fill*, la méthode de référence dans ce cadre. L'évaluation expérimentale nous a permis de montrer que nous améliorons la qualité des décompositions sur une majorité des instances, et souvent de manière significative. De plus, les temps de calculs sont, du moins pour *Least-TD*, améliorés. Ces deux nouvelles heuristiques servent ainsi à élargir profitablement le catalogue d'algorithmes de décomposition opérationnels sur le plan pratique, celui-ci étant limité depuis de nombreuses années à la seule heuristique *Min-Fill*.

Au niveau des perspectives, il semble rester beaucoup à faire. Déja pour des améliorations de la mise en œuvre de ces deux algorithmes, en particulier pour *Min-Fill-MG* qui semble significativement optimisable au niveau des temps de calcul. Au-delà, ces deux approches ouvrent sur un champ d'investigation prometteur, en particulier concernant *Least-TD*. En effet, le schéma d'algorithme utilisé doit permettre la mise en œuvre d'heuristiques différentes en proposant d'autres choix pour la sélection associée à la construction d'un nouveau cluster. En effet, nous avons arrêté ce choix ici à une unique possibilité alors même que d'autres sont envisageables comme par exemple choisir un ensemble de sommets de X_i qui accroîtrait le nombre de sommets de V_i directement supprimés plutôt que de minimiser le nombre de sommets de X_i ayant un voisin commun dans V_i comme *Least-TD* le fait. Enfin, il reste aussi à analyser ces nouvelles décompositions au regard de leur apport pour la résolution de CSP, pour le problème de décision, mais plus particulièrement, pour les problèmes de comptage et d'optimisation, et éventuellement aussi, étudier leur impact dans le cas de la compilation.

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.

Instances	n	e	*Min-Fill*		*Least-TD*		*Min-Fill MG*	
			tps	w^+	tps	w^+	tps	w^+
composed-25-10-20-5_ext	105	620	< 0,01	**24**	< 0,01	28	0,02	**24**
graph14-f28	916	4 638	0,45	239	0,48	**229**	119,47	230
par-8-2	700	2 800	0,04	47	0,06	**41**	1,32	43
par-16-4-c	648	3 723	0,05	**43**	0,08	83	1,11	**43**
radar-10-20-4.5-0.95-98	906	10 211	0,20	**112**	0,6	121	19,70	114
s4-4-3-6	624	9 152	0,43	192	0,27	**177**	147,38	187
tsp-25-3_ext	76	400	< 0,01	**25**	< 0,01	**25**	< 0,01	**25**
bf-0432-007	2 080	7 473	0,79	145	6,47	141	32,20	**132**
ii-8e2	1 740	10 785	1,70	179	11,04	163	178,71	**149**
js-taillard-20-15-95-2	300	3 130	0,06	117	0,04	**109**	3,66	110
4-insertions-4-5	475	1 795	0,07	94	0,14	**81**	7,67	88
fapp04-0300-1	300	1 799	0,07	134	0,03	**123**	30,82	131
bqwh-18-141-37_ext	141	883	0,01	73	< 0,01	**54**	0,43	69
graph4	400	2 244	0,05	100	0,05	109	3,71	**97**
ssa-0432-003	870	2 022	0,12	34	0,46	65	2,00	**33**
games120-7	120	638	< 0,01	39	< 0,01	43	0,14	**36**

TABLE 2 – Nombre de sommets et d'arêtes, largeur des décompositions produites par *Min-Fill*, *Least-TD* et *Min-Fill-MG* pour des instances CSP de la compétition de solveurs de 2008. Les meilleures largeurs obtenues pour chaque instance sont en gras.

[2] J. Berg and M. Järvisalo. Sat-based approaches to treewidth computation : An evaluation. In *Proceedings of ICTAI*, pages 328–335, 2014.

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

[4] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs : Basic properties and comparison. *Constraints*, 4(3) :199–240, 1999.

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

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

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

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

[9] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg 25*, MR24 -57 :71–76, 1961.

[10] H. Fargier and M.-C. Vilarem. Compiling CSPs into tree-driven automata for interactive solving. *Constraints*, 9(4) :263–287, 2004.

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

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

[13] P. Jégou and C. Terrioux. Tree-decompositions with connected clusters for solving constraint networks. In *Proceedings of CP*, pages 407–423, 2014.

[14] P. Jégou and C. Terrioux. Un nouveau paramètre de graphes pour la résolution de CSP par décomposition. In *Actes des JFPC*, pages 77–88, 2014.

[15] U. Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical report, Judex R.R. Aalborg, Denmark, 1990.

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

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

[18] S. Subbarayan. An empirical comparison of csp decomposition methods. In *Proceedings of the CP Doctoral Program*, 2007.

[19] 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.

[20] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3) :410–421, 1979.