Calcul de décompositions arborescentes

Sommaire

3.1 Introduction

Nous avons déjà vu que les méthodes de résolution classiques de (W)CSP ont une complexité en temps théorique en $O(exp(n))$ avec n le nombre de variables du problème. Une deuxième approche consiste à exploiter la structure du problème en s'appuyant sur la notion de *décomposition arborescente*, par exemple. L'exploitation d'une décomposition permet de décomposer le problème original en plusieurs sous-problèmes indépendants et d'éviter beaucoup de redondances grâce à l'enregistrement de certaines informations pendant la résolution. D'un point de vue théorique, l'intérêt d'une telle approche réside dans sa complexité en temps qui est de l'ordre de $O(exp(w))$ avec w la largeur arborescente du réseau de contraintes. En pratique, ces approches sont particulièrement justifiées car il existe de nombreux problèmes réels pour lesquels w est significativement plus petit que n comme, par exemple, les problèmes d'allocation de fréquence radio [Cabon et al., 1999]. Nous avons aussi souligné que malheureusement, calculer une décomposition optimale est un problème NP-difficile. Ainsi, en pratique, la complexité en temps théorique est en $O(exp(w^+))$ avec $w^+ \geq w$ qui est une approximation de w qui correspond à la largeur de la décomposition employée. Alors, calculer efficacement une décomposition d'une largeur la plus proche possible de la largeur optimale est aujourd'hui un défi majeur. Par conséquent, les décompositions sont généralement calculées par le biais d'approches heuristiques. Ces approches ont clairement montré leur intérêt pratique par rapport aux méthodes exactes qui sont généralement inopérantes en pratique. Toutefois, notamment du point de vue de la résolution d'instances (W)CSP, ces heuristiques souffrent de multiples défauts que nous détaillons dans la section suivante. Ensuite, nous proposons, dans la section 3.3, un cadre général de calcul de décompositions, appelé *H-TD*, qui vise à minimiser, voire éviter ces défauts. Nous évaluons son intérêt pratique dans la section 3.4 avant de conclure.

3.2 Défauts des décompositions existantes

Les décompositions existantes, dont *Min-Fill* constitue l'heuristique de l'état de l'art pour la communauté CP et au-delà, visent notamment à minimiser la taille des clusters de la décomposition et ainsi la largeur w^+ de celles-ci. *Min-Fill* est surtout connue pour sa bonne approximation de la largeur arborescente w et par son temps de calcul qui reste raisonnable par rapport aux méthodes exactes de calcul de décompositions. Pour rappel, la décomposition issue de *Min-Fill* est calculée en deux étapes :

- La première étape consiste à trianguler le graphe G, c'est-à-dire à rajouter les arêtes nécessaires dans le but d'établir un ordre d'élimination parfait σ et d'obtenir un graphe triangulé G'_σ à partir de G. Pour construire l'ordre d'élimination parfait σ, *Min-Fill* ordonne les sommets de G de 1 à n en choisissant comme sommet suivant le sommet qui nécessite d'ajouter un minimum d'arêtes pour compléter son voisinage ultérieur. Généralement, *Min-Fill* casse les égalités d'une façon arbitraire. Après avoir rajouté les arêtes en question, *Min-Fill* continue à procéder de la même façon jusqu'à ce que tous les sommets de G soient numérotés. Cet algorithme est une instanciation spécifique de l'algorithme *Heuristique-H* de la partie 1.3.2.3. Il est montré par l'algorithme 3.1.

- La deuxième étape consiste, une fois le graphe G triangulé, à identifier les cliques maximales de G'_σ, grâce à σ, qui formeront chacune un cluster de la décomposition.

En ajoutant *a priori* moins d'arêtes, nous pouvons espérer produire des cliques maximales plus petites et par voie de conséquence une décomposition de largeur plus proche de

Algorithme 3.1 : Min-Fill (G)

Entrées : Un graphe $G = (X, C)$
Sorties : Un ordre d'élimination parfait σ, le graphe triangulé G'_σ

```
G'_σ ← G
Calcul du nombre d'arêtes nécessaires pour la complétion du voisinage de chaque sommet
pour i ← 1 à n faire
    Choisir un sommet non numéroté x de G'_σ qui nécessite d'ajouter un minimum d'arêtes pour compléter son voisinage non numéroté
    σ(x) ← i
    Compléter le sous-graphe induit par G'_σ[N_u(x)]
    Mettre à jour le nombre d'ajouts d'arêtes nécessaires pour chaque sommet non numéroté
retourner (σ, G'_σ)
```

l'optimum. Cependant, *Min-Fill*, comme toutes les méthodes de calcul de décompositions basées sur la triangulation, présente plusieurs inconvénients liés d'une part à la façon dont elle procède pour calculer une décomposition et d'autre part à l'intérêt de l'utilisation de cette dernière vis-à-vis de la résolution. Nous utilisons, par la suite, la notation G' au lieu de G'_σ lorsqu'il n'y a pas d'ambiguïté.

3.2.1 Effet boule de neige

Nous rappelons tout d'abord que la triangulation obtenue par les heuristiques de triangulation n'est généralement pas minimum, ni même minimale [Kjaerulff, 1990]. Autrement dit, il est possible de pouvoir supprimer des arêtes ajoutées à G' tout en gardant un graphe G' triangulé. L'ajout des arêtes additionnelles non nécessaires vis-à-vis de la triangulation entraîne forcément un surcoût au niveau du temps de triangulation en pratique. Plus important, cela peut entraîner une augmentation de la taille des clusters de la décomposition et de sa largeur w^+. En outre, ces heuristiques présentent un phénomène qui peut être qualifié de *l'effet boule de neige*. Il est caractérisé par un ajout considérable d'arêtes au graphe triangulé (potentiellement en $O(n'^3)$ pour une grille ou *grid graph* de $n' \times n'$ sommets, par exemple). Informellement, *l'effet boule de neige* consiste en une augmentation du nombre d'arêtes ajoutées au graphe entraînant à leur tour l'ajout d'un plus grand nombre d'arêtes. 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 ces heuristiques ignore la topologie du graphe. Par exemple, *Min-Fill* ne tient compte que du nombre d'arêtes nécessaires pour compléter le voisinage ultérieur d'un sommet. Au niveau de la topologie du graphe, nous nous intéressons notamment à l'indépendance entre des sous-graphes du graphe G, c'est-à-dire à l'absence d'arêtes entre deux sous-graphes de G. La plupart des heuristiques de triangulation ne prennent pas en compte, du moins explicitement, ce paramètre pendant la triangulation et peuvent potentiellement ajouter des arêtes entre deux sous-graphes indépendants, ce qui serait inutile du point de vue de la triangulation. Ainsi, la survenue de *l'effet boule de neige* est tout à fait possible avec ces heuristiques.

Nous illustrons ce phénomène par la figure 3.1 en utilisant *Min-Fill*. Toutefois, nous pouvons trouver des exemples similaires pour les autres heuristiques comme MCS [Tarjan and Yannakakis, 1984], *Lex* [Rose et al., 1976], *Minimum-Degree*[*Markowitz*, 1957]...

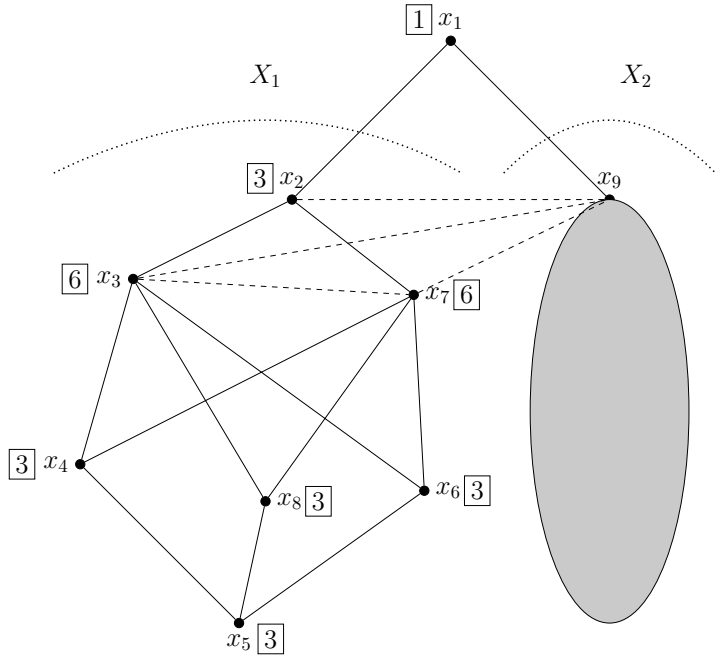

FIGURE 3.1 – Illustration de *l'effet boule de neige*.

Seuls 9 sommets du graphe sont représentés par la figure 3.1. Les arêtes pleines sont les arêtes du graphe original G et les arêtes en tirets sont les arêtes ajoutées par la triangulation. Chaque sommet est annoté d'un nombre encadré qui désigne le nombre initial (avant tout ajout d'arêtes) d'arêtes à ajouter entre les voisins de ce sommet afin de compléter le sous-graphe induit par ce voisinage. Par exemple, le sommet x_1 nécessite l'ajout d'une seule arête qui est $\{x_2, x_9\}$ et le sommet x_2 nécessite l'ajout de trois arêtes $\{x_1, x_3\}$, $\{x_1, x_7\}$ et $\{x_3, x_7\}$. Si nous supprimons x_1, ce graphe contient deux composantes connexes indépendantes X_1 et X_2 qui sont délimitées par les deux arcs en pointillés. Plus précisément, $X_1 = \{x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ et X_2 est représentée par la partie grisée par souci de simplification. Ces deux composantes connexes sont dites indépendantes puisqu'il n'existe aucune arête de G liant un sommet de l'une à un sommet de l'autre. Elles sont cependant liées grâce au sommet x_1 qui est considéré comme étant leur séparateur. Autrement dit, X_1 et X_2 sont les deux composantes connexes induites par la suppression du sommet x_1 du graphe. Nous supposons que les sommets de X_2 nécessitent tous un nombre d'ajout d'arêtes supérieur ou égal à 3. Compte tenu de l'heuristique suivie par *Min-Fill* pour ordonner les sommets, *Min-Fill* choisit le sommet nécessitant le minimum d'ajout d'arêtes, c'est-à-dire x_1 dans ce cas. En choisissant le sommet x_1, nous ajoutons l'arête $\{x_2, x_9\}$. Une fois mis à jour, les sommets du graphe ont toujours besoin du même nombre d'arêtes à ajouter. Désormais, X_1 et X_2 forment une seule composante connexe même après avoir supprimé le sommet x_1. Ce phénomène est susceptible de se reproduire si nous choisissons, à l'étape suivante, le sommet x_2, ce qui conduira à rajouter les arêtes $\{x_3, x_7\}$, $\{x_3, x_9\}$ en plus de $\{x_7, x_9\}$. Toutes les arêtes ajoutées jusqu'à ce niveau ne sont pas nécessaires au regard de la triangulation parce qu'il ne s'agit pas d'arêtes reliant des sommets non-adjacents d'un cycle (vu l'indépendance de X_1 et X_2 une fois x_1 supprimée). Une solution à ce problème consiste à traiter indépendamment d'abord les sommets de X_1 ou de X_2. Pour généraliser, l'ajout d'une arête inutile du point de vue de la triangulation entre deux composantes indépendantes pourrait engendrer de plus en plus d'ajouts non nécessaires. En effet, la rétroaction permettrait de renforcer *l'effet boule de neige*.

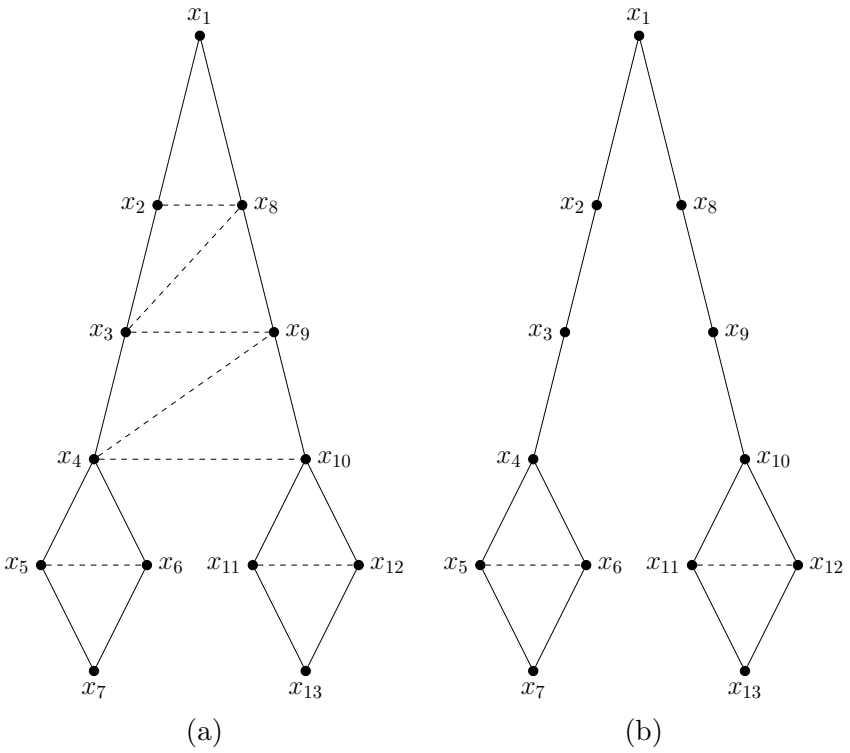

FIGURE 3.2 – Triangulation ne respectant pas la topologie (a) et une triangulation respectant la topologie (b).

Une solution possible à ce problème consiste à trianguler chaque composante connexe indépendamment des autres.

La figure 3.2 montre deux triangulations possibles d'un graphe. Dans la figure 3.2, les deux composantes connexes induites par la suppression de x_1 sont $X_1 = \{x_2, x_3, x_4, x_5, x_6, x_7\}$ et $X_2 = \{x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}\}$. Cette figure montre que la triangulation qui respecte la topologie du graphe ajoute moins d'arêtes que la version qui n'en tient pas compte. En effet, *Min-Fill* établit dans la figure 3.2(a) un ordre alternant des sommets de X_1 et de X_2, c'est-à-dire un ordre $O = [x_1, x_2, x_8, x_3, x_9, x_7, x_5, x_6, x_4, x_{10}, x_{11}, x_{12}, x_{13}]$. Cependant dans la figure 3.2(b), *Min-Fill* réussit à ajouter moins d'arêtes en traitant tout d'abord tous les sommets de X_1 avant ceux de X_2 avec un ordre $O = [x_7, x_5, x_6, x_4, x_3, x_2, x_1, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}]$.

Les inconvénients de *l'effet boule de neige* ne se limitent pas à l'ajout excessif d'arêtes qui augmente 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. En conclusion, malgré son intérêt au niveau de la minimisation du w^+, la triangulation menée par des heuristiques comme *Min-Fill* peut augmenter considérablement le temps de décomposition mais aussi sa largeur w^+.

3.2.2 Efficacité des décompositions vis-à-vis de la résolution

Grâce aux heuristiques de calcul de décompositions, les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances (W)CSP et pour le comp-

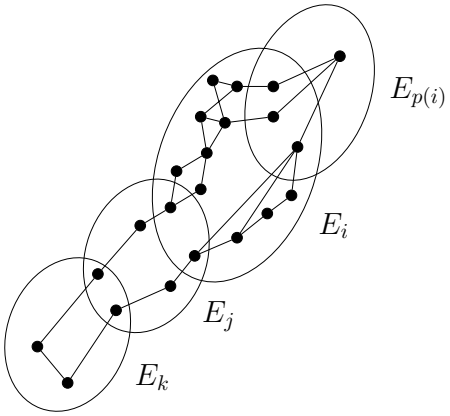

FIGURE 3.3 – Une décomposition avec le cluster E_i non connexe.

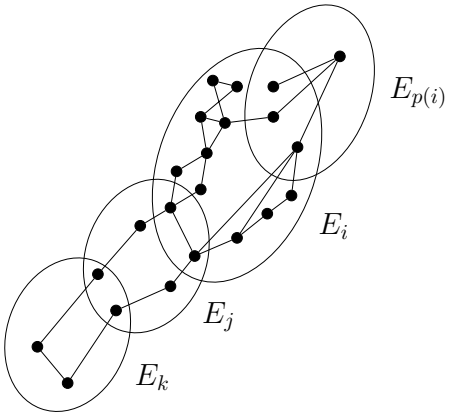

FIGURE 3.4 – Une décomposition avec le cluster E_i non connexe.

tage [Jégou and Terrioux, 2003, 2004b; Givry et al., 2006; Favier et al., 2009; Sanchez et al., 2009; Allouche et al., 2015]. Cependant, les décompositions existantes ne sont pas nécessairement les plus adaptées du point de vue de la résolution [Jégou et al., 2005; Jégou and Terrioux, 2014b]. D'ailleurs, le temps de calcul de décomposition peut effectivement largement dépasser le temps de résolution des méthodes non basées sur une décomposition dans certains cas. En plus, même si la largeur w^+ des décompositions calculées demeure très intéressante par rapport à la largeur optimale w, il ne semble pas que ce critère soit le plus pertinent au regard de l'efficacité de la résolution. En effet, les décompositions existantes, qu'elles soient ou non basées sur la triangulation, présentent les problèmes listés ci-dessous.

Limitation de la liberté de l'heuristique de choix de variables Afin de garantir une complexité en temps en $O(exp(w^+))$, les méthodes structurelles efficaces comme BTD utilisent un ordre d'affectation des variables qui est partiellement induit par la décomposition considérée. Quand des heuristiques comme $Min\text{-}Fill$ sont utilisées, cette liberté est même encore plus restreinte du fait du nombre limité de sommets propres dans les clusters. Le nombre de sommets propres peut d'ailleurs atteindre un seul sommet pour certains clusters. Dans ce cas, la liberté du choix de la prochaine variable se limite au choix du prochain cluster fils. Si, en plus, un cluster n'a qu'un seul cluster fils, cela implique une résolution exploitant un ordre de choix de variables total, donc statique. Or, il est bien

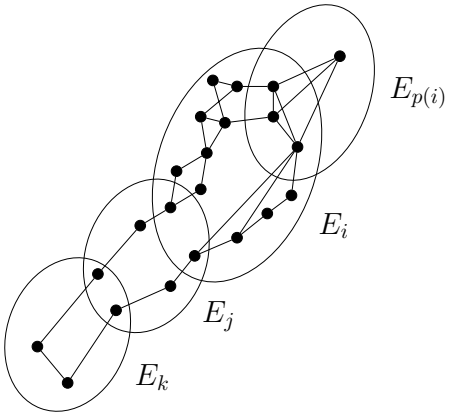

FIGURE 3.5 – Une décomposition avec le cluster E_i connexe.

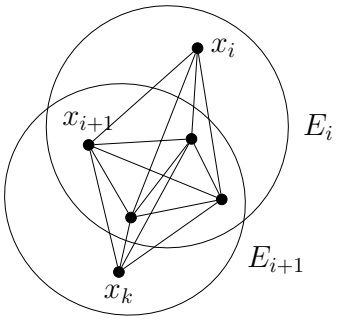

FIGURE 3.6 – Deux clusters d'une décomposition (cliques issues d'une triangulation) ayant $s = w^+ = |E_i| - 1$.

connu que pour avoir une résolution efficace, il est souhaitable d'avoir une liberté maximale pour le choix des prochaines variables à affecter et le plus de dynamicité possible à ce niveau.

Non-connexité des clusters La décomposition calculée peut éventuellement contenir des clusters non connexes (c'est-à-dire des clusters E_i tels que $G[E_i]$ possède plusieurs composantes connexes), ce qui peut conduire à une forte dégradation de l'efficacité de la résolution [Jégou and Terrioux, 2014b]. Les figures 3.3, 3.4 et 3.5 montrent trois décompositions formées de 4 clusters, E_i ayant comme père le cluster $E_{p(i)}$ et un fils noté E_j. Le cluster E_k est à son tour le fils de E_j. Nous nous focalisons sur le cluster E_i et nous distinguons trois cas possibles :

- Cas de la figure 3.3 : E_i est non connexe et le graphe $G[E_i \backslash (E_i \cap E_{p(i)})]$ contient deux composantes connexes indépendantes. Tout d'abord, la présence de plusieurs composantes connexes dans un cluster E_i peut diminuer l'efficacité de la résolution localement. En effet, si l'une des composantes connexes est insatisfaisable et que ce n'est pas le cas des autres composantes connexes, la recherche peut potentiellement explorer toutes les solutions des autres composantes connexes avant de prouver l'insatisfaisabilité de celle-ci. Dans le cadre du problème CSP ou #CSP, en utilisant BTD, les variables de E_i distribuées dans plusieurs composantes connexes doivent être assignées d'une façon cohérente. Si le sous-problème enraciné en E_i (contenant

les variables de E_i, E_j et E_k) est facile, c'est-à-dire possède beaucoup de solutions, l'affectation du cluster E_i est *a priori* rapide et s'étend plus probablement à une solution du sous-problème en question. Dans ce cas, la non-connexité du cluster E_i ne pose pas un véritable problème. Si au contraire le sous-problème n'admet que très peu de solutions, voire aucune, la résolution assigne les variables de E_i plus difficilement en passant potentiellement d'une composante connexe à l'autre. En effet, si des techniques de filtrage sont utilisées, l'affectation d'une variable x_i d'une composante connexe n'a que très peu d'influence sur les domaines des variables des autres composantes connexes. Donc, ces variables seront assignées sans vraiment tenir compte de l'affectation de x_i. C'est ainsi qu'une incohérence pourrait être détectée plus tard dans la recherche lors du traitement de l'un des fils de E_i. Cela diminue l'efficacité de la résolution en gaspillant du temps, mais aussi, en consommant de l'espace, en augmentant essentiellement le nombre de nogoods enregistrés. De même, dans le cadre du problème WCSP, la non-connexité peut avoir un impact sur la qualité de l'affectation de E_i (en termes de coût) et peut ainsi demander plusieurs retours à E_i avant de pouvoir trouver la solution optimale.

- Cas de la figure 3.4 : Le fait que le cluster E_i soit non connexe et que $G[E_i \backslash (E_i \cap E_{p(i)})]$ soit connexe signifie que le cluster E_i est non connexe à cause de la non-connexité du séparateur. Vu que les variables du séparateur sont assignées avant l'affectation des variables propres de E_i, la non-connexité n'induit alors aucun problème dans ce cas au niveau de la résolution.

- Cas de la figure 3.5 : Le fait que le cluster E_i soit connexe et que $G[E_i \backslash (E_i \cap E_{p(i)})]$ ne le soit pas nous refait revenir au cas de la figure 3.3. En effet, comme les variables du séparateur sont instanciées avant celles des variables propres de E_i cela produit un cluster E_i non connexe lors de l'instanciation de ses variables propres.

En guise de conclusion, un cluster E_i doit avoir le sous-graphe $G[E_i \backslash (E_i \cap E_{p(i)})]$ connexe. Notons que le choix du cluster racine indique l'orientation de la décomposition et ainsi les liens de parenté et filiation qui existent entre les clusters. Donc, la présence non souhaitable des cas des figures 3.3 et 3.5 dépend du choix de ce cluster. Or, il n'y a pas de garantie de pouvoir trouver un cluster racine convenable, c'est-à-dire qui garantit l'absence des cas des figures 3.3 et 3.5.

Séparateurs de grande taille Les décompositions calculées par les heuristiques de triangulation comme *Min-Fill* peuvent avoir des séparateurs de très grande taille. Souvent, la taille du plus grand séparateur s est très proche de la largeur w^+, voire même égale à w^+ parfois. La figure 3.6 montre deux clusters E_i et E_{i+1} d'une décomposition ayant la taille du plus grand séparateur $s = w^+$. Considérons que le sommet x_i est le premier sommet de E_i au regard de l'ordre d'élimination établi par *Min-Fill*. Notons x_{i+1} le premier voisin de x_i par rapport à cet ordre. Supposons que x_{i+1} est voisin du sommet x_k qui n'est pas voisin de x_i. Pendant la triangulation le voisinage de x_i est complété et formera avec x_i le cluster E_i. Lorsque x_{i+1} est traité, son voisinage est complété. En particulier, x_k est lié à tous les sommets de E_i sauf x_i (éliminé à ce stade). Un nouveau cluster E_{i+1} est alors formé partageant avec E_i tous les sommets sauf un. Ainsi, la taille du plus grand séparateur s est égal à la taille de $E_i - 1$ soit à w^+. L'augmentation de s peut mener à un coût qui s'avère prohibitif en termes d'espace mémoire. En effet, les méthodes de résolution basées sur une décomposition ont une complexité spatiale en $O(exp(s))$. En plus, l'augmentation de la taille des séparateurs peut avoir un impact sur le taux d'utilisation des informations enregistrées. En effet, l'augmentation de la taille du séparateur augmente le nombre de ses

affectations possibles ce qui rend moins probable l'utilisation des informations enregistrées pour une affectation donnée. Cela peut être problématique pour l'efficacité de la résolution et peut la détériorer significativement. D'ailleurs, la limitation de la taille des séparateurs a été démontrée cruciale pour l'efficacité en pratique dans [Jégou et al., 2005].

Contraintes à portée étirée sur plusieurs clusters En ce qui concerne la résolution, *BTD* essaye d'exploiter les contraintes *dès que possible*. Nous distinguons deux cas possibles :

- Pour une contrainte binaire, dès qu'une variable est assignée et l'algorithme de filtrage appliqué, les valeurs restantes dans le domaine de l'autre variable sont garanties d'être cohérentes puisque le filtrage répercute les conséquences de l'affectation de la variable sur la deuxième variable.
- Dans le cas de contraintes non binaires, certaines peuvent se retrouver partagées dans plusieurs clusters. En d'autres termes, les variables sur lesquelles porte cette contrainte peuvent se retrouver distribuées dans plusieurs clusters. Ceci est dû à l'emploi de la 2-section de l'hypergraphe de contraintes qui transforme toute contrainte d'arité b en une b-clique. Par construction, nous savons qu'il existe forcément un cluster qui recouvre entièrement cette b-clique. Par contre, rien n'empêche que certaines arêtes de cette b-clique soient présentes dans d'autres clusters. Selon le choix du cluster racine, la résolution pourrait exploiter en premier un cluster E_i contenant au plus $b-2$ variables d'une contrainte d'arité b. Dans ce cas, cette contrainte pourrait n'être que très partiellement exploitée lors des propagations. De plus, l'exploiter pleinement pourrait requérir d'instancier une partie des variables manquantes, ces dernières se trouvant dans un ou plusieurs autres clusters plus profonds par rapport à la décomposition. Soit E_k le cluster contenant l'avant-dernière variable de la portée, par exemple. Cela signifie que toutes les variables présentes dans les clusters situés sur le chemin entre E_i et E_k seront instanciées au préalable. Il apparaît clairement que ceci peut être fortement préjudiciable en termes d'efficacité, en particulier, si la recherche n'aboutit pas à cause des affectations des premières variables de cette contrainte.

3.3 Nouveau cadre de calcul de décompositions : H-TD

Le but de ce chapitre est de proposer une méthode de calcul de décompositions qui évite d'une part l'effet boule de neige, soit en empêchant l'utilisation de la triangulation, soit en triangulant d'une manière mieux adaptée à la structure. D'autre part, elle permet d'améliorer l'efficacité de la décomposition vis-à-vis de la résolution en tenant compte des différents critères mentionnés ci-dessus. Nous proposons donc un nouveau cadre de calcul de décompositions, appelé *H-TD* (pour *Heuristic Tree-Decomposition*), qui calcule une décomposition arborescente d'un graphe $G = (X, C)$. Il est à préciser que l'algorithme *H-TD* construit l'ensemble des clusters E de la décomposition sans calculer explicitement l'arbre T. D'ailleurs, l'arbre T peut être calculé en post-traitement grâce à l'algorithme de Jarnìk ou mieux encore en liant au fur et à mesure de la création des clusters le nœud correspondant à un cluster à celui correspondant à son cluster père. Comme les autres heuristiques de calcul de décompositions, aucune garantie n'est offerte quant à l'optimalité de la largeur obtenue. *H-TD* vise à :

- Permettre de calculer des décompositions sans forcément se baser sur la triangulation,

Algorithme 3.2 : H-TD (G)

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

1 $X_0 \leftarrow X$
2 Choix d'un premier cluster E_0 dans G
3 $X' \leftarrow E_0$
4 Soient $X_{0_1}, \ldots, X_{0_{k_0}}$ les composantes connexes de $G[X_0 \backslash E_0]$
5 $F \leftarrow \{X_{0_1}, \ldots, X_{0_{k_0}}\}$
6 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'un nouveau cluster` E_i `*/`
7 Enlever X_i la composante connexe courante de F
8 Soit $V_i \subseteq X'$ le voisinage de X_i dans G
9 Déterminer un sous-ensemble $X''_i \subseteq X_i$ tel que $X''_i \cup V_i$ constitue un cluster définitif de la décomposition
10 $E_i \leftarrow X''_i \cup V_i$
11 $X' \leftarrow X' \cup X''_i$
12 Soient $X_{i_1}, X_{i_2}, \ldots, X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
13 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \ldots X_{k_i}\}$

- Améliorer le temps de décomposition en pratique et la complexité théorique temporelle,

- Éviter les inconvénients de la triangulation en exploitant les propriétés topologiques du graphe,

- Permettre la paramétrisation de la décomposition dans le but de pouvoir prendre en compte plusieurs critères comme la largeur de la décomposition w^+ ou s la taille maximale des séparateurs.

H-TD s'inspire de l'algorithme *BC-TD* (pour *Bag Connected Tree-Decomposition*) [Jégou and Terrioux, 2014b] qui vise à calculer des décompositions sans triangulation formées de clusters connexes. D'ailleurs, *BC-TD* peut être considéré comme une des heuristiques de *H-TD* ou une paramétrisation possible de ce cadre avec en plus l'avantage d'empêcher de calculer des clusters inclus dans d'autres ce qui constitue un gain en temps non négligeable.

3.3.1 Schéma général

Le calcul des décompositions arborescentes par *H-TD* s'effectue en général grâce à un parcours du graphe G en se basant sur les propriétés liées aux séparateurs et à leurs composantes connexes associées. Ce calcul est constitué de deux éléments principaux, le calcul du premier cluster E_0 et le calcul du cluster suivant E_i à partir d'une composante connexe X_i et de son voisinage noté V_i. *H-TD* est décrit dans l'algorithme 3.2. Il prend en entrée un graphe $G = (X, C)$. Notons que nous partons de l'hypothèse que G est un graphe connexe. Si ce n'est pas le cas, il est traité comme plusieurs graphes connexes, un par composante connexe de G. L'ensemble de sommets X de G est considéré comme étant la composante connexe initiale X_0 (ligne 1 de l'algorithme 3.2). *H-TD* retourne l'ensemble de clusters $E = \{E_0, \ldots, E_l\}$ de la décomposition obtenue.

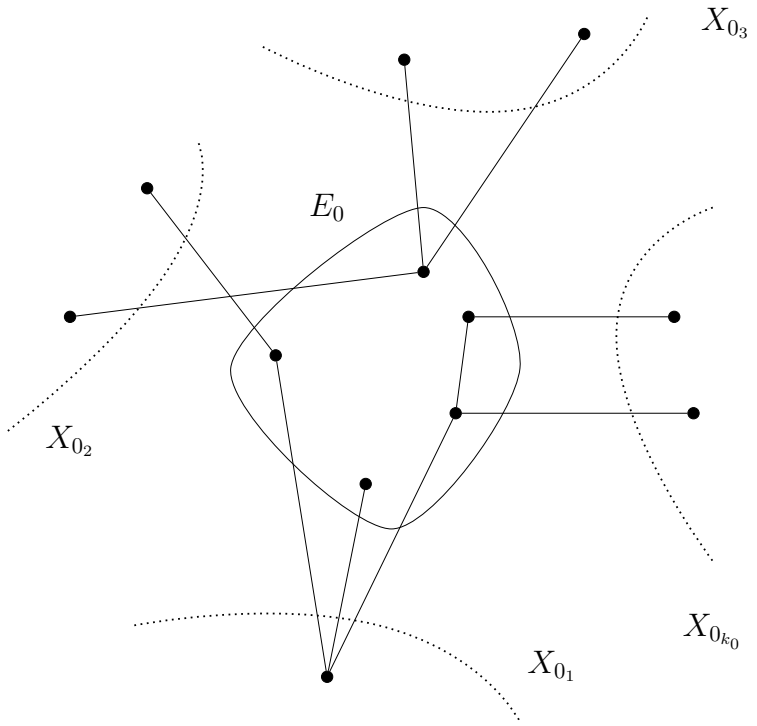

FIGURE 3.7 – Calcul du premier cluster E_0.

Calcul du premier cluster E_0 et traitement postérieur La figure 3.7 illustre le calcul du premier cluster E_0. Elle montre uniquement les arêtes liant des sommets de E_0 à une composante induite par la suppression de ce dernier de G. Le calcul du premier cluster E_0 se fait grâce à des techniques heuristiques (ligne 2 de l'algorithme 3.2). Ce premier cluster peut consister en une clique de grande ou de petite taille mais peut aussi être un séparateur de taille bornée. Le choix de E_0 dépend notamment des critères souhaités à l'égard de la décomposition calculée. Ainsi, si les clusters de la décomposition obtenue doivent être connexes, alors il est impératif que E_0 soit connexe. Notons aussi que l'heuristique de choix du premier cluster doit avoir un coût raisonnable afin de ne pas augmenter le temps de calcul de *H-TD*. X' représente l'ensemble des sommets déjà considérés de G et est initialisé à E_0 (ligne 3). Chaque sommet de X' est alors présent dans au moins un cluster parmi les clusters déjà formés de la décomposition. $X_{0_1}, X_{0_2}, \ldots, X_{0_{k_0}}$ représentent les composantes connexes relatives à $G[X_0 \backslash E_0]$ (ligne 4), induites par la suppression dans G des sommets de E_0. Chacune de ces composantes connexes est insérée dans une file d'attente F (ligne 5).

Calcul du cluster suivant E_i et traitement postérieur Le deuxième élément est le calcul du cluster suivant E_i. Il est calculé à partir de la composante connexe courante X_i extraite de la file d'attente F. Pour chaque élément X_i supprimé de F (ligne 7), V_i représente l'ensemble de sommets de X' adjacents à au moins un sommet de X_i (ligne 8). Nous pouvons constater que V_i est un séparateur du graphe G vu que la suppression des sommets de V_i de G déconnecte G (X_i étant déconnecté du reste de G). V_i constituera ainsi l'intersection entre le cluster E_i à calculer et son cluster parent $E_{p(i)}$. D'ailleurs, la construction de l'algorithme garantit qu'il y a un cluster $E_{p(i)}$ contenant l'ensemble V_i. Nous considérons maintenant 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 (ligne 9) peut être paramétrée. Elle recherche un sous-ensemble

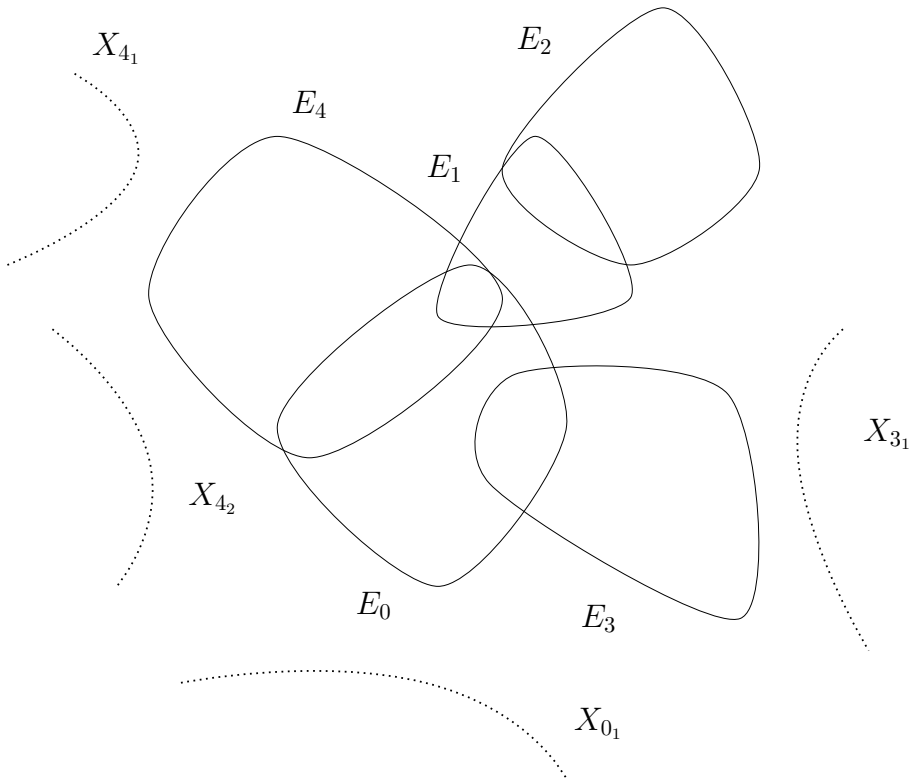

FIGURE 3.8 – La fin du calcul du cluster E_4.

de sommets $X''_i \subseteq X_i$ tel que $X''_i \cup V_i$ sera un nouveau cluster E_i de la décomposition. Pour que $X''_i \cup V_i$ soit un cluster de la décomposition, il faut garantir que l'ensemble des sommets de $X''_i \cup V_i$ ne sera pas inclus dans de futurs clusters de la décomposition. Certes, le fait d'avoir un cluster inclus dans un autre n'a pas d'impact sur la validité de la décomposition. Cependant, cette propriété permet d'empêcher de calculer un cluster contenant un cluster déjà calculé et ayant une taille plus grande. Ainsi, ceci permet d'économiser le temps de calcul des clusters inclus dans d'autres clusters ainsi que le temps de leur suppression notamment que leur présence peut éventuellement être contre-productive vis-à-vis de la résolution. Cela peut être garanti s'il existe au moins un sommet v de V_i tel que tous ses voisins dans X_i figurent dans X''_i. Plus précisément, si $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$, nous devons garantir l'existence d'un sommet v de V_i avec $N(v, X_i) \subseteq X''_i$. En d'autres termes, nous garantissons par cette condition que le cluster E_i n'est pas inclus dans d'autres clusters de la décomposition comme c'est le cas de la figure 3.9. En effet, si $V_{i-1} = \{x_1\}$, la construction du cluster E_{i-1} ne respecte pas la condition vu que E_{i-1} ne contient pas tous les voisins de x_1, soit x_2, x_3 et x_4. Ainsi, à l'étape suivante, $X_i = \{x_4\}$ et $V_i = \{x_1, x_2, x_3\}$ d'où $E_i = \{x_1, x_2, x_3, x_4\}$. De ce fait, $E_{i-1} \subset E_i$. Notons que cette condition est suffisante mais pas nécessaire. D'ailleurs, il se peut qu'il existe $X'''_i \subset X''_i$ tel que le cluster $E_i = X'''_i \cup V_i$ obtenu est garanti non inclus dans de futurs clusters. En outre, d'autres conditions peuvent garantir que le cluster E_i ne sera pas inclus dans de futurs clusters. C'est le cas de l'heuristique *Min-Fill-MG* que nous allons décrire plus tard. Nous définissons alors un nouveau cluster $E_i = X''_i \cup V_i$ (ligne 10). Puis, nous rajoutons à X' les sommets de X''_i (ligne 11) avant de calculer les composantes connexes du sous-graphe issu de la suppression des sommets de E_i dans $G[X_i]$, notées $X_{i_1}, X_{i_2}, \ldots X_{i_{k_i}}$ (ligne 12). Le calcul de ces composantes connexes permet la prise en compte de la topologie du graphe et empêche la création de clusters contenant des sommets de différentes composantes

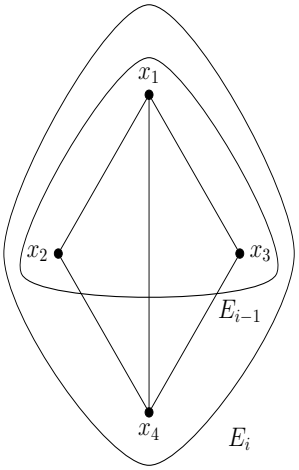

FIGURE 3.9 – Deux clusters d'une décomposition ayant $E_{i-1} \subset E_i$.

connexes. Chacune de ces composantes connexes est insérée à son tour dans F pour être traitée ultérieurement (ligne 13). Ce processus est répété jusqu'à ce que la file F soit vide. La figure 3.8 montre la progression du calcul des clusters. Après le choix du cluster E_0, les clusters E_1, E_2, E_3 et E_4 ont été calculés. Le prochain cluster à calculer est le cluster E_5. *H-TD* fera alors un choix entre les composantes connexes disponibles dans F, à savoir X_{0_1}, X_{3_1}, X_{4_1} ou X_{4_2}.

La figure 3.10 récapitule finalement les étapes principales de *H-TD*.

Selon les heuristiques choisies aux lignes 2 et 9, *H-TD* peut posséder la propriété suivante :

Propriété 3 *Soit $E_0, \dots E_l$ l'ensemble de clusters de la décomposition calculée par H-TD selon l'heuristique employée. $\forall E_i, \nexists E_j$ tel que $i \neq j$ et $E_i \subseteq E_j$.*

En effet, les heuristiques choisies à la ligne 2 et 9 doivent garantir cette propriété. Si l'heuristique du calcul du cluster E_0 ne tient pas compte de cette propriété, le cluster E_0 calculé peut être éventuellement inclus dans d'autres clusters. Cependant, comme l'heuristique de la ligne 9 garantit que le cluster calculé ne sera pas inclus dans de futurs clusters, seulement le cluster E_0 serait inclus dans d'autres clusters et peut éventuellement être supprimé.

H-TD possède la propriété suivante :

Propriété 4 *Pour tout séparateur V_i de la décomposition calculée par H-TD, G admet au moins une composante connexe pleine associée à V_i.*

En effet, chaque sommet v de V_i est lié à au moins un sommet de X_i (par construction). X_i est alors une composante connexe pleine associée à V_i. Si G admet en plus une deuxième composante connexe pleine, alors V_i est minimal.

3.3.2 Heuristiques proposées non basées sur une triangulation

Nous déclinons tout d'abord ce schéma selon cinq heuristiques non basées sur la triangulation : H_1-*TD*, H_2-*TD*, H_3-*TD*, H_4-*TD* et H_5-*TD* [Jégou et al., 2015a, 2016b]. Pour chacune de ces heuristiques le sous-ensemble X_i'' choisi à la ligne 9 est tel qu'il existe au moins un sommet $v \in V_i$ avec $N(v, X_i) \subseteq X_i''$. Cette condition garantit que le cluster E_i nouvellement construit ne sera pas inclus dans de futurs clusters (comme illustré dans la figure 3.9). L'objectif des différentes heuristiques est de :

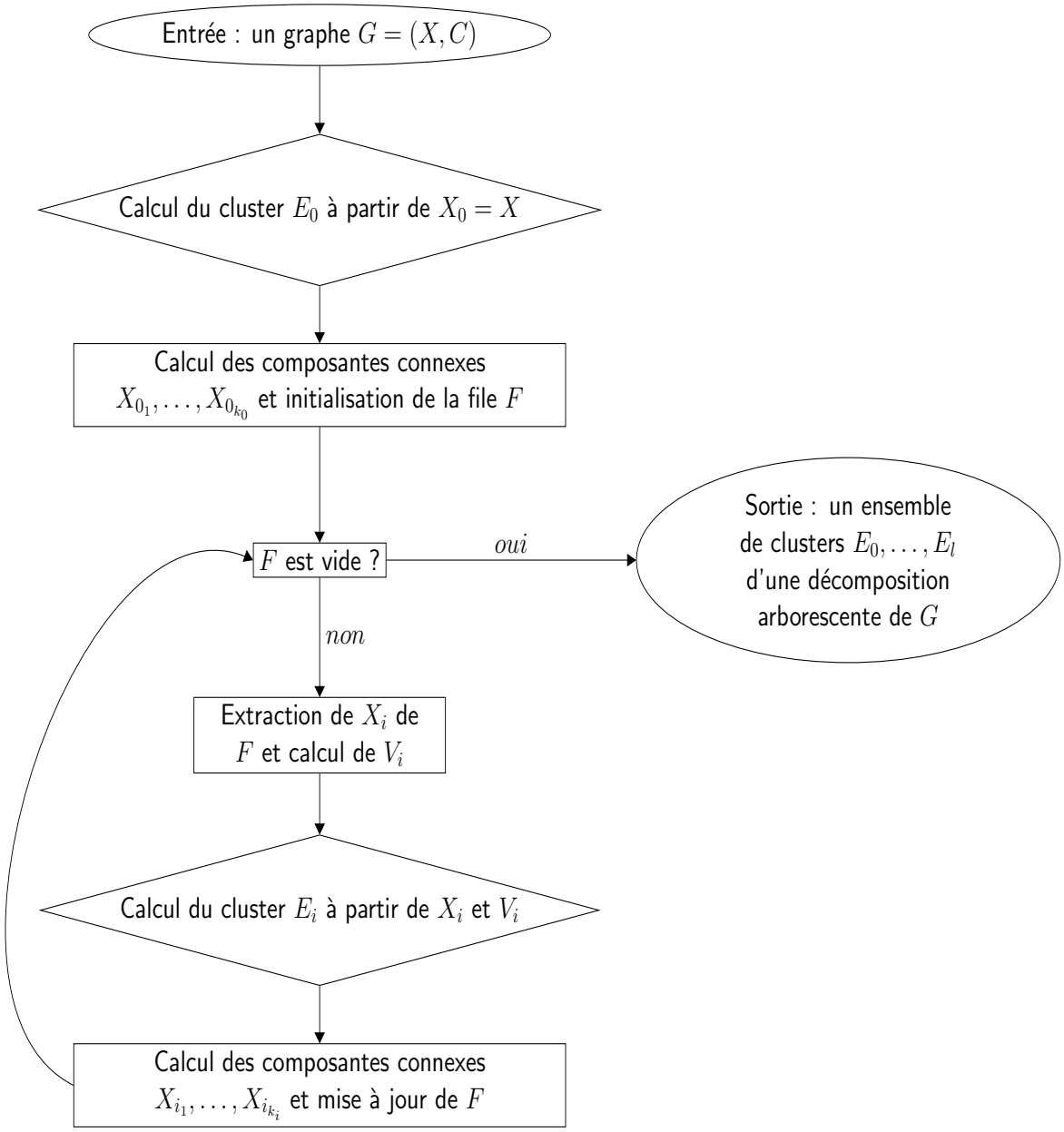

FIGURE 3.10 – Le schéma général de H-TD.

- Minimiser la largeur de la décomposition : H_1-TD
- Calculer des décompositions plus adaptées à la résolution d'instances (W)CSP
  - H_2-TD : calculer une décomposition avec des clusters connexes,
  - H_3-TD : calculer une décomposition telle que les clusters ont plusieurs fils,
  - H_4-TD : limiter la taille des séparateurs,
  - H_5-TD : limiter la taille des séparateurs tout en essayant d'augmenter leur nombre.

Par la suite, nous illustrons la façon dont chaque heuristique calcule le prochain cluster E_i grâce au graphe de la figure 3.11. Le premier cluster choisi E_0 est l'ensemble $\{x_1, x_2, x_3, x_4\}$.

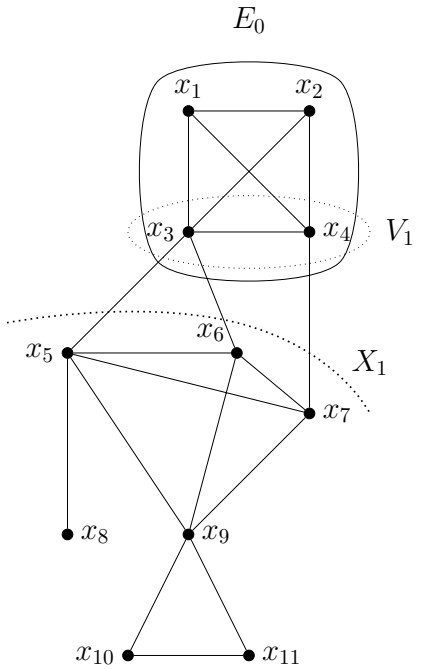

FIGURE 3.11 – Un graphe à décomposer par H-TD.

La composante connexe courante induite par la suppression des sommets de E_0 de G est $X_{0_1} = X_1 = \{x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$ (représentée par un arc en pointillés). Son voisinage correspondant est $V_1 = \{x_3, x_4\}$ (représenté par une ellipse en pointillés). Nous expliquons alors comment est calculé le prochain cluster E_1 à partir de V_1 et de X_1 selon l'heuristique considérée.

- H_1-TD : Comme indiqué ci-dessus, cette heuristique vise à minimiser localement la taille du prochain cluster E_i. Cela permet essentiellement de minimiser la largeur de la décomposition obtenue, c'est-à-dire le paramètre central en termes de complexité théorique. C'est pourquoi nous recherchons le plus petit sous-ensemble X_i'' pour qu'il forme avec V_i le prochain cluster E_i. Cela est possible en cherchant un sommet v de V_i ayant le minimum de voisins dans X_i. Ce sommet v choisi, $X_i'' = N(v, X_i)$ et ainsi $E_i = X_i'' \cup V_i$.

  Dans la figure 3.11, le séparateur V_1 contient les deux sommets x_3 et x_4. Dans X_1, x_3 a deux voisins (x_5 et x_6) et x_4 en a un seul (x_7). Ainsi, le sommet v de V_1 qui a le moins de voisins dans X_1 est x_4. Nous construisons alors le cluster $E_1 = \{x_3, x_4, x_7\}$. Nous calculons ensuite la composante connexe $X_{1_1} = \{x_5, x_6, x_8, x_9, x_{10}, x_{11}\}$ et nous ajoutons X_{1_1} à F.

- H_2-TD : Cette heuristique calcule le prochain cluster E_i qui doit être connexe vu l'importance montrée de ce paramètre au regard de la résolution. Elle se base sur la notion de *connected tree-width* récemment introduite en théorie des graphes dans [Müller, 2012; Diestel and Müller, 2017; Hamann and Weißauer, 2016]. Elle s'inspire de l'algorithme BC-TD présenté en détails dans [Jégou and Terrioux, 2014b] tout en empêchant le calcul de nouveaux clusters qui incluent des clusters déjà calculés. Notons que garantir la connexité de E_i lors de sa résolution nécessite que $G[E_i \backslash (E_i \cap E_{p(i)})]$ soit connexe. Or, si les redémarrages pendant la résolution sont exploitées, ils peuvent éventuellement s'accompagner d'un changement du cluster racine. Ainsi,

$E_{p(i)}$ est susceptible de changer. Garantir la connexité de $G[E_i \backslash (E_i \cap E_{p(i)})]$ quel que soit le cluster $E_{p(i)}$ peut augmenter considérablement la taille des clusters. C'est pourquoi nous nous contentons de la connexité du cluster E_i.

Nous montrons le calcul du prochain cluster E_1 du graphe représenté dans 3.11. Plusieurs possibilités sont envisageables. E_1 peut par exemple être $E_1 = \{x_3, x_4, x_5, x_6\}$ qui induit un sous-graphe connexe avec $N(v, X_1) = N(x_3, X_1) = \{x_5, x_6\} = X_i'' = \{x_5, x_6\}$. Les deux composantes connexes induites sont ainsi : $X_{1_1} = \{x_8\}$ et $X_{1_2} = \{x_7, x_9, x_{10}, x_{11}\}$.

- H_3-TD : Cette heuristique construit le prochain cluster E_i grâce aux propriétés topologiques du graphe. Elle vise à identifier plusieurs composantes indépendantes du graphe et à les séparer. Pour y parvenir, H_3-TD ajoute des sommets au cluster suivant E_i grâce à un parcours en largeur du graphe en commençant par les sommets de V_i. C'est ainsi que les voisins de V_i dans X_i constituent le premier niveau de X_i, les voisins des voisins de V_i dans X_i constituent le deuxième niveau de X_i et ainsi de suite. Alors, au niveau $u = 1$, $E_{i_1} = N[V_i, X_i]$ et au niveau u ($u > 1$), $E_{i_u} = N[E_{i_{u-1}}, X_i]$. Pour calculer E_i, cette heuristique continue à avancer à travers les niveaux et à ajouter des sommets jusqu'à ce que X_i soit séparée en plusieurs composantes connexes. En d'autres termes, H_3-TD continue à avancer jusqu'à un niveau $u = U$ tel que $G[X_i \backslash E_{i_U}]$ contient strictement plus qu'une composante connexe ou bien jusqu'à ce que $G[X_i \backslash E_{i_U}]$ soit vide.

  Si nous considérons l'exemple de la figure 3.11, $E_{1_1} = \{x_3, x_4, x_5, x_6, x_7\}$, $E_{1_2} = \{x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$ et $E_{1_3} = \{x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$ (dans cet exemple, il y a 3 niveaux au maximum). Dans cet exemple, le parcours en largeur est stoppé au niveau $U = 1$ parce que $G[X_1 \backslash E_{1_1}]$ contient deux composantes connexes $X_{1_1} = \{x_8\}$ et $X_{1_2} = \{x_9, x_{10}, x_{11}\}$ qui seront ajoutées à F. D'où, $E_1 = \{x_3, x_4, x_5, x_6, x_7\}$.

- H_4-TD : Cette heuristique vise à limiter la taille des séparateurs de la décomposition qui est un paramètre crucial à prendre en compte pour permettre une résolution plus efficace. Pour y parvenir, elle considère le paramètre S qui représente une borne supérieure sur la taille maximale permise pour un séparateur de la décomposition ($s \leq S$). Cette heuristique ajoute de nouveaux sommets à E_i de la même manière que H_3-TD. Néanmoins, H_4-TD s'arrête au niveau $u = U$ tel que $G[X_i \backslash E_{i_U}]$ ne contient pas des composantes connexes ayant des séparateurs de taille supérieure à S.

  Avec l'exemple de la figure 3.11, supposons que $S = 2$. $E_{1_1} = \{x_3, x_4, x_5, x_6, x_7\}$ et induit deux composantes connexes $X_{1_1} = \{x_8\}$ ayant un séparateur de taille 1 (contenant uniquement le sommet x_5) et $X_{1_2} = \{x_9, x_{10}, x_{11}\}$ ayant un séparateur de taille 3 (contenant x_5, x_6 et x_7). Ainsi, nous ne pouvons pas nous arrêter au niveau $u = 1$ puisque tous les séparateurs n'ont pas une taille au maximum égale à 2. Cependant, $E_{1_2} = \{x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$ induit une seule composante connexe $X_{1_1} = \{x_{10}, x_{11}\}$ ayant un séparateur contenant un seul sommet x_9. Ainsi, le parcours peut s'arrêter au niveau $U = 2$ et ainsi $E_1 = \{x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$.

- H_5-TD : Comme H_4-TD, cette heuristique vise aussi à limiter la taille des séparateurs de la décomposition tout en raffinant cette décomposition. En effet, elle permet de détecter plus de séparateurs de taille au plus S. Là où H_4-TD doit atteindre un niveau de la recherche en largeur auquel tous les séparateurs sont de taille au plus S, H_5-TD va être plus opportuniste. Si à un niveau donné, une nouvelle composante connexe apparaît avec un séparateur de taille au plus S, ce séparateur va être pris

en compte. Sa composante connexe sera rajoutée à la file d'attente, et la recherche va se poursuivre sur le reste du sous-graphe en cours d'exploitation, exceptée bien sûr, la partie associée à cette nouvelle composante connexe.

Nous illustrons le calcul du prochain cluster E_1 par H_5-TD sur l'exemple de la figure 3.11. Nous supposons aussi que $S = 2$. $E_{1_1} = \{x_3, x_4, x_5, x_6, x_7\}$ et induit deux composantes connexes $X_{1_1} = \{x_8\}$ ayant un séparateur de taille 1 (contenant uniquement le sommet x_5) et $X_{1_2} = \{x_9, x_{10}, x_{11}\}$ ayant un séparateur de taille 3 (contenant x_5, x_6 et x_7). X_{1_1} induit alors un séparateur d'une taille inférieure à 2. La recherche va certes se poursuivre mais l'existence du séparateur $\{x_5\}$ est exploitée. Cela va permettre, non pas d'identifier un nouveau cluster, mais de circonscrire la poursuite de la recherche en largeur à un sous-graphe duquel a été enlevé le sommet x_8 car il n'est plus connecté au reste des autres sommets. La recherche en largeur se poursuit donc mais sur la partie de X_1 qui n'a pas été parcourue, soit $\{x_9, x_{10}, x_{11}\}$ (dont $\{x_8\}$ a été supprimé). Nous constatons alors que le niveau suivant, uniquement constitué du sommet x_9, forme un séparateur de taille 1, donc inférieure à la borne fixée $S = 2$. Le parcours en largeur est alors stoppé et le nouveau cluster est maintenant constitué : $E_1 = \{x_3, x_4, x_5, x_6, x_7, x_9\}$. Les composantes connexes induites par la suppression de E_1 de X_1 sont : $X_{1_1} = \{x_8\}$ et $X_{1_2} = \{x_{10}, x_{11}\}$. Elles sont ajoutées à F.

3.3.3 Heuristique à base de triangulation

Une heuristique qui a recours à la triangulation est aussi proposée. Elle exploite et définit une nouvelle version de Min-$Fill$ que nous appelons Min-$Fill$-MG (pour Min-$Fill$ Mieux Guidé). À l'instar de Min-$Fill$, elle vise à minimiser la largeur de la décomposition w^+. Elle permet d'exploiter la topologie du graphe à décomposer contrairement à Min-$Fill$. Dans ce cas, la ligne 9 de l'algorithme 3.2 consiste à :

- Considérer le graphe $G[V_i \cup X_i] = G_i$,
- Compléter V_i dans G_i,
- Calculer le graphe G'_i qui représente l'application de Min-$Fill$ au graphe G_i,
- Calculer CM l'ensemble de cliques maximales de G'_i,
- Choisir E_i qui consiste en une clique de CM incluant V_i.

La complétion de V_i consiste à ajouter dans G les arêtes absentes entre chaque paire de sommets de V_i. L'étape suivante consiste à trianguler $G_i = G[V_i \cup X_i]$ en utilisant Min-$Fill$ et à construire un graphe triangulé G'_i des sommets de $V_i \cup X_i$. Nous précisons qu'il n'est pas nécessaire d'ordonner tous les sommets du graphe selon Min-$Fill$ mais uniquement ceux de V_i. Une fois les sommets de V_i ordonnés, les autres sommets restants peuvent être ignorés sachant que nous sommes uniquement intéressés par les cliques contenant V_i. Les cliques maximales de G'_i sont ensuite mémorisées dans CM. Comme $G[V_i]$ est une clique, l'inclusion de V_i dans au moins une des cliques mémorisées dans CM est garantie. En fait, cette clique va constituer le nouveau cluster noté E_i qui sera rajouté à l'ensemble des clusters déjà calculés.

Nous considérons maintenant le graphe de la figure 3.11 et nous illustrons le calcul de cluster suivant E_1 par Min-$Fill$-MG. L'application de Min-$Fill$ au graphe $G[V_1 \cup X_1]$ génère l'ensemble des cliques (si nous calculons toutes les cliques) $CM = \{\{x_3, x_4, x_7\}, \{x_3, x_5, x_6, x_7\}, \{x_5, x_6, x_7, x_9\}, \{x_5, x_8\}, \{x_9, x_{10}, x_{11}\}\}$. Comme $V_1 \subset \{x_3, x_4, x_7\}$ alors $E_1 =$

$\{x_3, x_4, x_7\}$ et donc $X_{1_1} = \{x_5, x_6, x_8, x_9, x_{10}, x_{11}\}$ sera ajoutée à F. Le fait que E_i soit une des cliques maximales de CM garantit que E_i ne serait jamais inclus dans un cluster formé ultérieurement.

3.3.4 Validité de H-TD

Pour chacune des heuristiques mentionnées ci-dessus, qu'elle soit basée sur une triangulation ou pas, le calcul de E_i est suivi du calcul de nouvelles composantes connexes induites par la suppression des sommets de E_i dans $G[X_i]$. Cela permet notamment de séparer les composantes indépendantes du graphe et de calculer des décompositions plus adaptées à la topologie du graphe. Ces composantes connexes sont insérées à leur tour dans la file d'attente F pour être traitées ultérieurement. Le graphe à décomposer sera entièrement recouvert par des clusters lorsque F devient vide. Ainsi, nous obtenons une collection de clusters correspondant à une décomposition valide du graphe. Avant de s'intéresser à la validité de la décomposition induite par les clusters calculés, nous démontrons que les heuristiques H_i-TD et l'heuristique Min-$Fill$-MG satisfont la propriété 3.

Lemme 3 *Les heuristiques H_i-TD satisfont la propriété 3.*

Preuve : Nous démontrons cette propriété en montrant qu'à la ligne 9 de l'algorithme 3.2, nous garantissons que $N(v, X_i) \subseteq X_i''$. Cette propriété est en effet garantie par construction. En ce qui concerne l'heuristique H_1-TD, elle consiste à choisir un sommet v de V_i ayant le plus petit nombre de voisins dans X_i. Une fois v choisi, tout le voisinage de v est ajouté à E_i. Ainsi, $N(v, X_i) \subseteq X_i''$. Quant à H_2-TD, la connexité est vérifiée tout en veillant à ce qu'il existe au moins un sommet v tel que $N(v, X_i) \subseteq X_i''$. En ce qui concerne les heuristiques H_3-TD, H_4-TD et H_5-TD, quel que soit le niveau auquel elles s'arrêtent, elles vont inclure le niveau $u = 1$ de X_i dans E_i. Ainsi, E_i inclut forcément $N(V_i, X_i)$. Par conséquent, il existe nécessairement $v \in V_i$ tel que $N(v, X_i) \subseteq X_i''$. Nous en déduisons que $N(v, X_i) \subseteq X_i''$ pour toutes les heuristiques H_i-TD. Le fait qu'au moins un nouveau sommet de X_i est ajouté au cluster E_i qui n'est présent dans aucun cluster déjà formé, garantit que E_i ne peut être égal à aucun autre cluster. En outre, comme tout le voisinage du sommet v est ajouté à E_i, le sommet v n'apparaîtra dans aucun cluster pouvant être formé ultérieurement à partir des nouvelles composantes connexes calculées $X_{i_1}, X_{i_2}, \ldots X_{i_{k_i}}$. En conséquence, $\forall E_i, \nexists E_j$ tel que $i \neq j$ et $E_i \subseteq E_j$. □

Lemme 4 *L'heuristique Min-$Fill$-MG satisfait la propriété 3.*

Preuve : Nous commençons par démontrer qu'au moins un nouveau sommet de X_i sera ajouté à E_i. Comme nous choisissons à chaque étape une clique maximale issue de la triangulation de $G[V_i \cup X_i] = G_i$ qui contient V_i, il suffit de démontrer que ce cluster inclura V_i strictement (donc que $V_i \subset E_i$). Tout graphe triangulé possède au moins deux sommets *simpliciaux* (sommets dont l'ensemble des voisins forment une clique) qui ne sont pas voisins si le graphe n'est pas complet (cf. théorème de Dirac [Dirac, 1961]). Ainsi, nous savons qu'il existe deux sommets x et x' dans G_i' qui sont simpliciaux. Nous avons alors deux possibilités :

- x ou $x' \in V_i$. Sans manque de généralité, considérons $x \in V_i$. Puisque $x \in V_i$, nous savons que x possède au moins un voisin v dans X_i. De plus x est simplicial alors l'ensemble de ses voisins forme une clique. Or, comme V_i a été complété, 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 nous avons ainsi au moins une clique E_i de CM telle que $V_i \subset E_i$.

- 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 $|X_i| = 1$, nous nous retrouvons dans le premier cas), il existe une clique de G'_i incluant strictement V_i. Pour la base de l'induction, nous avons $|X_i| = 2$. Dans ce cas, comme $G[X_i]$ est connexe, nécessairement x et x' sont voisins avant triangulation. Après triangulation, x et x' seront toujours voisins puisque la triangulation ne supprime pas d'arêtes. Ainsi nous concluons qu'une fois triangulé G_i n'aura pas deux sommets simpliciaux non voisins. Par conséquent, d'après la contraposée du théorème de Dirac, G'_i forme une clique car nous ne pouvons pas avoir deux sommets de X_i non voisins et simpliciaux. Ainsi, il existe bien une clique de G'_i induit par tous les sommets G'_i incluant strictement V_i.

  Nous supposons maintenant que la proposition est vérifiée pour $|X_i| = k \geq 2$ et nous prouvons qu'elle est aussi vérifiée pour $|X_i| = k + 1$. Considérons maintenant x. 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 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, nous avons bien un élément E_i de CM tel que $V_i \subset E_i$. Comme au moins un nouveau sommet de X_i est ajouté au cluster E_i qui n'est présent dans aucun cluster déjà formé, E_i, cela garantit que E_i n'est égal à aucun autre cluster.

En outre, comme le cluster E_i est une clique maximale de G'_i, alors il ne sera inclus dans aucun autre cluster formé ultérieurement. En effet, supposons que E_i induit une composante connexe X_{i_p} dont le voisinage s'avère être E_i. Alors tout sommet de E_i est lié à au moins un sommet de X_{i_p}. L'ensemble de ces arêtes est noté $C_{\subseteq}$. Cependant, comme G'_i est le graphe triangulé correspondant à $G[V_i \cup X_i]$ et comme tout sous-graphe d'un graphe triangulé l'est aussi, le sous-graphe de G'_{i_p} correspondant à $G[E_i \cup X_{i_p}]$ est triangulé. Ce dernier contient nécessairement les arêtes de $C_{\subseteq}$ puisque la triangulation ne supprime pas d'arêtes. Or, E_i est une clique maximale. En termes de décomposition arborescente, E_i constituerait alors un cluster. En plus, les arêtes de $C_{\subseteq}$ doivent être forcément recouvertes. Nous savons également que les sommets de $C_{\subseteq}$ qui n'appartiennent pas à E_i appartiennent à la même composante connexe si le cluster E_i est supprimé de $G[E_i \cup X_{i_p}]$. Ainsi, ces sommets appartiennent aux clusters de $Desc(E_j)$ telle que E_j est un cluster fils de E_i dans la décomposition enracinée en E_i. Finalement, nous pouvons constater que ces sommets ne peuvent alors appartenir qu'à un seul cluster en raison de la troisième condition d'une décomposition arborescente. Nous obtenons alors un cluster contenant tous les sommets de E_i en plus d'autres sommets ; il inclut ainsi E_i. Cela est contradictoire avec le fait que E_i est une clique maximale. Par conséquent, E_i ne peut pas être inclus dans de futurs clusters. □

Théorème 5 *H-TD calcule les clusters d'une décomposition arborescente et garantit qu'aucun cluster n'est inclus dans un autre.*

Preuve : Il suffit de prouver la correction des lignes 6 à 13 de l'algorithme. Nous montrons d'abord que l'algorithme s'arrête. À chaque passage dans la boucle, 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 $E_0, E_1, \dots E_l$ 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 recouverte par le cluster E_i.

(iii) Nous pouvons 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 $x \in E_i \backslash V_i$, x n'apparaît pas dans un autre cluster que E_i et la propriété est vérifiée.

Finalement, il est facile de voir que nous obtenons bien les clusters d'une décomposition arborescente du graphe $G[X']$, et par extension de G puisqu'en fin de traitement, nous avons $X' = X$. En addition, le fait que les heuristiques sur lesquelles se base *H-TD* satisfont la propriété 3 garantit qu'il n'existe aucun cluster de la décomposition arborescente qui est inclus dans un autre. □

3.3.5 Complexité de H-TD

Nous nous intéressons, dans cette partie, à la complexité du cadre de calcul de décompositions *H-TD*. Comme nous l'avons déjà vu, ce cadre peut être instancié par une heuristique de calcul du premier cluster E_0 et une heuristique qui se charge de calculer le prochain cluster E_i. La liste des heuristiques possibles risque d'être très longue. En effet, ces heuristiques peuvent avoir des objectifs très diversifiés cherchant à satisfaire des critères très différents l'un de l'autre allant des plus simples au plus sophistiqués. Ainsi, selon ce critère, l'heuristique peut être plus ou moins coûteuse. C'est pourquoi la complexité de *H-TD* dépend donc de la complexité de ces deux heuristiques.

Théorème 6 *La complexité en temps de H-TD est* $O(n(n+e)) + Complexité(H_{E_0}) + Complexité(H_{E_i})$ *avec* $Complexité(H_{E_0})$ *la complexité de l'heuristique de calcul du cluster* E_0 *et* $Complexité(H_{E_i})$ *la complexité de l'heuristique de calcul du cluster* E_i*.*

Preuve : La ligne 1 et les lignes 3-5 sont réalisables en temps linéaire, soit $O(n+e)$, puisque le coût de calcul des composantes connexes de $G[X_0 \backslash E_0]$ est borné par $O(n+e)$.

La complexité de la ligne 2 dépend de l'heuristique employée pour le calcul du cluster E_0. Sa complexité est alors $Complexité(H_{E_0})$. Nous analysons maintenant le coût de la boucle (ligne 6). Tout d'abord, notons qu'il y a nécessairement moins de n insertions dans la file F car, à chaque passage, dans la boucle, nous sommes assurés 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 7 : l'obtention du premier élément X_i de F est faisable en $O(n)$, c'est-à-dire en $O(n^2)$ globalement.
- Ligne 8 : l'obtention du voisinage $V_i \subseteq X'$ de X_i dans G est faisable en $O(n+e)$, et donc en $O(n(n+e))$ globalement.
- Ligne 9 : le coût de cette étape dépend de l'heuristique choisie pour le calcul du prochain cluster E_i et est $Complexité(H_{E_i})$.
- Lignes 10 et 11 : chacune de ces étapes est réalisable en $O(n)$, c'est-à-dire globalement en $O(n^2)$.
- Ligne 12 : le coût de la recherche des composantes connexes de $G[X_i \backslash E_i]$ est en $O(n+e)$. Ainsi le coût de cette étape est en $O(n(n+e))$.
- Ligne 13 : l'insertion de X_{i_j} dans F est réalisable en $O(n)$, c'est-à-dire globalement en $O(n^2)$ puisqu'il y a moins de n insertions dans F. □

Notons que si les heuristiques choisies ont des complexités ne dépassant pas $O(n(n+e))$ la complexité globale de H-TD serait alors en $O(n(n+e))$. Le coût de calcul de la décomposition reste alors raisonnable. En particulier, cette complexité dépend uniquement du nombre initial d'arêtes e et non pas de e' comme c'est le cas de certaines méthodes de calcul de décomposition basées sur la triangulation.

Nous allons voir dans ce qui suit que les heuristiques H_1-TD, H_2-TD, H_3-TD, H_4-TD et H_5-TD peuvent garantir une telle complexité.

Théorème 7 *La complexité en temps de H_1-TD, H_2-TD, H_3-TD, H_4-TD et H_5-TD est $O(n(n+e))$.*

Preuve : Pour chaque heuristique, nous fournissons le coût de la ligne 9.

- Pour H_1-TD, cette étape est réalisée en $O(n)$, c'est-à-dire en $O(n^2)$ globalement.
- Pour H_2-TD, cette étape est réalisée en $O(n+e)$, c'est-à-dire en $O(n(n+e))$ globalement. En effet, trouver X_i'' tel que $V_i \cup X_i''$ soit connexe est faisable en $O(n(n+e))$ globalement. Plus précisément, le test de connexité de $G[E_i]$ qui est faisable en $O(n+e)$, est fait une seule fois pour chaque sommet ajouté.
- Pour H_3-TD, la recherche des sommets au niveau u, c'est-à-dire les voisins des sommets au niveau $u-1$ dans $X_i \backslash E_{i_{u-1}}$ est réalisable en $O(n+e)$. Le coût du calcul des composantes connexes de $G[X_i \backslash E_{i_{u-1}}]$ est faisable en $O(n+e)$. Comme dans le pire cas, chaque sommet est à un niveau différent, cette étape est faite au plus n fois, c'est-à-dire un coût global en $O(n(n+e))$.

- Le cas de H_4-TD est similaire à celui de H_3-TD. En effet, la recherche des sommets au niveau u est faisable en $O(n+e)$. En outre, le calcul des composantes connexes de $G[X_i \backslash E_{i_{u-1}}]$ et de leur séparateurs associés est réalisable en $O(n+e)$. D'où le coût global de cette étape est en $O(n(n+e))$.

- Pour H_5-TD, l'analyse est identique à H_4-TD.

Finalement, la complexité temporelle des algorithmes H_1-TD, H_2-TD, H_3-TD, H_4-TD et H_5-TD est de $O(n(n+e))$. □

Théorème 8 *La complexité en temps de l'algorithme Min-Fill-MG est* $O(n^2(n+e'))$ *avec* e' *le plus grand nombre d'arêtes obtenu suite à une triangulation de Min-Fill.*

Preuve : La différence entre *Min-Fill-MG* et les autres heuristiques H_i-TD réside dans l'emploi de la triangulation lors de la construction du prochain cluster E_i. Plus précisément, à la ligne 9 de l'algorithme 3.2, le sous-ensemble X_i'' est calculé grâce à *Min-Fill*. En effet, *Min-Fill-MG* calcule une triangulation de $G_i = G[V_i \cup X_i]$ en utilisant l'heuristique *Min-Fill* et construit le cluster E_i qui n'est qu'une clique issue de la triangulation contenant V_i. Comme *Min-Fill* a une complexité de $O(n(n+e'))$, alors globalement le coût est en $O(n^2(n+e'))$ avec e' le plus grand nombre d'arêtes obtenu suite à une triangulation d'un G_i par *Min-Fill*. Ainsi, la complexité temporelle de l'algorithme *Min-Fill-MG* est de $O(n^2(n+e'))$. □

Notons que les complexités des heuristiques H_i-TD et *Min-Fill-MG* supposent que la complexité du calcul du premier cluster n'excèdent par leur complexités globales.

3.4 Étude expérimentale

Dans cette section, nous évaluons le cadre de calcul de décompositions H-TD. D'une part, nous nous intéressons à son utilisation dans l'optique de calculer des décompositions minimisant la largeur. D'autre part, nous évaluons son intérêt du point de vue de la résolution d'instances CSP et WCSP.

Benchmark utilisé Nous considérons 3 ensembles d'instances :

- I_1 : Le premier ensemble I_1 rassemble 33 instances issues du dépôt UCI sur les Réseaux de Croyance et quelques instances du problème de coloration de graphes. Il s'agit d'instances dont la largeur arborescente est connue [Berg and Järvisalo, 2014]. Le nombre n de sommets des graphes varie de 11 à 128 sommets et le nombre e d'arêtes varie de 20 à 2 556 arêtes.

- I_2 : Le deuxième ensemble I_2 porte principalement sur 1 859 instances CSP issues de la compétition CSP de 2008 [CP0, 2008]. Pour établir cette sélection, nous avons exclu les instances ayant des décompositions triviales (par exemple les instances ayant un graphe complet) ainsi que les instances ayant des contraintes globales. Les instances retenues représentent la majorité des familles d'instances. Le nombre n de variables varie de 6 à 14 930 et le nombre de contraintes m varie de 5 à 77 305. Le nombre e d'arêtes varie de 25 à 2 641 722 arêtes.

- I_3 : Le troisième ensemble I_3 rassemble le benchmark d'instances disponible à l'adresse `http://genoweb.toulouse.inra.fr/~degivry/evalgm`. Il est composé de plus de 3 000 instances incluant, entre autres, des modèles graphiques stochastiques de l'évaluation UAI de 2008 et 2010, des instances de la compétition PIC 2011 et des instances de la compétition MiniZinc 2012 et 2013. Nous avons écarté les instances résolues pendant la phase de prétraitement qui sera décrite dans la partie 3.4.3 pour obtenir au final un benchmark composé de 2 444 instances. Le nombre n de variables varie de 4 à 137 808 avec des domaines d'une taille variant de 2 à 503. Le nombre de fonctions de coûts varie de 11 à 1 799 150 et leur arité est comprise entre 2 à 372.

À 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.

Réalisation des expérimentations Les méthodes de décompositions sont implémentées en C++ au sein de notre bibliothèque. Toutes les expérimentations ont été réalisées sur des serveurs lame sous Linux Ubuntu 14.04 dotés chacun de deux processeurs Intel Xeon E5-2609 à 2,4 GHz et de 32 Go de mémoire.

3.4.1 Minimisation de la largeur de la décomposition calculée

Dans cette sous-section, nous nous focalisons sur la largeur des décompositions produites par *Min-Fill* et par les deux heuristiques de *H-TD* visant la minimisation de la largeur de la décomposition : H_1-TD et *Min-Fill-MG*. Nous évoquons tout d'abord le protocole expérimental.

3.4.1.1 Protocole expérimental

Pour *Min-Fill* et *Min-Fill-MG*, les égalités rencontrées au niveau du nombre d'arêtes à ajouter pour compléter le voisinage d'un sommet, sont cassées grâce à un critère sur 3 niveaux qui privilégie :

- le sommet ayant le plus petit nombre de voisins non encore numérotés dans le graphe courant triangulé,
- en cas d'égalité, le sommet ayant le plus petit degré dans le graphe courant triangulé,
- en cas d'égalité, le sommet apparaissant le premier dans l'ordre lexicographique.

Pour H_1-TD, le choix du premier cluster (E_0) consiste à calculer une clique maximale dans le graphe. Pour *Min-Fill-MG*, le choix du premier cluster consiste à prendre la première clique calculée par *Min-Fill*.

Les expérimentations portent sur les ensembles d'instances I_1 et I_2.

3.4.1.2 Observations et analyse des résultats

Comparaison de la largeur obtenue par rapport à la largeur exacte Afin de comparer la largeur des décompositions arborescentes produites par *Min-Fill*, H_1-TD et *Min-Fill-MG* par rapport à la largeur exacte, nous considérons les instances du benchmark I_1 dont la largeur arborescente est connue. La table 3.1 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. D'ailleurs, les méthodes exactes ne sont vraiment opérationnelles

Instances	n	e	w	*Min-Fill*	H_1-*TD*	*Min-Fill-MG*
				w^+	w^+	w^+
myciel4	23	71	10	**11**	**11**	**11**
mildew	35	80	4	**4**	7	**4**
queen6_6	36	290	25	26	**25**	**25**
barley	48	126	7	**7**	10	**7**
eil51.tsp	51	140	8	10	**9**	**9**
celar02	100	311	10	**10**	11	**10**
miles500	128	1170	22	23	28	**22**
child	20	30	3	**3**	**3**	**3**
hailfinder	56	99	4	**4**	6	**4**
hepar2	70	158	6	**6**	**6**	**6**

TABLE 3.1 – Nombre de sommets et d'arêtes, largeur arborescente (optimum) et largeur des décompositions produites par *Min-Fill*, H_1-*TD* et *Min-Fill-MG* pour des instances dont la largeur w est connue. Ces instances proviennent du dépôt UCI sur les Réseaux de Croyance et du problème de coloration de graphes (benchmark I_1). Les meilleures largeurs obtenues pour chaque instance sont en gras.

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 très rarement à déterminer w.

Nous avons observé que *Min-Fill-MG* trouve une décomposition optimale pour 25 instances, et que *Min-Fill* trouve une décomposition optimale pour 23 instances tandis que H_1-*TD* n'y parvient que pour 12 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 $0,06$ s. Cela justifie l'intérêt de ces heuristiques par rapport aux méthodes exactes. À titre d'exemple, l'instance *eil51* nécessite près de deux heures pour démontrer que la largeur arborescente est 8 sur une machine Intel Xeon quad core sous Linux Ubuntu 10.04 à 2,8 GHz et 32 Go de mémoire dans [Berg and Järvisalo, 2014].

Comparaison des largeurs obtenues selon l'heuristique considérée À présent, nous comparons les différentes heuristiques de décomposition entre elles. Nous considérons pour cela les instances du benchmark I_2. La table 3.2 fournit les largeurs des décompositions obtenues pour une sélection d'instances représentatives des différentes tendances observées. En comparant H_1-*TD* à *Min-fill*, nous nous apercevons que H_1-*TD* produit des décompositions ayant une largeur inférieure ou égale à celles de *Min-Fill* pour 1 119 des 1 859 instances. Pour 786 de ces instances, H_1-*TD* améliore la largeur des décompositions produites par *Min-Fill*. L'amélioration peut être très significative comme c'est la cas, par exemple, pour l'instance bqwh-18-141-37_ext avec une largeur de 49 pour H_1-*TD* contre 73 pour *Min-Fill*. Concernant *Min-Fill-MG*, les largeurs produites sont strictement meilleures que celles de *Min-Fill* pour 613 instances et de qualité égale pour 924 instances. 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, H_1-*TD* obtient des largeurs strictement inférieures à celles de *Min-Fill-MG* pour 751 instances, égale pour 322 instances et strictement supérieures pour 786 instances.

En ce qui concerne le temps d'exécution, *Min-Fill* calcule l'ensemble des décomposi-

Instances	n	e	*Min-Fill*		H_1-*TD*		*Min-Fill-MG*	
			tps	w^+	tps	w^+	tps	w^+
1-insertions-6-4	607	6 337	0,25	187	0,02	**173**	3,5	201
ii-8e2	1 740	10 785	0,57	179	0,05	167	7,76	**149**
3-fullins-4-7	405	3 524	0,07	123	0,01	**96**	0,5	106
aim-200-1-6-3	400	1 119	0,03	91	0,01	90	0,55	**87**
blast-floppy1-6	719	7 818	0,02	**44**	0,02	61	0,26	45
bmc-ibm-02-04	2 810	14 610	0,34	150	0,16	**139**	5,5	148
cache-inv8-ucl	2 355	7 896	0,15	71	0,1	147	3,5	**67**
graph4	400	2 244	0,05	100	0,01	101	0,43	**96**
js-taillard-20-100-2	400	4 180	0,12	152	0,02	**134**	1,64	143
elf-rf8	6 059	23 172	1,71	177	1,19	430	32,24	**165**
fapp17-0300-9	300	2 056	0,1	153	$<$ 0,01	149	1,9	**146**
geo50-20-d4-75-85_ext	50	418	$<$ 0,01	**17**	$<$ 0,01	21	$<$ 0,01	**17**
hanoi4	1 436	9 031	0,86	226	0,13	353	26,36	**203**
bqwh-18-141-37_ext	141	883	0,01	73	$<$ 0,01	**49**	0,1	67
ii-32c3	558	9 198	0,06	**92**	0,06	101	1,37	**92**
will199GPIA-6	701	6 772	0,12	106	0,02	**103**	2,65	105

TABLE 3.2 – Nombre de sommets et d'arêtes, largeur des décompositions produites par *Min-Fill*, H_1-*TD* et *Min-Fill-MG* pour des instances CSP de la compétition de solveurs de 2008 (benchmark I_2). Les meilleures largeurs obtenues pour chaque instance sont en gras.

tions en 12 460 *s* contre 21 643 *s* pour *Min-Fill-MG* tandis que H_1-*TD* n'a besoin que de 928 *s*. En effet, les instances décomposées en moins d'une seconde représentent 88, 27 % des instances pour *Min-Fill*, 94, 99 % pour H_1-*TD* et 70, 84 % pour *Min-Fill-MG*. En outre, les instances décomposées en moins d'une minute représentent 96, 28 % pour *Min-Fill*, 99, 94 % pour H_1-*TD* et 90, 47 % pour *Min-Fill-MG*. Ainsi, H_1-*TD* est clairement plus rapide que *Min-Fill* tandis que *Min-Fill-MG* requiert un temps d'exécution plus long que les deux autres méthodes. Ces résultats étaient prévisibles au regard de la complexité des algorithmes. Le coût élevé de *Min-Fill-MG* vient essentiellement du coût de la triangulation réalisée à chaque étape pour calculer le prochain cluster E_i. La qualité de la décomposition calculée par H_1-*TD* est parfois moins bonne que celle calculée par *Min-Fill-MG*. Ceci est notamment dûe au choix de l'ensemble X_i'' qui exige l'ajout de tous les voisins d'un sommet v de V_i dans X_i à X_i''. En particulier, si le sommet v est l'unique sommet de V_i il sera nécessairement choisi quel que soit son nombre de voisins dans X_i. Ce cas pourrait être très pénalisant pour le calcul de w^+.

Bilan En conclusion, H_1-*TD* et *Min-Fill-MG* ont clairement montré leur capacité à produire souvent des décompositions de meilleure qualité que celles obtenues par la méthode de référence *Min-Fill*. H_1-*TD* a notamment mis en évidence l'intérêt des méthodes de décomposition ne s'appuyant pas sur la triangulation, non seulement par la qualité des décompositions calculées, mais aussi et surtout par le temps de décomposition en permettant de décomposer l'ensemble des instances 13 fois plus rapidement que *Min-Fill*. Quant à *Min-Fill-MG*, elle produit particulièrement des décompositions de meilleure qualité sur une majorité des instances, mais est malheureusement pénalisée par un temps de décomposition qui peut être parfois long. D'ailleurs, *Min-Fill-MG* ne parvient pas à calculer certaines décompositions dans un temps limite de 15 minutes ce qui n'est pas sans

impact sur le nombre total d'instances où *Min-Fill-MG* améliore les décompositions de *Min-Fill* et H_1-TD. H_1-TD et *Min-Fill-MG* prouvent finalement que la prise en compte de la topologie du graphe permet d'améliorer le calcul des décompositions en qualité et/ou en temps.

3.4.2 Efficacité de la résolution pour le problème CSP

Nous comparons ici les décompositions calculées du point de vue de l'efficacité de la résolution des instances CSP. Par souci de simplicité, nous notons parfois H_i l'heuristique H_i-TD. Nous évoquons tout d'abord le protocol expérimental.

3.4.2.1 Protocole expérimental

Au niveau des décompositions, nous considérons :

- *Min-Fill* : c'est l'heuristique de l'état de l'art qui est connue pour sa bonne approximation de la largeur arborescente,
- H_1-TD et *Min-Fill-MG* : leur intérêt au regard de la minimisation de la largeur de la décomposition w^+ a été mis en évidence dans la partie 3.4.1,
- H_2-TD : pour sa garantie de construire des clusters connexes,
- H_3-TD : pour sa construction de clusters ayant plusieurs fils,
- H_4-TD et H_5-TD : pour leur contrôle de la taille des séparateurs.

L'heuristique H_2-TD a été introduite dans [Jégou and Terrioux, 2014b] et présentée en plusieurs versions selon le choix des sommets suivants permettant de construire le cluster connexe. Nous choisissons ici l'heuristique qui consiste à choisir comme sommet suivant le sommet ayant le nombre maximum de voisins dans le cluster (ce qui correspond à l'heuristique $NV4$ dans [Jégou and Terrioux, 2014b]). Au niveau des heuristiques H_4-TD et H_5-TD, elles exploitent une taille maximale de séparateurs dépendant de l'instance, fixée à 5% du nombre de variables de l'instance dans la limite d'au moins 4 variables et au plus 50 et sont notées $H_4^{5\%}$ et $H_5^{5\%}$.

En ce qui concerne les algorithmes exploités, comme référence des méthodes énumératives, nous considérons l'algorithme MAC+RST exploitant les techniques de redémarrage et l'enregistrement des nogoods comme décrit dans les parties 2.2.4.3 et 2.2.4.7. Pour la résolution en se basant sur une décomposition arborescente, nous prenons en compte BTD que nous désignons par BTD-MAC vu son exploitation de MAC pour résoudre chaque cluster comme décrit dans la partie 2.2.5.1. Aussi, nous exploitons BTD-MAC+RST intégrant en plus de BTD-MAC les techniques de redémarrage et d'enregistrements de nogoods comme décrit dans la partie 2.2.5.1. Nous choisissons comme premier cluster racine le cluster ayant le plus grand rapport nombre de contraintes sur la taille du cluster moins un. L'intuition derrière cette heuristique découle du principe *first-fail*. Nous essayons de choisir le cluster induisant le sous-problème le plus contraint. Il en est de même pour le choix du cluster racine à chaque redémarrage. En effet, à chaque redémarrage le cluster racine choisi est celui ayant la somme maximale de poids de contraintes (pondération de contraintes de dom/wdeg) qui intersectent le cluster. Comme le cluster racine est le premier cluster à être affecté, veiller à ce que ses variables soient jugées parmi les plus pertinentes peut être très avantageux pour la suite de la résolution. Au niveau du choix du prochain cluster, lorsqu'un cluster possède plusieurs fils, celui qui a le séparateur de

Algorithme	*Min-Fill*		H_1		*Min-Fill-MG*	
	#rés	temps	#rés	temps	#rés	temps
BTD-MAC	1 348	34 416	1 305	28 768	1 281	39 113
BTD-MAC+RST	1 507	33 632	1 485	28 433	1 425	40 117

TABLE 3.3 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-MAC* et *BTD-MAC+RST* selon les décompositions minimisant la largeur w^+.

Algorithme	H_2		H_3		$H_4^{5\%}$		$H_5^{5\%}$	
	#rés	temps	#rés	temps	#rés	temps	#rés	temps
BTD-MAC	1 424	21 860	1 487	28 792	1 527	26 170	1 524	26 143
BTD-MAC+RST	1 536	22 854	1 558	26 418	1 588	24 038	1 586	24 520

TABLE 3.4 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-MAC* et *BTD-MAC+RST* selon les décompositions satisfaisant d'autres critères.

plus petite taille est choisi d'abord. La cohérence d'arc est appliquée en prétraitement via *AC*-3^{rm} et durant la résolution via *AC*-8^{rm} (visitées dans la partie 2.2.4.3). En effet, les expérimentations montrent que ces deux méthodes de filtrage sont parmi les méthodes les plus efficaces. Toutes les méthodes de résolution utilisent l'heuristique dom/wdeg pour choisir la prochaine variable à instancier. En ce qui concerne les algorithmes exploitant les redémarrages, ils utilisent une politique de redémarrage géométrique avec un ratio de 1,1 et 100 comme nombre initial de retours en arrière autorisés.

Pour chaque instance, le temps d'exécution (incluant le temps de calcul de la décomposition dans le cas échéant) est limité à 15 minutes et l'utilisation mémoire à 16 Go.

Les expérimentations portent sur l'ensemble d'instances I_2.

3.4.2.2 Observations et analyse des résultats

Comparaison générale des décompositions Les deux tables 3.3 et 3.4 montrent le nombre d'instances résolues parmi les 1 859 instances et le temps total d'exécution pour *BTD-MAC* et pour *BTD-MAC+RST* selon les décompositions exploitées. La table 3.3 montre les résultats pour les décompositions visant à minimiser la largeur de la décomposition w^+ tandis que la table 3.4 s'occupe des décompositions dont le but est de satisfaire des critères jugés pertinents au regard de l'efficacité de la résolution comme la connexité des clusters, le nombre de fils d'un cluster et la taille des séparateurs de la décomposition. Nous remarquons que quel que soit l'algorithme utilisé (sans ou avec redémarrages) l'exploitation des décompositions minimisant la largeur n'est pas bénéfique à l'égard de la résolution. Certes, le nombre d'instances résolues par *BTD-MAC+RST* augmente par rapport à *BTD-MAC* pour *Min-Fill*, H_1 et *Min-Fill-MG*. En effet, l'exploitation des redémarrages permet, entre autres, de créer plus de dynamicité au niveau de l'heuristique de choix de variables en rendant possible le changement du cluster racine à chaque redémarrage. Ainsi, cela compense en partie les restrictions imposées par la décomposition notamment celles qui visent à minimiser la taille des clusters, c'est-à-dire ayant le moins de variables propres comme décrit dans la partie 3.2.2. Cependant, l'emploi des redémarrages semble tout de même insuffisant au vu du nombre d'instances résolues par les décompositions cherchant à satisfaire d'autres critères comme le montre la table 3.4.

Algorithme	*Min-Fill*	H_2	H_3	$H_4^{5\%}$	$H_5^{5\%}$
BTD-MAC+RST	350 432	313 554	297 318	267 938	270 220

TABLE 3.5 – Temps d'exécution en secondes pour *BTD-MAC+RST* selon les décompositions pour toutes les instances de I_2 (non résolues incluses).

Comparaison de *Min-Fill* vis-à-vis de $H_{\{2,3,4,5\}}$ Par la suite, nous ne retenons que *Min-Fill* parmi les heuristiques de la table 3.3 étant donné qu'il s'agit de l'heuristique de l'état de l'art et celle donnant les meilleurs résultats dans la table 3.3. En plus, nous ne considérons que les méthodes exploitant les redémarrages vu que la tendance reste la même en comparant *BTD-MAC+RST* à *BTD-MAC* selon les différentes décompositions exploitées. En comparant le comportement de *BTD-MAC+RST* selon la décomposition employée, nous constatons que la prise en compte de la connexité des clusters permet d'augmenter le nombre d'instances résolues par rapport à celles résolues grâce à *Min-Fill*. D'ailleurs, dans ce benchmark, seulement 10% des décompositions calculées par *Min-Fill* ne contiennent que des clusters connexes. Mais, comme expliqué dans la partie 3.2.2, la présence des clusters non connexes a un impact négatif sur l'efficacité de la résolution. Ainsi, la construction de clusters connexes permet d'augmenter le nombre d'instances résolues. En particulier, si *Min-Fill* permet de résoudre 1 507 instances, H_2 en résout 1 536. Nous avons aussi examiné la connexité des clusters des autres décompositions. En ce qui concerne H_3-*TD*, pour 91% des instances tous les clusters sont connexes. Quant à H_4-*TD* et H_5-*TD*, pour respectivement 86% et 84% des instances la décomposition ne contient que des clusters connexes. Nous remarquons que pour ces heuristiques, un bon pourcentage des décompositions n'inclut que des clusters connexes. En effet, pour calculer le prochain cluster E_i, ces heuristiques effectuent un parcours en largeur à partir de V_i en rajoutant à E_i tous les sommets du niveau parcouru. Ce faisant, elles augmentent leur chance de construire des clusters connexes. L'heuristique H_3 montre aussi son intérêt pratique. Plus précisément, 1 558 instances sont résolues grâce à son exploitation. Ces résultats prouvent que la prise en compte de la topologie du graphe en essayant de détecter ses parties indépendantes lors du calcul des décompositions permet d'améliorer l'efficacité de la résolution. Finalement, H_4 et H_5 visent à limiter la taille des séparateurs produits. Nous observons une augmentation significative du nombre d'instances résolues atteignant respectivement 1 588 et 1 586 instances pour $H_4^{5\%}$ et $H_5^{5\%}$. Ces résultats soulignent l'intérêt de borner la taille des séparateurs déjà évoqué dans la partie 3.2.2. Notons aussi que ces résultats sont cohérents avec ceux de [Jégou et al., 2005], qui insistent sur ce paramètre pour renforcer l'efficacité de la résolution. L'augmentation du nombre d'instances résolues ne semble pas pénaliser les temps cumulés d'exécution. En effet, l'augmentation du nombre d'instances résolues grâce à H_2 (29 instances en plus par rapport à *Min-Fill*) est accompagnée d'une diminution du temps total de résolution qui passe de 33 632 *s* à seulement 22 854 *s* pour H_2. Le temps d'exécution de *BTD-MAC+RST* exploitant H_3 s'élève à 26 418 *s* s'expliquant par le nombre d'instances additionnelles résolues grâce à H_3. Quant à $H_4^{5\%}$ et $H_5^{5\%}$, elles permettent de résoudre le plus grand nombre d'instances en moins de 24 500 *s*. Notons que les temps de décompositions sont assez remarquables du fait que *Min-Fill* requiert 13 895 *s* pour décomposer l'ensemble des instances tandis que les autres décompositions n'ont pas besoin de plus de 550 *s*, voire seulement 15 *s* pour $H_4^{5\%}$. D'ailleurs, si nous retranchons les temps de décomposition des temps cumulés d'exécution, le temps d'exécution avec *Min-Fill* devient 24 719 *s* (8 913 *s* pour décomposer les instances résolues). En revanche, il n'y a pas de changements significatifs des temps d'exécution de *BTD* avec les autres décompositions. Finalement, nous montrons dans la table 3.5 les

Algorithme	$Min\text{-}Fill^{5\%}$		$H_2^{5\%}$		$H_3^{5\%}$	
	#rés	temps	#rés	temps	#rés	temps
BTD-MAC+RST	1 561	32 640	1 578	28 396	1 576	24 993

TABLE 3.6 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-MAC+RST* selon les décompositions après l'application de la stratégie de fusion.

temps d'exécution de *BTD-MAC+RST* avec les décompositions $Min\text{-}Fill^{5\%}$, $H^{5\%}_{\{2,3,4,5\}}$-*TD* pour toutes les instances de I_2. Autrement dit, nous ajoutons 900 *s* par instance non résolue au temps d'exécution cumulé. Nous remarquons alors que le fossé s'élargit entre *Min-Fill* et $H^{5\%}_{\{4,5\}}$ pour le temps d'exécution cumulé total.

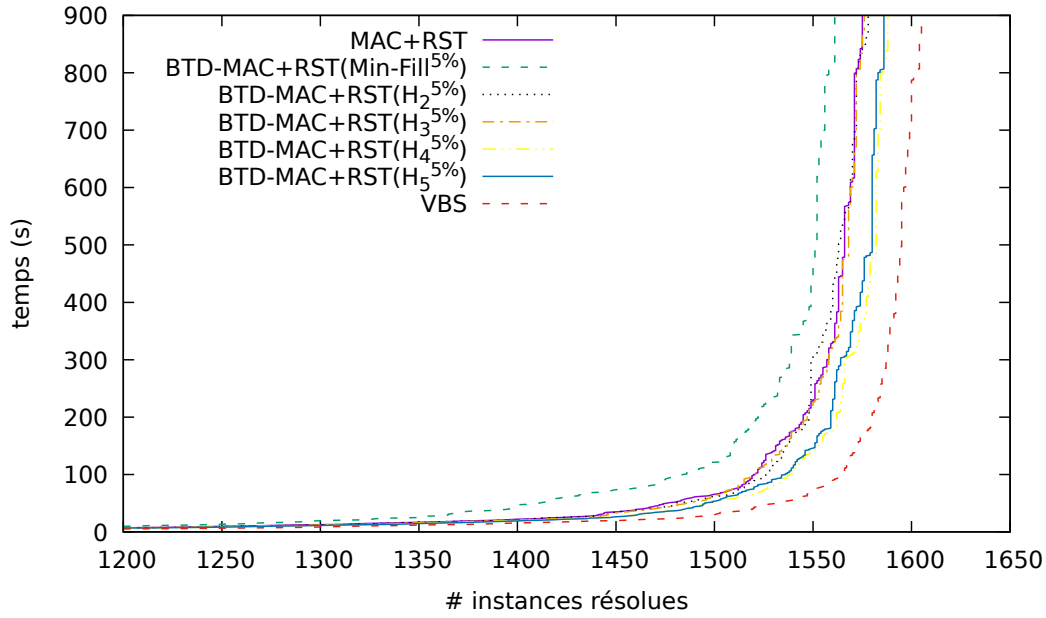

FIGURE 3.12 – Le nombre cumulé d'instances résolues grâce à chaque décomposition pour les 1 859 instances considérées du benchmark I_2.

Comparaison de *Min-Fill* vis-à-vis de $H_{\{2,3,4,5\}}$ pour la même taille de séparateur Dans le but de comparer d'une façon équitable la qualité des différentes décompositions vis-à-vis de l'efficacité de la résolution, nous appliquons la stratégie de fusion [Jégou et al., 2005] pour les décompositions calculées par *Min-Fill*, H_2 et H_3 (ainsi notées $Min\text{-}Fill^{5\%}$, $H_2^{5\%}$ et $H_3^{5\%}$). Nous limitons ainsi la taille des séparateurs à 5% du nombre de variables de l'instance dans la limite d'au moins 4 variables et au plus 50. Ainsi, n'importe quel cluster dont la taille de son séparateur dépasse cette limite sera fusionné avec son père (en mettant en commun des sommets du cluster et de son père pour former un seul cluster). Ce processus est répété jusqu'à ce qu'il n'y ait plus de séparateurs de taille non convenable. Comme prévu, le tableau 3.6 montre que toutes les décompositions permettent désormais de résoudre plus d'instances. Plus précisément, $Min\text{-}Fill^{5\%}$ permet de résoudre 1 561 instances, soit 54 instances de plus que *Min-Fill*. $H_2^{5\%}$ et $H_3^{5\%}$ permettent aussi à leur tour de résoudre plus d'instances avec respectivement 1 578 et 1 576 instances résolues au lieu de 1 536 et 1 558 instances. La tendance reste alors presque la même, avec $H_4^{5\%}$ et $H_5^{5\%}$ surpassant les autres heuristiques de décomposition. Notons

que l'augmentation du nombre d'instances résolues s'accompagne soit d'une amélioration de temps de résolution cumulé, soit d'une légère augmentation de temps justifiée par le nombre additionnel d'instances résolues (cf. tables 3.4 et 3.6).

Comparaison de BTD-MAC+RST avec les décompositions Min-$Fill^{5\%}$ et $H^{5\%}_{\{2,3,4,5\}}$ vis-à-vis de MAC+RST et du VBS Nous nous intéressons également à la comparaison de ces heuristiques à l'égard de MAC+RST et du VBS (pour *Virtual best solver*) qui correspond au meilleur temps de résolution parmi MAC+RST et BTD-MAC+RST utilisant chacune des différentes décompositions. La figure 3.12 montre le nombre cumulé d'instances résolues par BTD-MAC+RST avec chaque décomposition, par MAC+RST ou par le VBS. De son côté, MAC+RST résout 1 575 instances en 25 567 s. Quant au VBS, il résout 1 605 instances en 22 693 s. Il paraît clairement que BTD-MAC+RST exploitant n'importe quel $H^{5\%}_i$ est maintenant beaucoup plus compétitif face à MAC+RST notamment avec $H^{5\%}_4$ et $H^{5\%}_5$ qui permettent de surpasser certainement MAC+RST. Cependant, comme prévu, MAC+RST permet d'obtenir de meilleurs résultats qu'avec BTD exploitant Min-$Fill^{5\%}$. Finalement, les résultats de $H^{5\%}_4$ et $H^{5\%}_5$ semblent très intéressants au vu des résultats du VBS qui ne résout que 17 instances additionnelles.

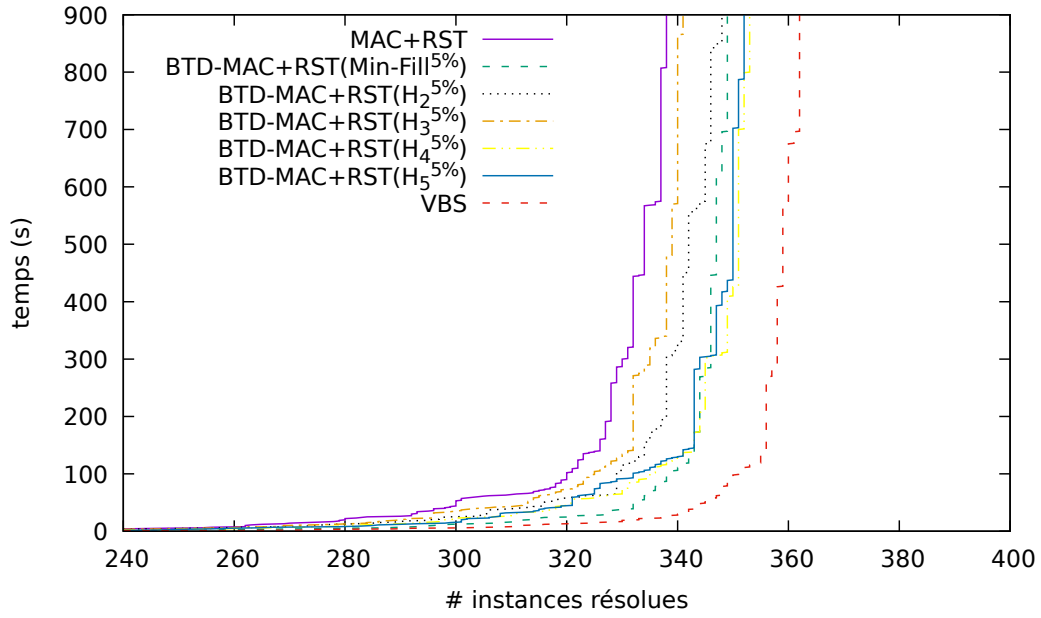

FIGURE 3.13 – Le nombre cumulé d'instances résolues grâce à chaque décomposition uniquement pour les instances ayant une décomposition de w^+ tel que $\frac{n}{w^+} \geq 5$ parmi les instances du benchmark I_2.

Nous nous limitons maintenant aux instances ayant de bonnes propriétés topologiques, c'est-à-dire une largeur w^+ (celle calculée par Min-$Fill$) telle que $\frac{n}{w^+} \geq 5$. Nous disposons alors de 417 instances. Nous constatons particulièrement dans la figure 3.13 que Min-$Fill^{5\%}$ a une meilleure performance sur ces instances grâce à sa bonne approximation de la largeur arborescente. Ainsi, Min-$Fill^{5\%}$ permet de résoudre plus d'instances qu'avec $H^{5\%}_2$, $H^{5\%}_3$ et MAC+RST qui résout le plus petit nombre d'instances. Néanmoins, nous remarquons que BTD-MAC+RST parvient toujours à résoudre plus d'instances avec $H^{5\%}_4$ et $H^{5\%}_5$ qu'avec Min-$Fill^{5\%}$. Notons que, pour ces instances, le pourcentage de décompositions ne contenant que des clusters connexes est de 20% pour Min-$Fill^{5\%}$, 66%

pour $H_3^{5\%}$, 60% pour $H_4^{5\%}$ et de 52% pour $H_5^{5\%}$. Ainsi, sur ces instances, bien que 80% des décompositions de *Min-Fill* contiennent des clusters non connexes, son exploitation permet d'améliorer BTD avec $H_2^{5\%}$ ou $H_3^{5\%}$. Notons d'ailleurs que si la non-connexité d'un cluster E_i correspond au cas de la figure 3.4, elle n'induit aucun problème au niveau de la résolution. En effet, seule la non-connexité de $G[E_i \backslash (E_i \cap E_{p(i)})]$ est susceptible d'avoir une influence sur l'efficacité de la résolution. Celle-ci dépend donc du choix du cluster racine qui peut changer à chaque redémarrage. En tous cas, nous tenons à rappeler que la non-connexité ne dégrade pas systématiquement l'efficacité de la résolution.

Bilan Pour conclure, les expérimentations affirment clairement que les heuristiques de calcul de décomposition visant uniquement à minimiser la largeur de la décomposition ne sont pas les mieux adaptées pour résoudre les instances CSP. Au vu de l'efficacité des algorithmes classiques énumératifs comme MAC+RST, la question de l'intérêt des méthodes basées sur une décomposition se pose légitimement. Cependant, l'introduction du cadre de calcul de décompositions H-TD a permis de proposer d'autres heuristiques de calcul de décompositions dont le but ne se limite pas à s'occuper de la taille des clusters mais se soucie au-delà d'autres critères qui semblent plus intéressants à l'égard de la résolution des instances CSP. C'est ainsi que la connexité des clusters a pu être prise en compte avec H_2-TD, en plus du nombre de fils d'un cluster avec H_3-TD et de la taille des séparateurs avec H_4-TD et H_5-TD. Toutes ces heuristiques ont particulièrement montré que les méthodes basées sur une décomposition peuvent être compétitives vis-à-vis des algorithmes basés sur MAC pour la résolution des instances CSP, voire les surpasser comme avec H_4-TD et H_5-TD.

3.4.3 Efficacité de la résolution pour le problème WCSP

Dans cette sous-section, nous évaluons l'intérêt pratique du cadre de calcul de décompositions H-TD pour la résolution des problèmes d'optimisation sous contraintes WCSP. Mais, au préalable, nous décrivons le protocole expérimental suivi.

3.4.3.1 Protocole expérimental

Nous considérons les algorithmes $HBFS$ et BTD-$HBFS$ décrits respectivement dans la partie 2.4.4.1 et la partie 2.4.4.2. Nous exploitons leurs implémentations fournies dans *Toulbar2* [TOU, 2006]. Ces deux algorithmes sont connus pour leur efficacité incontestable. Leur point fort principal est leur comportement *anytime* leur permettant d'améliorer la borne inférieure et la borne supérieure en permanence. Le paramétrage de $HBFS$, à savoir les valeurs de α_{hbfs}, de β_{hbfs} et de N_{hbfs}, sont identiques à celles utilisées dans [Allouche et al., 2015], c'est-à-dire respectivement 5%, 10% et 10 000.

En ce qui concerne les décompositions, nous considérons *Min-Fill* en tant qu'heuristique de l'état de l'art en utilisant son implémentation fournie dans *Toulbar2*. Une deuxième version de *Min-Fill* est aussi utilisée et serait référencée par *Min-Fill*[4]. Elle se distingue de *Min-Fill* par l'absence de séparateurs de taille supérieure à 4. Elle résulte de l'application de la stratégie de fusion utilisée dans [Jégou et al., 2005] comme dans la partie 3.4.2. Son utilisation avec BTD-$HBFS$ correspond à l'algorithme BTD-$HBFS^{r_4}$ dans [Allouche et al., 2015]. Du côté de H-TD, nous retenons les décompositions H_2, H_3 et H_5. Nous ne rapportons pas les résultats obtenus par H_1 et par *Min-Fill-MG* qui comme pour le problème CSP (cf. la partie 3.4.2) n'ont pas montré d'intérêt vis-à-vis de l'efficacité de la résolution. Nous choisissons de représenter uniquement les résultats de H_5 qui sont très proches de ceux de H_4. Toutefois, nous gardons H_5 vu qu'elle permet de

Algorithme	*Min-Fill*		*Min-Fill*^{4}	
	#rés.	temps	#rés.	temps
BTD-HBFS	1 712	26 291	1 995	91 232

TABLE 3.7 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-HBFS* selon les décompositions de l'état de l'art.

Algorithme	H_2		H_3		H_5^{25}		$H_5^{5\%}$	
	#rés.	temps	#rés.	temps	#rés.	temps	#rés.	temps
BTD-HBFS	1 946	76 018	1 989	57 348	2 006	58 826	2 028	58 043

TABLE 3.8 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-HBFS* selon les décompositions de *H-TD*.

détecter plus de séparateurs que H_4. Elle est déclinée en deux variantes notées H_5^{25} et $H_5^{5\%}$ permettant de limiter respectivement la taille maximale des séparateurs à 25 et à 5% du nombre de variables de l'instance dans la limite d'au moins 4 variables et au plus 50. Les décompositions H_i sont calculées au sein de notre propre bibliothèque et communiquées à *Toulbar2* par l'intermédiaire d'un fichier. Notons qu'à l'heure actuelle, compte tenu de leur efficacité pratique [Allouche et al., 2015], *HBFS* et *BTD-HBFS* avec *Min-Fill*^{4} peuvent être considérés comme étant les références en tant qu'algorithmes de résolution des instances WCSP respectivement sans et avec exploitation de la structure.

En ce qui concerne la configuration de *Toulbar2*, un prétraitement reposant sur la cohérence d'arc est appliqué via *VAC* [Cooper et al., 2008, 2010] en plus de l'application de l'algorithme *MSD* (pour *Min Sum Diffusion*) [Kovalevsky and Koval, 1975; Cooper et al., 2010] avec 1 000 itérations. La cohérence d'arc est ensuite maintenue pendant la résolution grâce à *EDAC* [Givry et al., 2005]. L'heuristique de choix de variables est dom/wdeg associée à l'heuristique du dernier conflit toutes les deux décrites dans la partie 2.2.4.6. Pour les algorithmes comme *BTD*, le cluster racine choisi est le plus grand cluster (pour *Min-Fill* car c'est l'heuristique choisie dans [Allouche et al., 2015]) ou le cluster maximisant le ratio entre le nombre de contraintes du cluster et sa taille (pour les autres décompositions).

Pour chaque instance, chacun des algorithmes de résolution considérés dispose de 20 minutes (ce temps incluant, le cas échéant, le temps requis pour calculer une décomposition) et de 16 Go de mémoire.

L'ensemble d'instances utilisé est le benchmark I_3.

3.4.3.2 Observations et analyse des résultats

Nous comparons le comportement de *BTD-HBFS* en fonction des différentes décompositions exploitées. En ce qui concerne le temps de calcul des décompositions, nous constatons que le calcul des décompositions avec H_i ($i \in \{2, 3, 5\}$) est nettement plus rapide qu'avec *Min-Fill*. Plus précisément, le temps total de calcul des décompositions ne dépasse pas 1 200 s pour les 2 430 instances décomposées par H_i ($i \in \{3, 5\}$) et 4 655 s pour les 2 427 instances décomposées par H_2 tandis que *Min-Fill* requiert 19 044 s pour décomposer 2 415 instances.

Comparaison de *H-TD* à *Min-Fill* et *Min-Fill*^{4} Les tables 3.7 et 3.8 fournissent le nombre d'instances résolues et le temps d'exécution cumulé pour *BTD-HBFS* respectivement avec les décompositions de l'état de l'art et les décompositions de *H-TD*. Tout

d'abord, notons que *Min-Fill* résout le plus petit nombre d'instances en 20 minutes (seulement 1 712 instances). Cela montre que, malgré la difficulté du problème de l'optimisation sous contraintes, les heuristiques de décomposition cherchant uniquement à minimiser la taille des clusters ne sont pas les plus efficaces. Les résultats montrent aussi que les autres paramètres ont plus d'impact, comme la connexité des clusters (H_2), le nombre de fils d'un cluster (H_3) ou la taille des séparateurs ($Min\text{-}Fill^4$ et H_5). Le fait de borner la taille des séparateurs semble jouer un rôle crucial dans l'augmentation de l'efficacité de la résolution. En effet, si $BTD\text{-}HBFS$ avec H_2 et H_3 résout respectivement 1 946 et 1 989 instances, les décompositions dont la taille de séparateurs est bornée permettent de résoudre 1 995 pour $Min\text{-}Fill^4$ et plus de 2 000 pour H_5 (2 006 pour H_5^{25} et 2 028 pour $H_5^{5\%}$). La comparaison de $Min\text{-}Fill^4$ avec H_5 montre cependant que H_5 permet à $BTD\text{-}HBFS$ de résoudre plus d'instances notamment avec $H_5^{5\%}$ qui résout 2 028 instances contre 1 995 instances pour $Min\text{-}Fill^4$. La figure 3.14 compare les temps d'exécution de $BTD\text{-}HBFS$ avec $H_5^{5\%}$ à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$. Elle montre que la plupart des instances sont résolues plus rapidement par $BTD\text{-}HBFS$ avec $H_5^{5\%}$ que par $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$. En

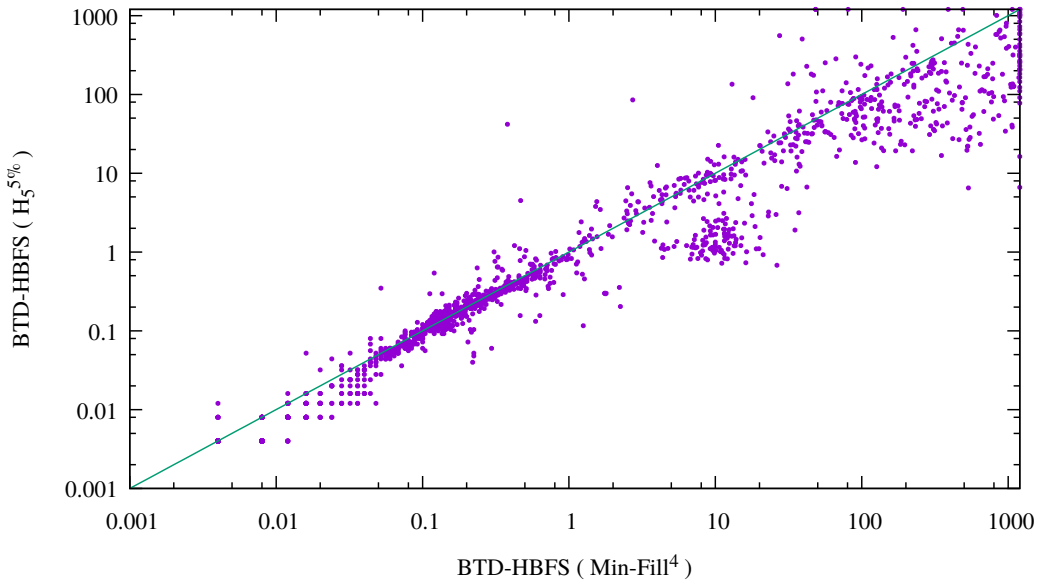

FIGURE 3.14 – Comparaison des temps d'exécution de $BTD\text{-}HBFS$ avec $H_5^{5\%}$ à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ pour les 2 444 instances du benchmark I_3.

outre, $BTD\text{-}HBFS$ associé à $Min\text{-}Fill^4$ requiert 91 232 s en temps d'exécution cumulé contre seulement 58 043 s pour $H_5^{5\%}$. La figure 3.15 montre le nombre cumulé d'instances résolues grâce à chaque décomposition. La figure 3.15 illustre l'analyse réalisée ci-dessus. Elle montre de nouveau clairement d'un côté l'inefficacité de *Min-Fill* par rapport aux autres décompositions pour la résolution des instances WCSP et de l'autre côté l'intérêt des heuristiques bornant la taille des séparateurs notamment $H_5^{5\%}$. Notons enfin que ces résultats sont cohérents avec ceux obtenus pour le problème de décision CSP dans [Jégou et al., 2015a] et ceux obtenus dans la partie 3.4.2.

Comparaison de $BTD\text{-}HBFS(H_5^{5\%})$ à $HBFS$ Nous comparons maintenant $BTD\text{-}HBFS$ à $HBFS$. La figure 3.15 permet de situer $HBFS$ par rapport à $BTD\text{-}HBFS$ selon la décomposition employée. Nous retenons par la suite la décomposition $H_5^{5\%}$ qui permet

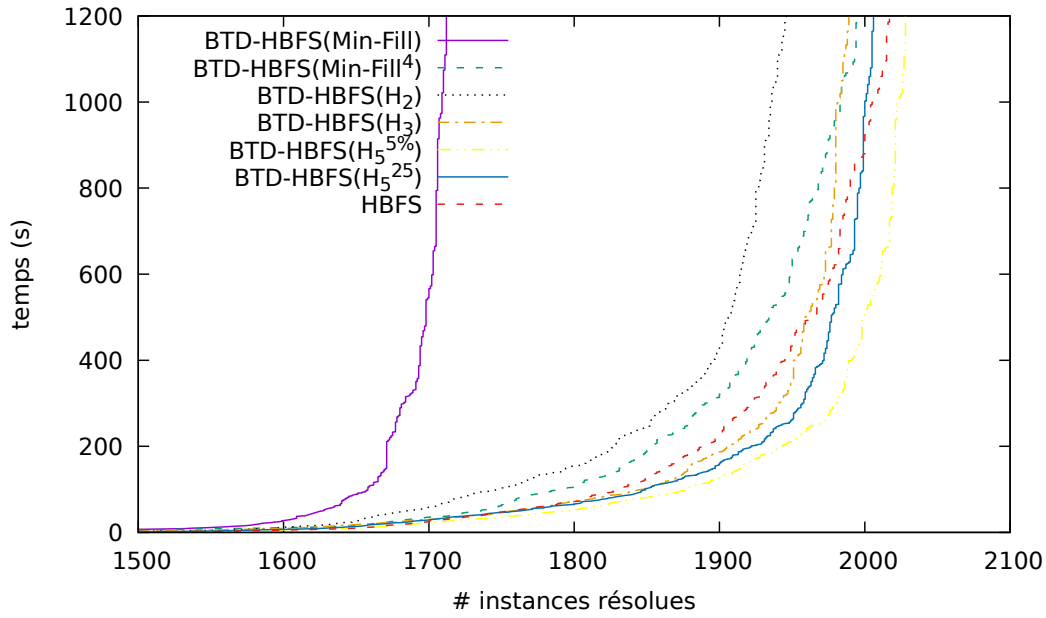

FIGURE 3.15 – Le nombre cumulé d'instances résolues grâce à chaque décomposition et par $HBFS$ pour les 2 444 instances considérées du benchmark I_3.

d'obtenir les meilleurs résultats avec $BTD\text{-}HBFS$ par rapport aux autres décompositions. $BTD\text{-}HBFS$ résout plus d'instances que $HBFS$, à savoir 2 028 instances contre 2 017 instances pour $HBFS$. En plus, $BTD\text{-}HBFS$ ne nécessite que 58 043 s en temps cumulé d'exécution contre 84 657 s pour $HBFS$. Cela peut s'expliquer essentiellement par les enregistrements faits par $BTD\text{-}HBFS$ au niveau des séparateurs, à savoir une borne inférieure et une borne supérieure de l'optimum d'un sous-problème. De plus, $BTD\text{-}HBFS$ détecte les indépendances entre les sous-problèmes, ce qui n'est pas le cas de $HBFS$. La figure 3.16 présente une comparaison des temps d'exécution de $BTD\text{-}HBFS$ à $HBFS$. $BTD\text{-}HBFS$ associée à $H_5^{5\%}$ prouve son intérêt pratique par rapport à $HBFS$. En effet, la plupart des instances qui sont mieux résolues par $HBFS$ ont un temps de résolution comparable à celui de $BTD\text{-}HBFS$. Cependant, nous pouvons remarquer qu'il y a de nombreuses instances qui sont résolues bien plus rapidement avec $BTD\text{-}HBFS$ qu'avec $HBFS$. Lorsque nous nous limitons aux instances non triviales pour $HBFS$ (i.e. résolues par $HBFS$ en plus de 10 secondes ou non résolues), les résultats sont encore plus favorables pour $BTD\text{-}HBFS$. Ce résultat est montré par la figure 3.17 qui compare les temps d'exécution cumulés de $BTD\text{-}HBFS$ et $HBFS$ sur ces instances. Finalement, en examinant parmi ces instances, les instances résolues par les deux méthodes nous obtenons 350 instances. Ces instances sont résolues par $HBFS$ en 74 019 s et en 46 192 s par $BTD\text{-}HBFS$.

Nous comparons aussi les bornes supérieures et les bornes inférieures rapportées par $HBFS$ et $BTD\text{-}HBFS$ dans le cas de dépassement du temps limite. Parmi les 377 instances qui ne sont pas résolues ni par $HBFS$, ni par $BTD\text{-}HBFS$, pour 151 instances, la borne inférieure calculée par $BTD\text{-}HBFS$ est strictement supérieure à celle de $HBFS$ contre 109 pour $HBFS$. En outre, pour 233 instances, la borne supérieure calculée par $BTD\text{-}HBFS$ est strictement inférieure à celle de $HBFS$ contre seulement 73 pour $HBFS$. Cela montre que même lorsque l'instance n'est pas résolue, $BTD\text{-}HBFS$ est capable de donner des approximations de meilleure qualité que $HBFS$. Pour avoir une idée plus

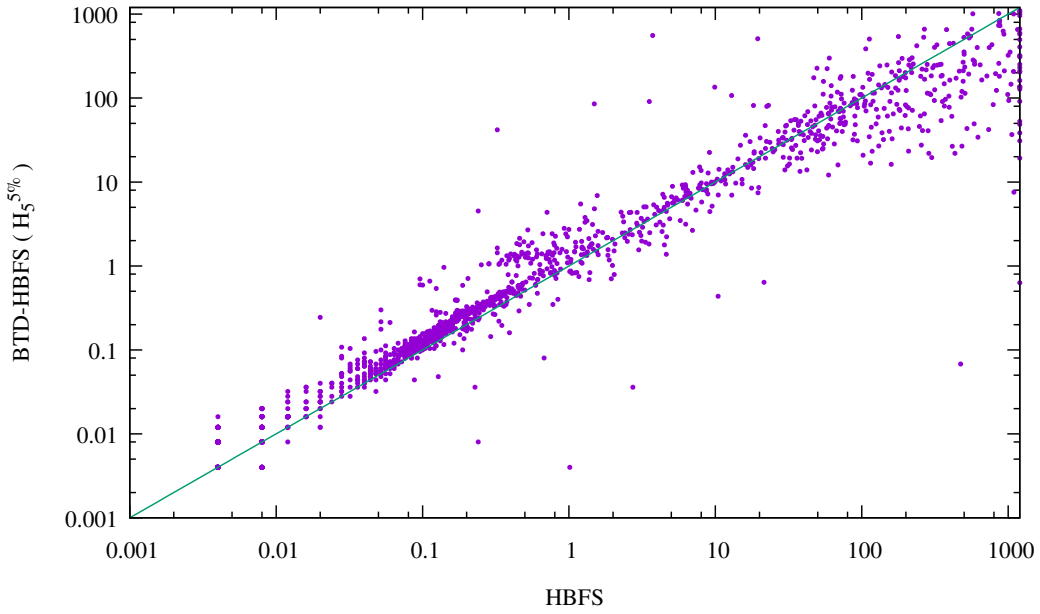

FIGURE 3.16 – Comparaison des temps d'exécution de $BTD\text{-}HBFS$ avec $H_5^{5\%}$ à $HBFS$ pour les 2 444 instances du benchmark I_3.

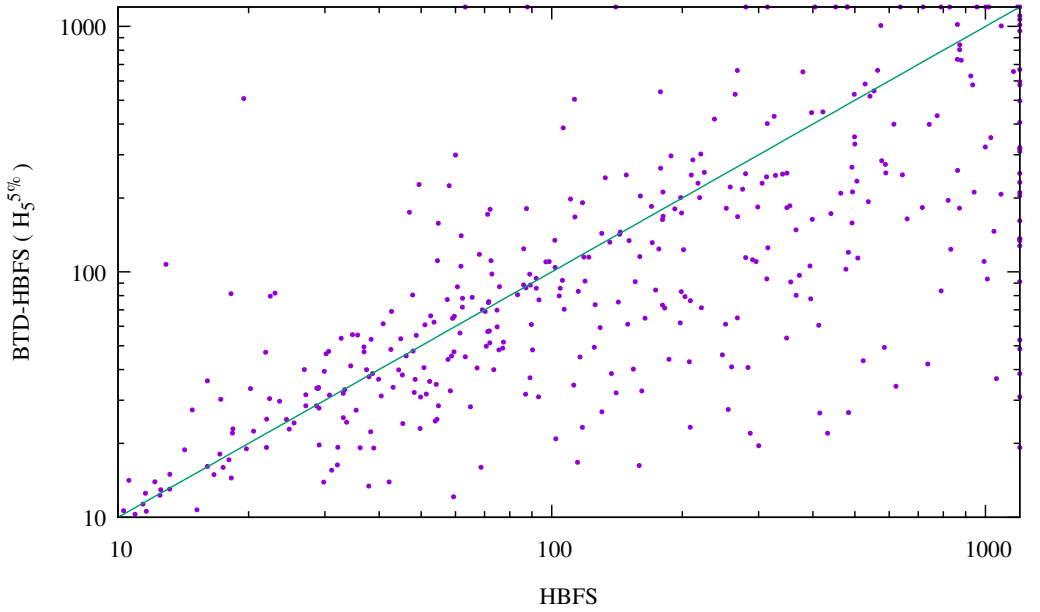

FIGURE 3.17 – Comparaison des temps d'exécution de $BTD\text{-}HBFS$ avec $H_5^{5\%}$ à $HBFS$ sur les instances non résolues par $HBFS$ en moins de 10 secondes du benchmark I_3.

précise sur la progression de la recherche menée par les deux méthodes, nous comparons les écarts obtenus entre la borne supérieure et la borne inférieure pour chaque instance. La figure 3.18 montre une comparaison de l'écart entre la borne supérieure et la borne inférieure pour $HBFS$ et $BTD\text{-}HBFS$. Dans la figure 3.19, nous limitons l'écart maximum à 10^5. Nous constatons à travers les deux figures que $BTD\text{-}HBFS$ semblent avoir des écarts plus petits que ceux obtenus avec $HBFS$. $BTD\text{-}HBFS$ est alors dans de nombreux

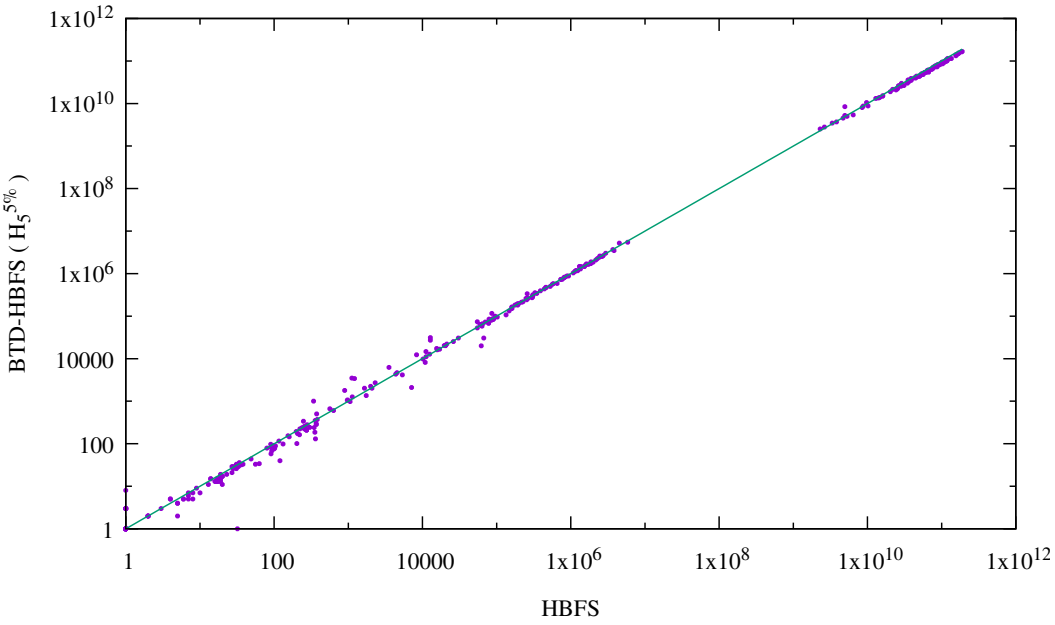

FIGURE 3.18 – Comparaison de l'écart entre la borne supérieure et la borne inférieure pour les 377 instances non résolues.

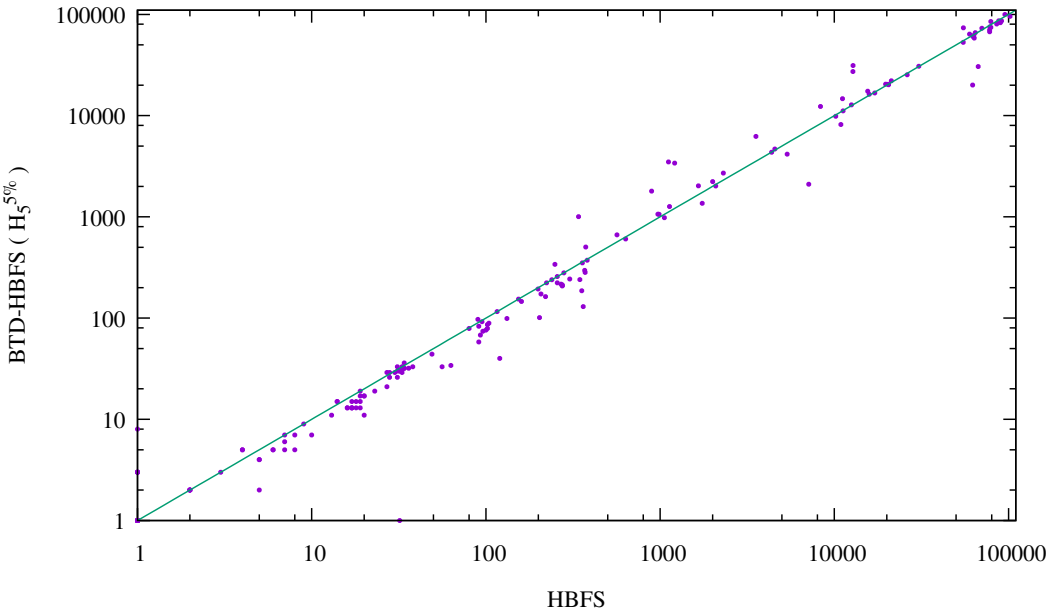

FIGURE 3.19 – Comparaison de l'écart entre la borne supérieure et la borne inférieure pour les 377 instances non résolues (écart limité à 10^5).

cas le plus proche de trouver l'optimum.

Bilan En conclusion, l'évaluation de *H-TD* dans le cadre de la résolution du problème *WCSP* rejoint les conclusions réalisées lors de son évaluation pour la résolution des instances *CSP*. En effet, l'utilisation de *Min-Fill* conjointement avec *BTD-HBFS* montre que les heuristiques dont l'optique est de minimiser la taille des clusters semble avoir un

intérêt moindre à l'égard de la résolution des instances WCSP. En revanche, l'exploitation des décompositions H_2, H_3 et H_5 améliore la performance de $BTD\text{-}HBFS$ et souligne son efficacité notamment avec H_5, en bornant la taille des séparateurs de la décomposition. Ces décompositions permettent également de mettre en valeur $BTD\text{-}HBFS$ vis-à-vis de $HBFS$, la référence en tant que méthode n'exploitant pas la structure du problème. Plus précisément, l'exploitation de $BTD\text{-}HBFS$ avec $H_5^{5\%}$ permet de résoudre plus d'instances que $HBFS$ tout en réalisant un temps cumulé nettement inférieur à celui de $HBFS$.

3.5 Conclusion

Les méthodes de résolution des instances (W)CSP basées sur la décomposition arborescente ont démontré leur intérêt théorique car elles permettent de garantir une borne de complexité en temps en $O(exp(w))$ tout en ayant une complexité en espace en $O(exp(s))$. Lorsque w est borné par une constante, ces méthodes assurent ainsi un temps de résolution polynomial. Le fait de calculer une décomposition optimale ayant une largeur $w^+ = w$ est un problème NP-difficile. C'est ainsi que les efforts se sont focalisés sur l'élaboration de méthodes heuristiques capables d'estimer au mieux cette largeur w dans un temps raisonnable.

Ces heuristiques, dont $Min\text{-}Fill$ fait office de référence dans la communauté CP et bien au-delà [Darwiche, 2009; Koller and Friedman, 2009; Dechter, 2013], sont, pour la plupart d'entre elles, basées sur la triangulation. Toutefois, la triangulation souffre de multiples défauts. D'une part, le fait qu'elle soit réalisée uniquement en se basant sur des critères ne tenant pas compte de la topologie du graphe, comme le degré des sommets dans le cas de $Min\text{-}Fill$, permet de renforcer *l'effet boule de neige*. Cela induit l'ajout d'arêtes non nécessaires du point de vue de la triangulation. D'autre part, l'ajout excessif d'arêtes provoque une augmentation du temps de triangulation et ainsi celui de la décomposition. Plus important, il pourrait être à l'origine de la sur-estimation du w^+ de la décomposition obtenue au final. Au-delà, même si la théorie affirme l'intérêt de minimiser w^+, la pratique ne semble pas en faveur de cette démarche, du moins pour la résolution des instances CSP et WCSP. La conception des méthodes de calcul des décompositions n'étant pas à la base faite en vue de résoudre des instances (W)CSP, elle s'oriente essentiellement vers la minimisation de ce paramètre. En revanche, elle néglige d'autres paramètres comme la connexité des clusters, la taille des séparateurs et la liberté de l'heuristique de choix de variables lors de la résolution.

Nous avons alors proposé un nouveau cadre algorithmique de calcul de décompositions, appelé $H\text{-}TD$. Il permet de calculer des décompositions arborescentes en traversant le graphe, en se basant sur des propriétés liées aux séparateurs et leurs composantes connexes associées. Il n'a pas alors forcément recours à une triangulation ce qui a permis d'améliorer considérablement le temps de calcul des décompositions en pratique. En plus, ce cadre est paramétrable afin de prendre en compte divers critères et répondre aux multiples besoins concernant les caractéristiques des décompositions à calculer. Ces critères peuvent effectivement être liés, par exemple, à la taille des clusters ou à la taille des séparateurs. Grâce à $H\text{-}TD$, nous avons alors introduit deux nouvelles heuristiques, $H_1\text{-}TD$ et $Min\text{-}Fill\text{-}MG$, dans le but de minimiser w^+. $H_1\text{-}TD$, contrairement à $Min\text{-}Fill$, n'est pas basée sur la triangulation ce qui lui permet d'être 13 fois plus rapide que $Min\text{-}Fill$ sur le benchmark I_2 (cf. la partie 3.4.1). Toutefois, $H_1\text{-}TD$ est capable de produire des décompositions de bonne qualité par rapport à $Min\text{-}Fill$. De son côté, $Min\text{-}Fill\text{-}MG$ emploie la triangulation de façon mieux guidée afin de prendre en compte la topologie du graphe. Bien que son temps de calcul soit élevé, $Min\text{-}Fill\text{-}MG$ permet d'obtenir des décompositions d'une

qualité significativement meilleure que $Min\text{-}Fill$. Malgré leur intérêt au niveau de la minimisation de w^+, ces heuristiques ne semblent pas être efficaces vis-à-vis de la résolution. C'est pourquoi nous avons proposé les heuristiques H_2-TD, H_3-TD, H_4-TD et H_5-TD qui tiennent compte respectivement de la connexité des clusters, du nombre de fils et de la taille des séparateurs. Qu'il s'agisse de résoudre des instances CSP ou WCSP, ces heuristiques ont permis d'améliorer l'efficacité des méthodes de résolution structurelles notamment en exploitant les décompositions bornant la taille des séparateurs comme H_5-TD. En outre, les algorithmes basés sur une décomposition sont désormais plus compétitifs face à ceux n'exploitant pas sur une décomposition comme MAC et $HBFS$ (cf. les parties 3.4.2 et 3.4.3).

Finalement, les décompositions employées dans ce chapitre sont calculées en amont de la résolution uniquement sur la base de propriétés structurelles du graphe. Ces décompositions seront utilisées tout au long de la résolution. Nous remarquons ainsi l'importance de cette première et seule décomposition calculée qui va imposer partiellement un ordre de choix de variables durant toute la résolution. Toutefois, cet ordre peut ne pas être en totale adéquation avec la nature de l'instance à résoudre. L'ordre de choix de variables étant particulièrement déterminant pour permettre une résolution efficace, il est donc primordial d'avoir plus de liberté quant au choix de la prochaine variable à instancier que celle offerte à travers l'ordre d'origine. Dans le prochain chapitre, nous proposons ainsi un cadre algorithmique permettant de relâcher progressivement les restrictions imposées par la décomposition. Ce faisant, l'algorithme vise à s'adapter aux spécificités de l'instance à résoudre en combinant l'*exploitation de la décomposition* et la *liberté d'instanciation de variables*.