Vers une exploitation dynamique de la décomposition pour les CSPs pondérés

Philippe Jégou **Hanan Kanso** **Cyril Terrioux**
Aix Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France
{philippe.jegou, hanan.kanso, cyril.terrioux}@lsis.org

Résumé

Pour résoudre les problèmes de satisfaction de contraintes pondérés, les méthodes basées sur une décomposition arborescente constituent une approche intéressante selon la nature des instances considérées. Souvent, les décompositions exploitées visent à réduire la taille maximale des clusters, connue comme étant la largeur de la décomposition. En effet, l'intérêt de ce paramètre est lié à son importance par rapport à la complexité théorique de telles méthodes. À ce niveau, Min-Fill constitue l'heuristique de référence pour le calcul de décompositions. Cependant, son intérêt pratique pour la résolution de problèmes demeure limité au vu de ses multiples défauts, notamment au niveau de la restriction de la liberté de l'heuristique de choix de variables.

Ainsi, nous proposons, dans un premier temps, d'exploiter de nouvelles décompositions pour le problème d'optimisation sous contraintes. Le but de ces décompositions est de capturer des critères permettant d'augmenter l'efficacité de la résolution. Dans un second temps, nous proposons d'exploiter ces décompositions plus dynamiquement dans le sens où la résolution d'un sous-problème ne se baserait sur la décomposition que lorsque cela semble utile. Les expérimentations réalisées montrent l'intérêt pratique des nouvelles décompositions ainsi que l'apport de leur exploitation dynamique.

Abstract

When solving weighted constraint satisfaction problems, methods based on a tree-decomposition constitute an interesting approach depending on the nature of the considered instances. The exploited decompositions often aim to reduce the maximal size of the clusters, which is known as the width of the decomposition. Indeed, the interest of this parameter is related to its importance with respect to the theoretical complexity of these methods. However, its practical interest for the solving of instances remains limited if we consider its multiple drawbacks, notably due to the restriction imposed on the freedom of the variable ordering heuristic.

So, we first propose to exploit new decompositions for solving the constraint optimization problem. These decompositions aim to take into account criteria allowing to increase the solving efficiency. Secondly, we propose to use these decompositions in a more dynamic manner in the sense that the solving of a sub-problem would be based on the decomposition only when it seems useful. The performed experiments show the practical interest of these new decompositions and the benefit of their dynamic exploitation.

1 Préliminaires

Le problème de satisfaction de contraintes pondéré (Weighted Constraint Satisfaction Problem (WCSP)) offre un cadre général permettant de modéliser et de résoudre efficacement toute sorte de problèmes issus du monde réel comme par exemple, les problèmes d'allocation de fréquence [4] ou certains problèmes de bio-informatique [22]. Notre objectif, dans ce papier, est d'améliorer l'efficacité des algorithmes résolvant ce problème grâce à des approches structurelles. Pour commencer, nous rappelons la définition d'une instance WCSP :

Définition 1 *Une* instance WCSP *est définie par un triplet* (X, D, W) *avec :*

- *un ensemble* $X = \{x_1, ..., x_n\}$ *de* n variables*,*
- *un ensemble* $D = \{D_{x_1}, ..., D_{x_n}\}$ *de* domaines *finis de* valeurs*, à raison d'un domaine par variable,*
- *un ensemble* W *de* fonctions de coût *de taille* e*. Une fonction de coût portant sur les variables* $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ *est une fonction qui associe à chaque tuple de* $D_{x_{i_1}} \times D_{x_{i_2}} \times \ldots \times D_{x_{i_k}}$ *un entier compris entre 0 et* k *avec* k *le coût maximum associé à un tuple complètement interdit (c'est-à-dire qui exprime une contrainte dure).*

Afin de simplifier le propos, dans la suite de la présentation, nous ne considérerons, que le cas des instances binaires, c'est-à dire des instances dont chaque fonction de coût porte sur au plus deux variables. Toutefois, ce travail peut s'étendre sans problème au cas des instances non binaires. Dans la partie expérimentale, nous exploiterons d'ailleurs indifféremment des instances binaires et des instances non binaires. Par la suite, nous noterons w_{ij} la fonction de coût binaire portant sur les variables x_i et x_j, w_i la fonction de coût unaire portant sur la variable x_i et $w_\emptyset$, la fonction de coût d'arité nulle qui correspond à un coût que doivent payer toutes les affectations.

Le coût d'une affectation complète $(v_1, \ldots, v_n) \in D_{x_1} \times \ldots \times D_{x_n}$ est définie par $w_\emptyset + \sum_{x_i \in X} w_i(v_i) + \sum_{w_{ij} \in W} w_{ij}(v_i, v_j)$. Étant donnée une instance, le problème de satisfaction de contraintes pondéré consiste à trouver le coût minimum associé à une affectation complète. Ce problème d'optimisation est bien connu pour être NP-difficile.

La résolution des instances s'effectue généralement par le biais d'algorithmes de type *séparation et évaluation* (Branch and Bound). Ces algorithmes exploitent traditionnellement deux bornes notées, souvent *clb* et *cub*, qui représentent respectivement un minorant et un majorant du coût optimum. Ils procèdent en étendant progressivement une affectation tant que le minorant courant *clb* est inférieur au majorant courant *cub*. Ce faisant, l'obtention d'une affectation complète permet de mettre à jour le majorant courant avec la valeur de la dite affectation. Ce majorant peut être initialisé avec la valeur k ou grâce à une méthode incomplète. L'efficacité d'une telle approche dépend grandement de la qualité du minorant utilisé à chaque nœud de la recherche. Sur ce point, de nombreux progrès ont été accomplis, ces dernières années, notamment grâce à la définition de différentes formes de cohérences locales [6, 9, 8, 5].

La plupart des travaux effectués dans le cadre du problème WCSP reposent sur des algorithmes de type séparation et évaluation basés sur un parcours en profondeur de l'espace de recherche. Très récemment, une approche hybridant un parcours le meilleur d'abord et un parcours en profondeur a été proposée dans [1]. Elle possède notamment l'avantage de pouvoir fournir, à tout instant, un minorant global ainsi qu'un majorant du coût optimum. Son efficacité pratique est telle qu'elle surclasse, en général, les approches existantes.

Une autre alternative consiste à exploiter la structure de l'instance. Cette structure peut être représentée par un graphe, appelé *graphe de contraintes*, dont les sommets correspondent aux variables et dont les arêtes lient deux sommets dont les variables correspondantes participent ensemble à une fonction de coût binaire de W. Pour exploiter cette structure, certaines méthodes [16, 23, 11, 1] reposent sur la notion de *décomposition arborescente de graphes* [20] pour identifier des sous-problèmes indépendants.

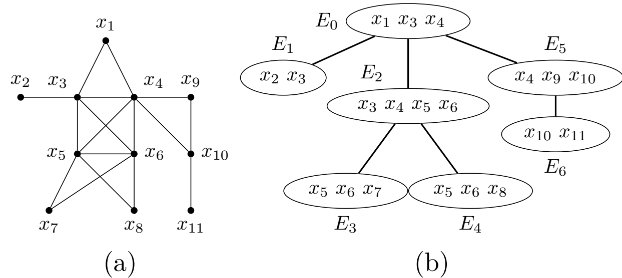

FIGURE 1 – Un graphe (a) et une de ses décompositions arborescentes optimales (b).

Définition 2 *Une* décomposition arborescente *d'un graphe $G = (X, C)$ est un couple (E, T) où $T = (I, F)$ est un arbre (I est un ensemble de nœuds et F un ensemble d'arêtes) et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé cluster) E_i est un nœud de T et vérifie : (i) $\cup_{i \in I} E_i = X$, (ii) pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et (iii) pour tout $i, j, k \in I$, si k est sur le chemin entre i et j dans T, alors $E_i \cap E_j \subseteq E_k$. La largeur d'une décomposition arborescente est égale à $max_{i \in I}|E_i| - 1$. La largeur arborescente dite* tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

La figure 1 présente un graphe et une de ses décompositions arborescentes dont la largeur est 3.

L'utilisation d'une décomposition arborescente permet alors de décomposer le problème original en plusieurs sous-problèmes, d'exploiter des bornes locales de meilleure qualité et d'éviter certaines redondances dans le calcul grâce à l'enregistrement de certaines informations. D'un point de vue théorique, l'intérêt d'une telle approche réside dans sa complexité en temps qui est généralement de l'ordre de $O(n.e.d^{w^+ + 1})$ (avec w^+ la largeur de la décomposition arborescente exploitée) tandis que les méthodes classiques ont une complexité en $O(n.e.d^n)$. D'un point de vue pratique, l'exploitation simultanée des décompositions arborescentes et des cohérences locales a conduit à définir des méthodes ayant une efficacité remarquable, dépassant celles des méthodes classiques [11, 1]. Cette efficacité dépend, en partie, des décompositions employées.

Le plan de cet article est le suivant. Dans la section suivante, nous abordons les méthodes de calcul de ces décompositions. Puis, dans la section 3, nous décrivons comment exploiter dynamiquement une dé-

composition dans le cadre de la résolution d'instances WCSP. Enfin, dans la section 4, nous évaluons l'intérêt pratique de certaines décompositions ainsi que l'apport des décompositions dynamiques avant de conclure dans la section 5.

2 Calcul de décompositions arborescentes

Calculer une décomposition optimale (c'est-à-dire de largeur minimum) est un problème NP-difficile [2]. Aussi, le calcul d'une décomposition arborescente est souvent réalisé par le biais de méthodes heuristiques. Dans ce contexte, la méthode Min-Fill [21] fait office de référence. Grâce à elle, les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances WCSP [1]. Pour autant, leur potentiel n'a pas été pleinement exploité notamment du fait de la qualité des décompositions calculées. En effet, Min-Fill, par exemple, a été conçue pour calculer une décomposition de largeur la plus petite possible, et rien ne garantit que la décomposition produite soit adaptée du point de vue de la résolution d'instances WCSP. D'ailleurs, très récemment, il a été montré, dans [15, 12], que les décompositions produites par Min-Fill n'étaient pas dépourvues de défauts du point de vue de la résolution d'instances du problème de décision CSP et que l'emploi de décompositions spécifiques conduisaient souvent à résoudre plus efficacement les instances.

Dans [12], un cadre générique, appelé *H-TD-WT* (pour *Heuristic Tree-Decomposition Without Triangulation*), a été proposé afin de calculer des décompositions arborescentes adaptées selon des critères donnés. Nous rappelons maintenant ce cadre. Étant donné un graphe $G = (X, C)$ à décomposer, la première étape de *H-TD-WT* (ligne 1 dans l'algorithme 1) calcule un premier cluster, noté E_0, à l'aide d'une heuristique. X' qui représente l'ensemble des sommets déjà considérés est initialisé à E_0 (ligne 2). On notera par $X_1, X_2, \dots X_k$ les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0 [1]. Chacun de ces ensembles X_i est inséré dans une file d'attente F (ligne 4). Pour chaque élément X_i retiré de F (ligne 6), V_i note l'ensemble des sommets de X' qui sont adjacents à au moins un sommet de X_i (ligne 7). V_i constitue un séparateur dans le graphe G puisque la suppression de V_i dans G déconnecte G (X_i étant déconnecté du reste de G). Nous considérons alors le sous-graphe de G induit par V_i et X_i, c'est-à-dire $G[V_i \cup X_i]$. L'étape suivante (ligne 8) peut être paramétrée à l'aide d'une heuristique. Elle recherche un sous-ensemble de sommets $X''_i \subseteq X_i$ tel que $X''_i \cup V_i$

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

Algorithme 1 : *H-TD-WT*

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

1 Choix d'un premier cluster E_0 dans G
2 $X' \leftarrow E_0$
3 Soient $X_1, \dots X_k$ les composantes connexes de $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \dots X_k\}$
5 **tant que** $F \neq \emptyset$ **faire** `/* calcul d'un nouveau cluster` E_i `*/`
6 Enlever X_i de F
7 Soit $V_i \subseteq X'$ le voisinage de X_i dans G
8 Déterminer un sous-ensemble $X''_i \subseteq X_i$ tel qu'il existe au moins un sommet $v \in V_i$ tel que $N(v, X_i) \subseteq X''_i$
9 $E_i \leftarrow X''_i \cup V_i$
10 $X' \leftarrow X' \cup X''_i$
11 Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

sera un nouveau cluster E_i de la décomposition. 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 $N(v, X_i) \subseteq X''_i$. Nous définissons alors un nouveau cluster $E_i = X''_i \cup V_i$ (ligne 9). Puis, nous rajoutons à X' les sommets de X''_i (ligne 10), avant de calculer les composantes connexes du sous-graphe issu de la suppression des sommets de E_i dans $G[X_i]$, composantes qui sont alors rajoutées à la file F (ligne 11). Ce processus est répété jusqu'à ce que la file F soit vide.

Au final, *H-TD-WT* calcule une décomposition arborescente du graphe $G = (X, C)$ sans triangulation en temps polynomial (à savoir en $O(n(n + e))$). Bien sûr, comme pour Min-Fill, aucune garantie n'est offerte quant à l'optimalité de la largeur obtenue. Par contre, dans le cadre de la résolution d'instances (W)CSP, un choix judicieux de paramètres peut conduire à produire des décompositions plus adaptées que celles produites par Min-Fill. Parmi les paramètres envisageables, on peut citer, par exemple, la connexité des clusters, la taille des clusters produits ou encore la taille des séparateurs (c'est-à-dire des intersections entre clusters). Nous considérons, par la suite, les trois déclinaisons suivantes proposées dans le cadre de la résolution du problème de décision CSP et qui visent à rendre la résolution plus efficace :

— H_2 [14] qui garantit que tous les clusters produits sont connexes,
— H_3 [12] qui identifie des parties indépendantes dans le graphe et les sépare aussitôt que possible en effectuant un parcours en largeur à partir des sommets de V_i,
— H_5 [13] qui vise à produire des décompositions dont les séparateurs entre clusters sont de taille au plus S.

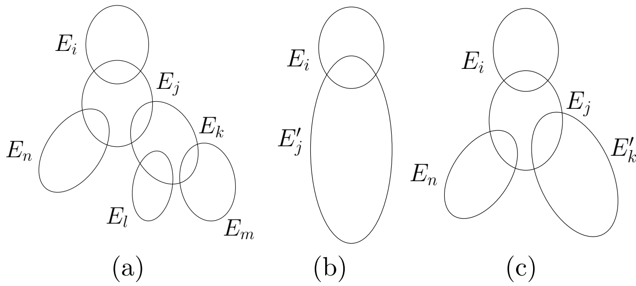

FIGURE 2 – Ensemble de clusters de la décomposition : initial (a) lorsque E_j est fusionné avec sa descendance (cluster E'_j) (b) lorsque E_j est exploité (c).

Une fois une décomposition arborescente calculée, elle peut être exploitée de façon statique comme dans [23, 11, 1] ou de manière dynamique comme présenté dans la section suivante.

3 Exploitation dynamique de la décomposition

Nous visons à exploiter la décomposition arborescente plus intelligemment en évitant de l'utiliser systématiquement. En effet, selon la nature de l'instance à résoudre, sa résolution grâce à des méthodes n'ayant pas recours à une décomposition peut être très efficace. Cela s'explique par les améliorations qu'ont connues ces méthodes notamment en s'associant aux techniques dites *adaptatives*. Ces techniques permettent de faire des choix plus judicieux pendant la résolution en tenant compte de l'état courant de la résolution mais aussi des états précédents. Ce faisant, elles sont capables de guider la recherche vers les parties les plus problématiques. Grâce à elles, par exemple, une heuristique de choix de variables basée sur les conflits va se concentrer sur les variables jugées les plus pertinentes et les plus difficiles. Nous citons à l'appui l'intérêt de guider la recherche grâce à la pondération des contraintes et l'enregistrement des conflits [3], ou grâce à la notion d'activité des variables [18] ou leur impact [19]. Ces techniques profitent aussi de l'absence de contraintes engendrées par l'usage d'une décomposition. En effet, l'exploitation d'une décomposition induit un ordre partiel sur les variables qui impose des restrictions à l'heuristique de choix de variables et limite potentiellement l'intérêt des techniques adaptatives. D'ailleurs, ces techniques peuvent parfois s'avérer inutiles si l'ordre induit par la décomposition est total aboutissant à un choix de variables statique avec tous les inconvénients que cela pourrait engendrer par rapport à un choix de variables dynamique. En plus de leur association avec les techniques adaptatives, ces algorithmes ont connu une amélioration remarquable au niveau de leur stratégie de recherche. Plus précisément, dans [1], la combinaison de deux stratégies de recherche, à savoir le parcours en profondeur (DFS) et le parcours le meilleur d'abord (BFS) a amélioré significativement la résolution des problèmes d'optimisation sous contraintes. Naturellement, si les limitations imposées par la décomposition au niveau de l'heuristique de choix de variables sont très restrictives, l'efficacité pratique de cette hybridation peut s'en retrouver détériorée. En revanche, l'utilisation d'une décomposition peut être parfois très bénéfique pour la résolution. C'est essentiellement le cas des instances qui présentent des propriétés structurelles intéressantes. En particulier, l'enregistrement des informations au niveau des séparateurs entre les clusters permet d'élaguer des parties de l'espace de recherche et de rendre la résolution plus efficace. Dans le cadre de l'optimisation, ces enregistrements permettent de mémoriser, pour chaque sous-problème considéré, les meilleures bornes inférieure et supérieure calculées, ou mieux encore, son optimum. Afin de concilier ces deux points de vue, nous proposons d'exploiter la décomposition dynamiquement d'une façon plus flexible et plus appropriée au vu du contexte courant de la résolution. Cela permet de s'adapter progressivement à la nature de l'instance à résoudre. La forme de dynamicité que nous proposons dans ce cadre est une utilisation de la décomposition vis-à-vis d'un sous-problème justifiée par une stagnation de la recherche sans décomposition. En d'autres termes, la décomposition n'est utilisée que lorsque son utilisation semble pertinente.

L'algorithme BTD+DYN (voir l'algorithme 2) se base sur l'algorithme BTD-DFS proposée dans [11] et repris dans [1]. Les modifications apportées représentent une adaptation de l'algorithme afin de prendre en compte l'exploitation dynamique de la décomposition. Les deux algorithmes se basent sur une décomposition arborescente calculée en amont de la résolution et qui est enracinée en un cluster racine E_r. En ce qui concerne BTD-DFS, la décomposition induit un ordre partiel sur les variables. Si E_j est le cluster courant, le choix de la prochaine variable s'effectue soit parmi les variables non instanciées du cluster E_j, soit, une fois le cluster E_j entièrement instancié, parmi les variables non instanciées d'un cluster fils de E_j (celui-ci étant choisi de manière heuristique parmi tous les fils de E_j). Comme la décomposition ne change pas pendant la résolution, il est en de même pour l'ordre partiel sur les variables. Cependant, BTD+DYN exploite la décomposition différemment. Plus précisément, la décomposition calculée initialement ne sera exploitée pour un sous-problème donné que si la résolution sans décomposition est jugée inefficace. Prenons l'exemple de la décomposition de la figure 2(a) représentée par l'en-

Algorithme 2 : BTD+DYN ($\mathcal{A}, E_i, V, V_{desc}, clb, cub$)

Entrées : L'affectation courante $\mathcal{A}$, le cluster courant E_i, la borne inférieure clb

Entrées-Sorties : L'ensemble V des variables non instanciées de E_i, l'ensemble V_{desc} des variables non instanciées de $Desc(E_i)$, la borne supérieure courante cub

1 **si** $Fusion(E_i)$ **alors**
2 $\quad V' := V_{desc}$
3 **sinon**
4 $\quad V' := V$
5 **si** $V' \neq \emptyset$ **alors**
6 $\quad x$:=dépiler(V') /* Choisir une variable de V' */
7 $\quad$ Mettre à jour V et V_{desc}
8 $\quad a$:=dépiler(D_x) /* Choisir une valeur */
9 $\quad$ Appliquer la cohérence locale au sous-problème $P_i|\mathcal{A} \cup \{(x = a)\}$
10 $\quad clb' := max(clb, lb(P_i|\mathcal{A} \cup \{(x = a)\}))$
11 $\quad$ **si** $clb' < cub$ **alors**
12 $\quad\quad cub$:=BTD+DYN($\mathcal{A} \cup \{(x = a)\}, E_i, V, V_{desc}, clb', cub$)
13 $\quad$ **si** $max(clb, lb(P_i|\mathcal{A})) < cub$ **alors**
14 $\quad\quad$ Appliquer la cohérence locale au sous-problème $P_i|\mathcal{A} \cup \{(x \neq a)\}$
15 $\quad\quad clb' := \max(clb, lb(P_i|\mathcal{A} \cup \{(x \neq a)\}))$
16 $\quad\quad$ **si** $clb' < cub$ **alors**
17 $\quad\quad\quad cub :=$ BTD+DYN ($\mathcal{A} \cup \{(x \neq a)\}, E_i, V, V_{desc}, clb', cub$)
18 **sinon**
19 $\quad$ **si** $\neg Fusion(E_i)$ **alors**
20 $\quad\quad S := Fils(E_i)$
21 $\quad\quad$ /*Résoudre les fils dont l'optimum n'est pas connu */
$\quad\quad$ **tant que** $S \neq \emptyset$ **et** $lb(P_i|A) < cub$ **faire**
22 $\quad\quad\quad E_j$:=dépiler(S) /* Choisir un cluster fils */
23 $\quad\quad\quad$ **si** $LB_{P_j|\mathcal{A}} < UB_{P_j|\mathcal{A}}$ **alors**
24 $\quad\quad\quad\quad cub' := \min(UB_{P_j|\mathcal{A}}, cub - [lb(P_i|\mathcal{A}) - lb(P_j|\mathcal{A})])$
25 $\quad\quad\quad\quad cub'' :=$ BTD+DYN ($\mathcal{A}, E_j, E_j \backslash (E_i \cap E_j), Desc(E_j) \backslash (E_i \cap E_j), lb(P_j|\mathcal{A}), cub'$)
26 $\quad\quad\quad\quad$ Mettre à jour $LB_{P_f|\mathcal{A}}$ et $UB_{P_f|\mathcal{A}}$ en se basant sur cub''
27 $\quad\quad cub := \min(cub, w^i_\emptyset + \sum_{E_j \in Fils(E_i)} UB_{P_j|\mathcal{A}})$
28 $\quad$ **sinon**
29 $\quad\quad cub := \min(cub, \sum_{E_j \in Desc(E_i)} w^j_\emptyset)$
30 **retourner** cub

semble de clusters $E = \{E_i, E_j, E_k, E_l, E_m, E_n\}$. Soit E_j le cluster courant et $\mathcal{A}$ l'affectation courante sur $E_j \cap E_i$. On s'intéresse au sous-problème $P_j|\mathcal{A}$ enraciné en E_j induit par l'affectation $\mathcal{A}$ du séparateur $E_j \cap E_i$ avec E_i le cluster parent de E_j. La résolution du sous-problème $P_j|\mathcal{A}$ n'utilisera pas forcément la décomposition, soit l'ensemble des clusters $\{E_j, E_k, E_l, E_m, E_n\}$. En effet, au début, la résolution est basée sur le cluster E'_j résultant de la fusion du cluster E_j avec ses descendants E_k, E_l, E_m et E_n. Cette fusion consiste en une mise en commun de toutes les variables de la descendance de E_j (voir la figure 2(b)). Ainsi E'_j contient toutes les variables de la descendance de E_j. En ce qui concerne l'heuristique de choix de variables, elle acquiert une liberté totale quant au choix des variables suivantes sur la descendance de E_j. À ce niveau, deux cas se présentent :

- Le sous-problème $P_j|\mathcal{A}$ est facilement résolu en se basant sur E'_j. Dans ce cas, l'exploitation de la décomposition ne semble pas utile.
- La résolution de $P_j|\mathcal{A}$ est au contraire inefficace. Dans ce cas, l'exploitation de la décomposition semble judicieuse. Le cluster E_j est alors réexploité et le même raisonnement est répété au niveau du cluster E_k qui sera au début considéré fusionné avec sa descendance formant le cluster E'_k comme le montre la figure 2(c).

Quant aux clusters feuilles (c'est-à-dire n'ayant pas de fils, comme E_n), BTD+DYN se comporte comme BTD-DFS. À noter que ce raisonnement est refait pour chaque nouvelle affectation de $E_i \cap E_j$. Ce choix est motivé par le fait qu'une affectation $\mathcal{A}$ du séparateur $E_i \cap E_j$ induit un sous-problème différent de celui induit par une affectation $\mathcal{A}'$. À noter aussi, qu'au niveau du cluster racine E_r, BTD+DYN se comporte comme l'algorithme de base ayant toute la liberté concernant le choix des variables du problème. Finalement, la décomposition initiale est utilisée complètement (tous ses clusters sont exploités) si à tous les niveaux BTD+DYN décide d'exploiter le cluster lui-même plutôt que sa descendance.

L'algorithme BTD+DYN prend en paramètres l'affectation courante $\mathcal{A}$, le cluster courant E_i, l'ensemble V des variables non affectées de E_i, l'ensemble V_{desc} des variables non affectées de la descendance $Desc(E_i)$ de E_i (c'est-à-dire l'ensemble des clusters situés dans le sous-arbre enraciné en E_i, E_i inclus), et les bornes inférieure et supérieure courantes clb et cub. Il vise à calculer l'optimum du problème enraciné en E_i et induit par l'affectation $\mathcal{A}$. Deux cas sont possibles :

- Si la valeur de cub retournée est strictement inférieure à celle de cub en entrée, alors cub est l'optimum correspondant à ce problème.
- Sinon, cub définit une borne inférieure de l'optimum.

L'appel initial est BTD+DYN($\emptyset$, E_r, V_r, $V_{r_{desc}}$, $lb(P_r|\emptyset)$, k) avec $V_{r_{desc}}$ l'ensemble des variables de la descendance de E_r, $lb(P_r|\emptyset)$ la borne inférieure initiale déduite grâce à l'application de la cohérence locale au problème initial et k le coût maximal. Notons que tout au long de l'algorithme, $w^i_\emptyset$ désigne la borne inférieure localisée relative au cluster E_i. Comme BTD-DFS, BTD+DYN suppose que le problème donné en entrée est cohérent selon la cohérence locale utilisée et renvoie son optimum. Les lignes 5 à 17 de BTD+DYN visent à instancier les variables de V' comme le ferait BTD-DFS pour les variables de V. Un couple (variable, valeur), (x, a), est sélectionné selon les heuristiques de choix de variables et de valeurs employées. Comme le

type de branchement utilisé est binaire, ce choix se décline en deux branches $x = a$ (ligne 9) et $x \neq a$ (ligne 14). Pour chaque branche, la cohérence locale est appliquée au sous-problème enraciné en E_i et induit par l'affectation courante permettant d'obtenir une borne inférieure *lb*. Si le maximum *clb′* entre la borne inférieure *lb*, déduite par la cohérence locale, et la borne inférieure courante *clb*, est toujours inférieure à *cub*, la recherche continue ; sinon le sous-arbre correspondant est élagué. Les lignes 20 à 27 de BTD+DYN résolvent les sous-problèmes enracinés en chaque cluster fils E_j de E_i de la même façon que BTD-DFS. Le sous-problème $P_j|\mathcal{A}$ n'est exploré que si son optimum n'est pas déjà connu. D'ailleurs, BTD-DFS et BTD+DYN mémorise pour chaque sous-problème $P_j|\mathcal{A}$ deux valeurs, notées $LB_{P_j|\mathcal{A}}$ et $UB_{P_j|\mathcal{A}}$, repésentant respectivement les meilleures bornes inférieure et supérieure connues pour $P_j|\mathcal{A}$. Si $LB_{P_j|\mathcal{A}} = UB_{P_j|\mathcal{A}}$, l'optimum de P_j est déjà calculé. L'appel récursif correspondant au fils E_j (ligne 25) exploite une borne supérieure initiale non triviale calculée à la ligne 24 comme expliqué dans [11]. Il est suivi par une mise à jour de $LB_{P_j|\mathcal{A}}$ et de $UB_{P_j|\mathcal{A}}$ (ligne 26). L'exploration de tous les clusters fils permet enfin de mettre à jour *cub*. Les similitudes entre BTD-DFS et BTD+DYN rappelées, nous nous intéressons maintenant aux modifications faites. La fonction $Fusion$ représente l'heuristique chargée de faire le choix d'exploiter le cluster E_i ou sa descendance E'_i. L'appel à $Fusion$ avec le cluster E_i en entrée renvoie vrai si et seulement si E'_i est exploité. Sinon, le cluster E_i est exploité et la liberté de choix de variables est restreinte aux variables de V. L'un des deux paramètres V ou V_{desc} est effectivement utilisé selon que l'on exploite E_i ou E'_i. Le choix entre V et V_{desc} est fait dans les lignes 1 à 4 et est retenu dans V'. Ils sont mis à jour convenablement dans la ligne 7. Si $V' = \emptyset$ et que E_i n'est pas fusionné avec sa descendance, BTD+DYN se comporte comme BTD-DFS (lignes 20-27). Si $V' = \emptyset$ et que E_i est fusionné avec sa descendance, BTD+DYN n'a plus de variables à instancier vu que toutes les variables de la descendance de E_i sont déjà affectées. Dans ce cas, BTD+DYN met à jour *cub* (ligne 29) en se référant à la borne inférieure localisée $w^j_\emptyset$ de chaque cluster E_j appartenant à $Desc(E_i)$. Si $V' \neq \emptyset$ (lignes 5-17), BTD+DYN se comporte de la même façon indépendamment du choix de l'exploitation de E_i ou de sa descendance E'_i en essayant d'instancier les variables de V'.

L'algorithme BTD+DYN est paramétrable par l'heuristique $Fusion$ qui se charge de décider de passer de l'exploitation du cluster E'_i à E_i, si E_i est le cluster courant (nous en proposons une dans la partie expérimentale). Un choix pertinent de cette heuristique est, bien sûr, essentiel pour l'amélioration de la résolution.

L'exploitation dynamique de la décomposition induit un changement au niveau des complexités en temps et en espace. Le comportement de BTD+DYN pouvant aller d'un BTD-DFS standard à un classique DFS, il en résulte que la complexité temporelle de BTD+DYN est comprise entre $O(exp(w^+ + 1))$ et $O(exp(n))$ avec w^+ la largeur de la décomposition calculée en amont de la résolution. Toutefois, il est possible de limiter cette complexité en utilisant une heuristique convenable qui se charge d'interdire l'exploitation des clusters de la descendance à des profondeurs faibles, comme dans le cas du cluster racine par exemple. Au niveau de la complexité en espace, elle est exponentielle en fonction de la taille du plus grand séparateur exploité au niveau de la décomposition. Donc, dans le pire des cas, la complexité en espace est la même que celle de BTD-DFS si le plus grand séparateur de la décomposition est utilisé.

4 Expérimentations

Dans cette section, nous évaluons d'une part l'intérêt pratique du cadre de calcul de décompositions *H-TD-WT* et d'autre part l'apport de l'exploitation dynamique des décompositions dans le cadre de la résolution des problèmes d'optimisation sous contraintes. Mais, au préalable, nous décrivons le protocole expérimental suivi.

4.1 Protocole expérimental

Nous considérons les algorithmes HBFS et BTD-HBFS introduits dans [1] et exploitons leurs implémentations fournies dans *Toulbar2*[2]. HBFS consiste en une hybridation de la stratégie de recherche qui combine à la fois les avantages du parcours en profondeur (DFS) et les avantages du parcours le meilleur d'abord (BFS). HBFS est alors capable de donner à tout moment une borne inférieure de l'optimum comme BFS mais aussi une borne supérieure comme DFS. Son intégration avec BTD permet d'exploiter ses caractéristiques au sein de chaque cluster améliorant ainsi le comportement global de ce dernier.

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^{r_4}. Elle se distingue de Min-Fill par l'absence de séparateurs de taille supérieure à 4. En effet, chaque cluster de la décomposition calculée par Min-Fill partageant plus de quatre variables avec son père est fusionné avec celui-ci (son

2. Présent dans le répertoire git à `https://mulcyber.toulouse.inra.fr/projects/toulbar2/`

utilisation avec BTD-HBFS correspond à l'algorithme BTD-HBFS^{r_4} dans [1]). Du côté de H-TD-WT, nous retenons les décompositions H_2 qui garantit d'obtenir des clusters connexes, H_3 qui permet de calculer des clusters ayant plusieurs clusters fils et H_5 pour sa maîtrise de la taille des séparateurs. L'heuristique H_5 est déclinée en deux variantes notées H_5^{25} et $H_5^{5\%}$. H_5^{25} limite la taille maximale des séparateurs à 25. Quant à $H_5^{5\%}$, elle exploite 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. 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 [1], HBFS et BTD-HBFS avec Min-Fill^{r_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.

L'exploitation dynamique de la décomposition repose sur une heuristique ($\mathcal{F}$) qui se charge de décider d'exploiter E_i au lieu d'exploiter toute la descendance E'_i de E_i. L'heuristique $\mathcal{F}$ se base sur le retour d'information fait par BTD-HBFS. En effet, chaque appel de BTD-HBFS sur un sous-problème $P_i|\mathcal{A}$ prend en entrée une borne inférieure *clb* et une borne supérieure *cub*. Si, dans la limite du nombre de backtracks autorisé, BTD-HBFS ne réussit à améliorer aucune de ces deux bornes, $\mathcal{F}$ considère que la résolution de $P_i|\mathcal{A}$ n'avance pas et incrémente un compteur qui lui est propre. Lorsque le compteur relatif à $P_i|\mathcal{A}$ atteint une certaine limite (5 dans nos expérimentations), on exploite par la suite E_i en stoppant l'exploitation de E'_i.

En ce qui concerne la configuration de *Toulbar2*, un pré-traitement reposant sur la cohérence d'arc est appliqué via VAC (pour *Virtual Arc Consistency* [7]) en plus de l'application de l'algorithme MSD (pour *Min Sum Diffusion*) avec 1 000 itérations. La cohérence locale est ensuite maintenue pendant la résolution grâce à EDAC (Existential Directional Arc Consistency [10]). L'heuristique de choix de variables est dom/wdeg [3] associée à l'heuristique du dernier conflit [17]. Pour les algorithmes comme BTD, le cluster racine choisi est le plus grand cluster (pour Min-Fill) ou le cluster maximisant le ratio entre le nombre de contraintes du cluster et sa taille (pour les autres décompositions). Les expérimentations ont été réalisées sur des serveurs lames sous Linux Ubuntu 14.04 dotés chacun de deux processeurs Intel Xeon E5-2609 à 2,4 GHz et de 32 Go de mémoire.

Nous avons utilisé 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 MiniZinc 2012 et 2013 et des instances de la compétition PIC 2011. Nous avons écarté les instances résolues pendant la phase de prétraitement pour obtenir au final un benchmark composé de 2 444 instances. 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 16 Go de mémoire.

4.2 H-TD-WT comparée à Min-Fill

Nous comparons, dans cette partie, le comportement de BTD-HBFS en fonction des différentes décompositions exploitées. Au niveau du calcul de 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 de décompositions ne dépasse pas 1 200 s pour les 2 431 instances décomposées tandis que Min-Fill requiert 19 044 s pour décomposer 2 415 instances. Les tables 1 et 2 fournissent le nombre d'instances résolues et le temps d'exécution cumulé pour BTD-HBFS avec les décompositions de l'état de l'art et les décompositions de H-TD-WT. 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 décompositions cherchant uniquement à minimiser la taille des clusters ne sont pas les plus efficaces. En effet, vu le faible nombre de variables propres (c'est-à-dire de variables appartenant à un cluster et pas à son cluster père) dans chaque cluster (souvent une seule variable) l'heuristique de choix de variables est quasiment inutile et l'ordre d'instanciation de variables est presque statique. Les résultats montrent aussi que les autres paramètres ont plus d'impacts comme la connexité des clusters (H_2), le nombre de fils d'un cluster (H_3) ou la taille des séparateurs (Min-Fill^{r_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-HBFS avec H_2 et H_3 résout respectivement 1 945 et 1 989 instances, les décompositions dont la taille de séparateurs est bornée permettent de résoudre plus de 2 000 instances. La comparaison de Min-Fill^{r_4} avec H_5 montre cependant que H_5 permet à BTD-HBFS de résoudre plus d'instances notamment avec $H_5^{5\%}$ qui résout 2 029 instances contre 2 000 instances pour Min-Fill^{r_4}. En outre, BTD-HBFS associé à Min-Fill^{r_4} requiert 98 861 s en temps d'exécution cumulé contre seulement 58 792 s pour $H_5^{5\%}$. Notons enfin que ces résultats sont cohérents avec ceux obte-

Algorithme	*Min-Fill*		*Min-Fill*^{r_4}	
	#rés.	temps	#rés.	temps
`BTD-HBFS`	1 712	26 291	2 000	98 861
`BTD-HBFS+DYN`	1 905	45 450	2 019	74 159

Table 1 – Nombre d'instances résolues et temps d'exécution en secondes pour BTD-HBFS et BTD-HBFS+DYN 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`	1 945	74 686	1 989	57 254	2 006	56 553	2 029	58 792
`BTD-HBFS+DYN`	2 023	50 445	2 023	56 978	2 039	52 304	2 038	60 674

Table 2 – Nombre d'instances résolues et temps d'exécution en secondes pour BTD-HBFS et BTD-HBFS+DYN selon les décompositions de H-TD-WT.

nus pour le problème de décision CSP dans [12].

4.3 BTD-HBFS contre HBFS

Nous comparons BTD-HBFS à HBFS. Nous retenons, dans cette partie, la décomposition $H_5^{5\%}$ qui permet d'obtenir les meilleurs résultats avec BTD-HBFS par rapport aux autres décompositions. BTD-HBFS résout plus d'instances que HBFS, à savoir 2 029 instances contre 2 017 instances pour HBFS. En plus, BTD-HBFS ne nécessite que 58 792 s en temps cumulé d'exécution contre 84 555 s pour HBFS. Cela peut s'expliquer essentiellement par les enregistrements effectués par BTD-HBFS au niveau des séparateurs, à savoir une borne inférieure et une borne supérieure de l'optimum d'un sous-problème. De plus, BTD-HBFS détecte les indépendances entre les sous-problèmes, ce qui n'est pas le cas de HBFS.

4.4 Évaluation de la dynamicité

Nous évaluons à présent BTD-HBFS+DYN. Les deux tables 1 et 2 montrent, que quelle que soit la décomposition exploitée, BTD-HBFS+DYN résout plus d'instances que BTD-HBFS. L'augmentation du nombre d'instances résolues peut être considérable. C'est notamment le cas pour Min-Fill pour lequel une exploitation dynamique permet de résoudre 1 905 instances contre 1 712 instances dans le cadre d'une exploitation statique. L'utilisation de H_2 et H_3 avec BTD-HBFS+DYN permet de résoudre 2 023 instances contre 1 945 et 1 989 résolues par BTD-HBFS. L'exploitation de Min-Fill^{r_4} et H_5 est aussi améliorée avec BTD-HBFS+DYN par rapport à BTD-HBFS. En particulier, H_5^{25} permet de résoudre 2 039 instances, ce qui constitue le plus grand nombre d'instances résolues parmi toutes les combinaisons d'algorithme de résolution et de décomposition présentées. L'augmentation du nombre d'instances résolues est souvent accompagnée d'une diminution du temps total d'exécution, sauf pour Min-Fill, mais cela s'explique naturellement par le nombre d'instances supplémentaires résolues par BTD-HBFS+DYN par rapport à BTD-HBFS. En outre, nous nous intéressons à la comparaison de H_5^{25} à l'heuristique de l'état de l'art Min-Fill^{r_4} avec BTD-HBFS+DYN et aussi à la comparaison de BTD-HBFS+DYN basé sur H_5^{25} à HBFS. La figure 3(a) présente une comparaison des temps d'exécution de BTD-HBFS+DYN avec H_5^{25} à BTD-HBFS+DYN avec Min-Fill^{r_4} tandis que la figure 3(b) le compare à HBFS. Dans les deux cas, BTD-HBFS+DYN associé à H_5^{25} prouve clairement son intérêt pratique. Pour une comparaison plus équitable des temps cumulés d'exécution, nous nous basons sur le benchmark des instances résolues à la fois par BTD-HBFS+DYN avec Min-Fill^{r4} et par BTD-HBFS+DYN avec H_5^{25}. Il compte 2 013 instances résolues par BTD-HBFS+DYN avec Min-Fill^{r4} en 71 249 s contre seulement 41 372 s par BTD-HBFS+DYN avec H_5^{25}. Nous procédons, de même, pour comparer HBFS et BTD-HBFS+DYN avec H_5^{25}. Nous obtenons alors un total de 2 008 instances résolues en 79 020 s par HBFS contre seulement 43 914 s par BTD-HBFS+DYN. Le couplage de BTD-HBFS+DYN avec H_5^{25} assure notamment que les méthodes basées sur la décomposition peuvent être compétitives vis-à-vis des méthodes n'exploitant pas la décomposition comme HBFS. Tous ces résultats montrent que la dynamicité exploitée au niveau de la décomposition semble permettre à BTD-HBFS+DYN de mieux s'adapter à la nature de l'instance et d'éviter de se baser sur une décomposition lorsqu'une résolution sans décomposition s'avère plus efficace que ce soit globalement ou localement.

Nous focalisons maintenant nos observations sur HBFS et BTD-HBFS+DYN associé à H_5^{25}. Nous écartons à ce stade les instances faciles, voire parfois triviales (c'est-à-dire celles résolues en moins de 10 s

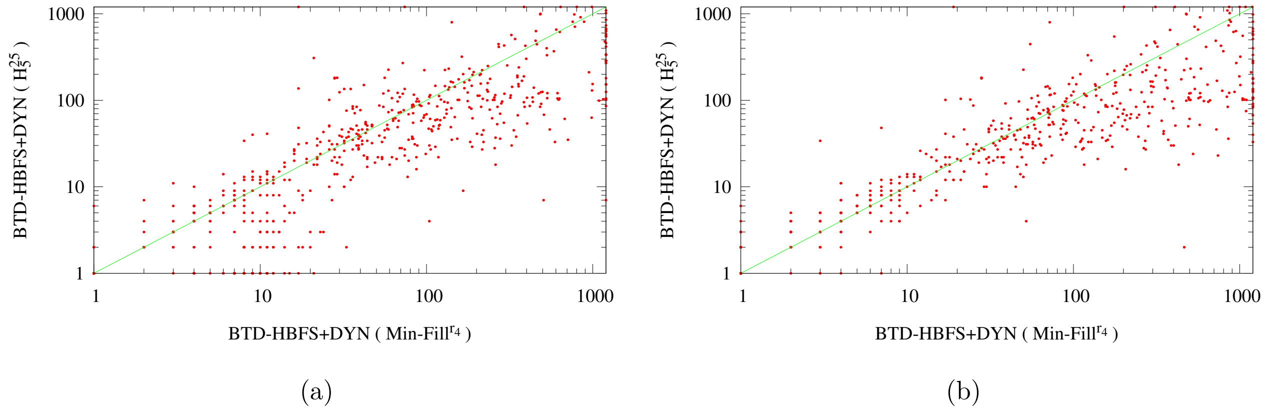

FIGURE 3 – Comparaison des temps d'exécution en secondes de BTD-HBFS+DYN avec H_5^{25} à BTD-HBFS+DYN avec Min-Fill^{r_4} (a) ou à HBFS (b) pour les 2 444 instances.

par HBFS). Notons que ces instances sont aussi facilement résolues par BTD-DYN+HBFS vu qu'au niveau du cluster racine BTD-DYN+HBFS se comporte comme HBFS (lorsqu'on exploite la descendance relative au cluster racine), avec aussi l'avantage de pouvoir distinguer les composantes connexes d'une instance ce qui n'est pas le cas de HBFS. Parmi les 794 instances restantes, pour 279 instances au moins un cluster feuille de la décomposition est exploité. Autrement dit, il existe au moins une branche de la décomposition qui est entièrement exploitée (au sens de l'ensemble de clusters d'origine de la décomposition). Pour 423 instances, la décomposition n'est jamais exploitée. Cela signifie que BTD-DYN+HBFS se comporte comme HBFS. Pour les instances restantes, la décomposition est exploitée jusqu'à une certaine profondeur sans pour autant atteindre un cluster feuille de la décomposition. Donc, en pratique, BTD-HBFS+DYN peut se comporter simplement comme HBFS ou faire des choix d'exploitation des clusters d'origine de la décomposition (au lieu de leur descendance) jusqu'à atteindre le comportement de BTD-HBFS.

Enfin nous comparons les bornes supérieures rapportées par HBFS et BTD-HBFS+DYN dans le cas de dépassement du temps limite. Parmi les 304 instances qui ne sont pas résolues par HBFS, ni par BTD-HBFS+DYN, pour 252 instances, la borne supérieure calculée par BTD-HBFS+DYN est strictement inférieure à celle de HBFS contre seulement 52 instances pour HBFS. Cela montre que même lorsque l'instance n'est pas résolue, BTD-HBFS+DYN est capable de donner des approximations de meilleure qualité que HBFS.

5 Conclusion

Les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances du problème WCSP. Pour autant, leur potentiel n'a pas été pleinement exploité notamment du fait de la qualité des décompositions calculées. En effet, à l'image de ce que fait Min-Fill, l'heuristique de référence, leur calcul vise souvent à minimiser la taille des clusters (afin de minimiser la borne de complexité théorique en temps) au détriment de l'efficacité pratique de la résolution. Aussi, pour y remédier, d'une part, nous avons exploité le cadre de décomposition H-TD-WT que nous avons auparavant introduit pour résoudre des instances du problème de décision CSP. Ce cadre vise à calculer des décompositions beaucoup plus rapidement mais aussi à capturer des propriétés intéressantes du point de vue de la résolution comme la maîtrise de la taille des séparateurs avec l'heuristique H_5. L'utilisation conjointe de H_5 avec BTD-HBFS a prouvé l'intérêt de H-TD-WT. En effet, elle a non seulement permis d'améliorer les performances obtenues par BTD-HBFS associé à Min-Fill mais a aussi conduit à surclasser HBFS. D'autre part, nous avons proposé une extension de BTD-HBFS, à savoir BTD-HBFS+DYN, qui exploite la décomposition d'une façon dynamique. L'idée consiste à n'utiliser la décomposition sur un sous-problème que lorsque la résolution sans décomposition semble difficile. Cela évite de restreindre inutilement la liberté de l'heuristique de choix de variables pour des sous-problèmes « faciles ». En pratique, nous avons montré que BTD-HBFS+DYN améliore BTD-HBFS, quelle que soit la décomposition utilisée, en nombre total d'instances et en temps. Aussi, grâce à cette extension, les algorithmes basés sur une décomposition présentent davantage d'intérêt par rapport à ceux non basés sur une décomposition.

Plusieurs extensions de ce travail semblent pertinentes. Par exemple, le calcul de la décomposition fait en amont de la résolution peut, à son tour, être fait dynamiquement. Cela semble d'autant plus intéressant que la décomposition n'est pas forcément exploitée avec BTD-HBFS+DYN. De plus, cela permettrait aussi de calculer des décompositions mieux adaptées en vue de la résolution que celles calculées uniquement sur la base de critères structurels. Au-delà, d'autres heuristiques d'exploitation dynamique de la décomposition sont envisageables.

Remerciements

Nous tenons à remercier les membres du projet *Toulbar2*, et plus particulièrement Simon de Givry, pour leur disponibilité et leur aide.

Références

[1] D. Allouche, S. de Givry, G. Katsirelos, T. Schiex, and M. Zytnicki. Anytime hybrid best-first search with tree decomposition for weighted csp. In *CP*, pages 12–29, 2015.

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

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

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

[5] M. Cooper, S.de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174 :449–478, 2010.

[6] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2) :199–227, 2004.

[7] M. C. Cooper, S. de Givry, M. Sánchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted csp. In *AAAI*, pages 253–258, 2008.

[8] M. C. Cooper, S. de Givry, and T. Schiex. Optimal Soft Arc Consistency. In *IJCAI*, pages 68–73, 2007.

[9] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency : closer to full arc consistency in weighted CSPs. In *IJCAI*, 2005.

[10] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted csps. In *IJCAI*, pages 84–89, 2005.

[11] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *AAAI*, pages 22–27, 2006.

[12] P. Jégou, H. Kanso, and C. Terrioux. An Algorithmic Framework for Decomposing Constraint Networks. In *ICTAI*, pages 1–8, 2015.

[13] P. Jégou, H. Kanso, and C. Terrioux. Improving Exact Solution Counting for Decomposition Methods. In *ICTAI*, pages 327–334, 2016.

[14] P. Jégou and C. Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *ECAI*, pages 465–470, 2014.

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

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

[17] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. Reasoning from last conflict (s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

[18] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *CP-AI-OR*, pages 228–243, 2012.

[19] P. Refalo. Impact-based search strategies for constraint programming. In *CP*, pages 557–571, 2004.

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

[21] D. J. Rose. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Application*, 32 :597–609, 1970.

[22] M. Sanchez, S. de Givry, and T. Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2) :130–154, 2008.

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