Fusion dynamique de la décomposition dans le cas du problème CSP

Sommaire

4.1 Introduction

Nous avons vu dans le chapitre précédent que le calcul de la décomposition arborescente se fait habituellement en amont de la résolution. Ce calcul se fait essentiellement sur la base de critères structurels comme la taille des clusters, la taille des séparateurs ou la connexité des clusters. Même si certains paramètres sont connus pour jouer un rôle crucial dans l'efficacité de la résolution comme la taille des séparateurs [Jégou et al., 2005, 2015a] que nous avons pu valider expérimentalement dans le chapitre précédent, cela ne garantit pas que la décomposition exploitée pendant la résolution soit la plus adaptée au contexte de la résolution. En effet, une décomposition induit un ordre partiel sur les variables qui sera imposé à la recherche tout au long de la résolution. Or, cet ordre n'est pas nécessairement pertinent du point de vue de la résolution. Notamment, il peut être assez différent d'un ordre choisi par une heuristique de choix de variables ayant une totale liberté, ou ne pas respecter le principe du *first-fail*. En outre, le fait que la décomposition soit calculée avant la résolution, empêche de prendre en compte, lors de son calcul, certains critères qui relèvent de *la sémantique* du problème. Par voie de conséquence, l'ordre de choix de variables qui en découle n'intégrera en aucun cas de tels critères. Parmi les critères sémantiques, nous nous intéressons aux caractéristiques de l'instance à résoudre qui se dévoilent au fur et à mesure de l'avancement de la résolution. Par exemple, pour certaines contraintes nous pourrions effectivement calculer en amont le nombre de tuples autorisés par rapport au nombre de tuples possibles, ce qui pourrait constituer un indicateur de la difficulté de la contrainte. Toutefois, sa difficulté réelle est celle constatée pendant la résolution suite aux affectations des variables. En accumulant des informations correspondant aux états précédents de la résolution et à son état courant, nous construisons le contexte de la résolution. Les techniques permettant d'intégrer la sémantique du problème et l'exploiter par les méthodes de résolution sont des techniques dites *adaptatives*. La principale motivation derrière est de tirer profit du contexte courant de la recherche et aussi de son historique pour permettre à la recherche d'effectuer des choix plus judicieux et plus opportuns. Sur un plan plus général, l'adaptativité permettrait de construire des solveurs plus robustes et plus efficaces. De tels solveurs sont très utiles puisque leur utilisation ne requiert pas de l'expertise. L'adaptativité leur permet d'avoir un comportement relativement proche du meilleur comportement qu'il aurait pu avoir en faisant un choix différent au niveau de son paramétrage. Dans ce contexte, l'adaptativité vise à trouver la meilleure décomposition possible pour la résolution de l'instance considérée. Nous nous focalisons alors sur les techniques adaptatives dans la section suivante et nous expliquons la difficulté rencontrée par BTD et plus généralement par les méthodes relevant de ce type d'approche pour profiter pleinement des techniques adaptatives. Nous proposons ainsi dans la section 4.3 une exploitation dynamique adaptative des décompositions dans le cadre du problème CSP qui permettrait à BTD de se libérer de plus en plus des restrictions imposées par la décomposition. Nous illustrons son intérêt pratique et sa pertinence par une étude expérimentale dans la section 4.4.

4.2 Prise en compte du contexte courant de la résolution

Dans cette partie, dans un premier temps, nous allons revoir quelques méthodes utilisées dans le but de s'adapter au contexte de la résolution. Dans un second temps, nous allons parler des obstacles qui empêchent BTD d'exploiter pleinement leurs avantages.

4.2.1 Stratégies existantes

Les techniques que nous abordons tout d'abord concernent les heuristiques de choix de variables, dont les améliorations qui ont été apportées ont été sensiblement avantageuses aux algorithmes de recherche arborescente. Les heuristiques de choix de variables qui s'avèrent les plus efficaces aujourd'hui sont les heuristiques *adaptatives*. Non seulement elles sont dynamiques, mais elles permettent de s'adapter au contexte de la résolution. En effet, elles permettent de concilier les deux aspects de la résolution : l'aspect rétrospectif et l'aspect prospectif. L'aspect rétrospectif détermine la démarche à effectuer à la rencontre d'une incohérence. Il décide du choix de retour-arrière à faire ainsi que du choix des informations à apprendre si une telle approche est prévue. De son côté, l'aspect prospectif concerne l'assignation de la prochaine variable, soit le choix du couple $(variable, valeur)$ à faire ainsi que la réalisation des changements résultant de ce choix notamment en termes de filtrage de valeurs incohérentes et de détection d'une éventuelle incohérence. Les techniques adaptatives fusionnent ces deux points de vue ; elles permettent d'apprendre des informations sur les états précédents de la recherche et les exploitent, en plus des informations sur son état courant, afin de faire des choix plus judicieux pour la suite de la résolution. Par la suite, nous détaillons quelques-unes parmi les plus connues.

Diriger la recherche par les conflits [Boussemart et al., 2004] Cette heuristique, notée $dom/wdeg$, est la plus utilisée parmi les heuristiques de choix de variables existantes principalement à cause de son efficacité et de la simplicité de sa mise en œuvre. Elle vise à diriger la recherche vers les parties les plus difficiles et les plus problématiques. Elle s'inspire évidemment du principe *first-fail* [Haralick and Elliott, 1980]. Cette heuristique s'avère pertinente essentiellement dans le cas des problèmes où certaines contraintes occupent plus d'importance que d'autres et ainsi où certaines parties du problème semblent plus difficiles que d'autres, voire incohérentes. Par exemple, supposons que deux problèmes sont combinés (sans interaction entre les deux) et que le premier est facile tandis que le deuxième est incohérent. Dans ce cas, les incohérences rencontrées vont se concentrer au niveau des contraintes du sous-problème incohérent tandis que les contraintes du sous-problème facile seront beaucoup moins souvent sujettes à une impasse. Le but de cette heuristique est d'éviter le phénomène du *trashing* pouvant résulter d'une telle configuration. En effet, une heuristique de choix de variables naïve pourrait commencer par affecter les variables du sous-problème facile avant d'affecter celles du sous-problème incohérent et rencontrer à chaque fois une incohérence tardivement. Une heuristique plus « intelligente » constaterait cette différence d'importance entre les contraintes et privilégierait l'affectation des variables impliquées dans les contraintes les plus difficiles. Ce faisant, l'incohérence du problème est facilement prouvé. C'est fondamentalement ce que $dom/wdeg$ cherche à faire. Elle se base sur la notion de pondération de contraintes. En effet, elle affecte un poids $wdeg$ à chaque contrainte du problème et l'initialise à 1. Ce compteur sera incrémenté à chaque fois que le domaine d'une de ses variables devient vide. L'adaptation au contexte de la résolution se fait alors par le biais de la pondération des contraintes suite aux conflits rencontrés durant la recherche. En pratique, pendant la progression de la résolution, les contraintes les plus dures subissent une augmentation remarquable de leur poids associé $wdeg$ par rapport aux poids des autres contraintes. L'aspect apprentissage se renforce alors grâce à l'accumulation et l'enregistrement des conflits rencontrés via le poids $wdeg$. De son côté, l'aspect progression exploite ses enregistrements en choisissant comme variable suivante la variable x_i ayant le plus petit rapport $dom(x_i)/\alpha_{wdeg}(x_i)$ où $dom(x_i)$ est la taille du domaine courant de x_i et $\alpha_{wdeg}(x_i) = \sum_{x_i \in S(c_i):|Fut(c_i)|>1} wdeg(c_i)$ avec $Fut(c_i)$ l'ensemble de variables de $S(c_i)$ non encore assignées. Il est à noter qu'au début

dom/wdeg est identique à *dom/deg* vu que le poids de chaque contrainte est initialisé à 1. En comparant la démarche de cette heuristique et celle des autres heuristiques dynamiques, nous constatons, qu'à un instant donné, l'heuristique *dom/wdeg* dispose des informations sur les états précédents de la recherche ainsi que sur son état courant. Cependant, les heuristiques dynamiques traditionnelles exploitent uniquement des informations correspondant à son état courant comme la taille courante du domaine d'une variable. C'est ainsi que les heuristiques adaptatives sont capables de faire de choix plus judicieux et plus appropriés permettant une résolution plus efficace. Cette heuristique a également un intérêt majeur lorsque les redémarrages sont exploités. Elle permet, lors d'un redémarrage, de choisir d'abord les variables jugées comme les plus pertinentes et ainsi de diversifier les choix selon la connaissance qu'il a pu acquérir jusqu'à cet instant. L'heuristique *dom/wdeg* a montré son efficacité sur un large spectre de problèmes réels, académiques et aléatoires.

Diriger la recherche selon l'impact des variables [Refalo, 2004] L'impact d'une variable est une mesure qui indique l'importance d'une variable dans la réduction de la taille de l'espace de recherche. La taille de l'espace de recherche $\mathcal{E}$ est définie comme étant le produit de la taille courante des domaines des variables. Sachant que l'affectation d'une variable $x_i \leftarrow v_i$ réduit la taille des domaines des autres variables (par propagation), son impact est alors défini par $I(x_i \leftarrow v_i) = 1 - \frac{\mathcal{E}_{après}}{\mathcal{E}_{avant}}$ où $\mathcal{E}_{avant}$ et $\mathcal{E}_{après}$ désignent respectivement la taille de l'espace de recherche avant et après la réalisation de l'affectation. L'impact d'une affectation est ainsi plus élevé lorsque la réduction de l'espace de recherche est plus importante. L'impact d'une variable est déduit de l'impact des valeurs de son domaine et est calculé en fonction des impacts obtenus pendant les étapes précédentes de la recherche. L'heuristique de choix de variables basée sur la notion d'impact, notée IBS (pour *Impact-Based Search*), est dite adaptative ; elle choisit comme variable suivante la variable ayant le plus grand impact sur l'espace de recherche qui est une notion calculée en tenant compte de l'impact de cette variable pendant les étapes précédentes de la recherche. En privilégiant d'abord l'affectation des variables ayant l'impact le plus grand, nous dirigeons la recherche vers les parties les plus contraintes du problème. Si au début de la recherche la valeur de l'impact de chaque variable n'est pas encore suffisamment fiable, la progression de la recherche augmente de plus en plus l'exactitude de cette information. Lorsque les redémarrages sont exploitées conjointement, l'algorithme de recherche est capable de modifier les décisions réalisées vers la racine de l'arbre de recherche en se basant sur les dernières mises à jour des impacts des variables. Cette heuristique peut ainsi contribuer à diminuer la taille de l'arbre de recherche. L'exploitation des impacts des variables a permis de résoudre efficacement certains problèmes comme le problème de *complétion des carrés latins*, le problème du *sac à dos* et le problème du *carré magique* [Refalo, 2004].

Diriger la recherche selon l'activité des variables [Michel and Hentenryck, 2012] L'activité d'une variable est une mesure qui reflète le nombre de fois où cette variable est filtrée par propagation. L'heuristique basée sur la notion d'activité, notée ABS (pour *Activity-Based Search*), vise à exploiter les algorithmes de filtrage qui impliquent l'utilisation de mécanismes sophistiqués et à en tirer des informations pertinentes pour mieux guider la résolution. Elle associe à chaque variable un compteur qui sera mis à jour à chaque nœud de l'arbre de recherche. Selon cette heuristique, l'application de l'algorithme de filtrage suite à une décision, partitionne les variables en deux parties :

- les variables $X_f \subseteq X$ dont le domaine est réduit,

- les variables $X \backslash X_f$ dont le domaine reste inchangé.

Soit γ un réel tel que $0 \leq \gamma \leq 1$ qui permet de faire vieillir une valeur. L'activité d'une variable x_i, notée $A(x_i)$, est alors mise à jour selon les deux règles suivantes :

- $\forall x_i \in X$ tel que $|D_{x_i}| > 1$, $A(x_i) = A(x_i) \times \gamma$
- $\forall x_i \in X_f$, $A(x_i) = A(x_i) + 1$

C'est ainsi que toutes les variables non encore affectées et dont le domaine contient au moins deux valeurs subissent un vieillissement de la valeur de leur activité afin de donner de moins en moins d'importance aux conséquences des anciennes affectations. Au contraire, les activités des variables de X_f sont toutes incrémentées. Cette heuristique tend à guider la recherche en privilégiant d'abord l'instanciation de la variable ayant l'activité la plus élevée. L'intuition découle du principe *first-fail* en admettant que les variables filtrées le plus fréquemment semblent les plus difficiles à instancier. L'heuristique est clairement adaptative parce qu'elle maintient, via les activités des variables, des informations sur les états précédents de la résolution et décide du choix de la variable suivante à assigner sur la base de ces informations. L'heuristique ABS, comme $dom/wdeg$, peut être mise en œuvre sans difficulté particulière et sans être coûteuse en temps contrairement à IBS qui dépend de la taille des domaines des variables du problème. L'avantage de ABS par rapport à $dom/wdeg$ est qu'elle est orientée vers les variables et non pas vers les contraintes, dans le sens où $dom/wdeg$ donne le même poids à toutes les variables d'une contrainte lors de sa violation même si seulement un sous-ensemble de ses variables est éventuellement impliqué dans l'échec. L'heuristique ABS semble être la plus robuste, face à IBS et $dom/wdeg$, sur une sélection de benchmarks dont les instances du problème du *sac à dos* et celles du problème du *carré magique* [Michel and Hentenryck, 2012].

Adaptativité pour les redémarrages [Biere, 2008] Les techniques adaptatives peuvent être également utilisées pour mieux gérer les redémarrages. Redémarrer fréquemment la recherche a permis d'améliorer l'efficacité de la résolution de certaines instances. Cependant, le redémarrage très fréquent peut aussi être contre-productif dans d'autres cas. Ce problème a été abordé dans les solveurs SAT. Dans [Biere, 2008], Biere compte sur les techniques adaptatives pour déterminer la fréquence des redémarrages. Il introduit la notion d'*agilité* qui indique le taux des affectations récemment modifiées (affectation à 0 qui passe à 1 ou vice versa). Plus précisément, les affectations qui sont au centre d'intérêt sont les affectations forcées et non pas celles relevant d'une décision ou d'un choix. Une agilité élevée permet de déduire que les redémarrages doivent être plus espacés dans le temps, voire être suspendus à partir d'un certain seuil. Cependant, une agilité basse favorise des redémarrages plus fréquents. L'agilité est une mesure globale initialisée à 0 et mise à jour à chaque fois que l'affectation d'une variable est forcée. À l'instar de la notion d'activité d'une variable, l'agilité subit un vieillissement qui vise à donner plus de poids aux informations récoltées récemment. Les expérimentations ont montré une hausse considérable du nombre d'instances résolues pour les instances SAT dites *crafted*, notamment pour des instances incohérentes.

Adaptativité dans les solveurs CDCL (pour *Conflict Driven Clause Learning*) [Eén and Sörensson, 2003] Les solveurs SAT modernes explorent l'espace de recherche

en affectant les variables et en construisant l'arbre de recherche correspondant. Conjointement, ils utilisent l'apprentissage de clauses afin de limiter la redondance dans l'exploration de l'arbre de recherche. L'adaptativité est très présente dans ces solveurs et a fortement contribué à leur essor. Elle est particulièrement constatée à travers les heuristiques de choix de variables qui sont guidées par les conflits. Elle est aussi mise en relief par les retours en arrière non chronologiques qui utilisent les informations des clauses apprises pour déterminer la profondeur à laquelle retourner dans l'arbre de recherche en cas d'échec. En outre, ils exploitent les techniques de redémarrage en conservant à chaque relance certaines informations apprises. Ces techniques jouent d'ailleurs un rôle primordial dans l'orientation de la recherche aux sous-espaces les plus prometteurs.

4.2.2 BTD : obstacles et motivations

Nous avons rappelé dans la partie précédente quelques techniques adaptatives parmi les plus connues. Les heuristiques de choix de variables et les techniques de redémarrage adaptatives amassent des informations tout au long de la résolution afin de conserver l'historique de celle-ci. La variable suivante à instancier ou la fréquence des redémarrages sont ainsi choisies en se basant sur l'état courant et les états précédents de la recherche. L'apport des techniques adaptatives est tellement impressionnant que leur exploitation par BTD demeure incontournable. Or, vu l'emploi d'une décomposition, bénéficier pleinement de certaines techniques adaptatives semble impossible. En particulier, l'intérêt des heuristiques de choix de variables adaptatives se trouve plus ou moins limité selon la décomposition utilisée. Dans cette partie, nous nous focalisons sur les heuristiques de choix de variables et nous nous intéressons aux limitations qui y sont imposées par l'utilisation de la décomposition arborescente.

4.2.2.1 Ordre de choix de variables sous BTD

Pendant la résolution, BTD exploite une décomposition calculée en amont de celle-ci. Une décomposition donnée de largeur w^+ impose un ordre partiel (ou même total parfois) pour le choix de variables afin de garantir une complexité temporelle en $O(exp(w^+))$.

Pour préciser l'ordre de choix de variables Λ exploité, nous utilisons la notation suivante :

- $\Lambda = [x_1, x_2, \ldots, x_p]$ est un ordre total pour le choix de variables sur p variables qui signifie que la variable x_1 apparaît nécessairement avant x_2 qui apparaît à son tour avant x_3 et ainsi de suite (une séquence de variables). Autrement dit, $x_1 \preceq_X x_2 \preceq_X \cdots \preceq_X x_p$.

- $\Lambda = \{x_1, x_2, \ldots, x_p\}$ est un ordre partiel pour le choix de variables sur p variables qui signifie que les variables $x_1, x_2, \ldots, x_p$ peuvent apparaître dans n'importe quel ordre (un ensemble de variables).

- Soit Λ_1 un ordre sur p_1 variables, Λ_2 un ordre sur p_2 variables, $\ldots$, et Λ_q un ordre sur p_q variables tel que $\Lambda_i, i \in \{1, \ldots, q\}$ peut être un ordre partiel ou total et $\Lambda_i \cap \Lambda_j = \emptyset$ si $i \neq j$. $\Lambda = [\Lambda_1, \Lambda_2, \ldots, \Lambda_q]$ est un ordre sur $p_1 + p_2 + \cdots + p_q$ variables qui signifie que les variables de Λ_1 doivent forcément apparaître avant toutes celles de Λ_2 qui doivent forcément apparaître avant toutes celles de Λ_3 et ainsi de suite.

- Soit Λ_1 un ordre sur p_1 variables, Λ_2 un ordre sur p_2 variables, $\ldots$, et Λ_q un ordre sur p_q variables tel que $\Lambda_i, i \in \{1, \ldots, q\}$ peut être un ordre partiel ou total et $\Lambda_i \cap \Lambda_j = \emptyset$ si $i \neq j$. $\Lambda = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_q\}$ est un ordre sur $p_1 + p_2 + \cdots + p_q$ variables qui signifie

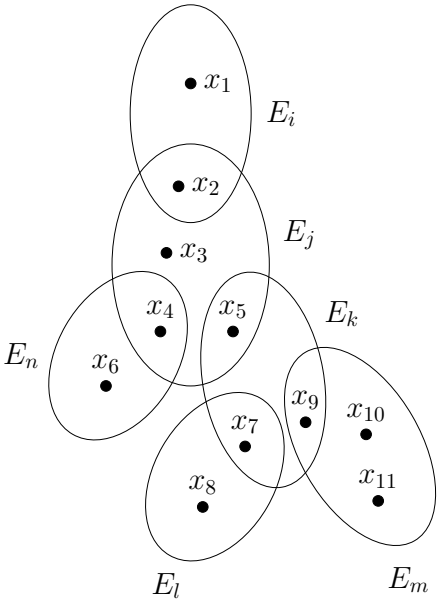

FIGURE 4.1 – La répartition des variables dans les clusters d'une décomposition.

que quel que soit Λ_i et Λ_j de Λ, les variables de Λ_i peuvent apparaître avant toutes celles de Λ_j et vice versa.

Il est à noter que si $\Lambda = \{x_i\}$ ou que $\Lambda = [x_i]$, Λ peut être noté tout simplement x_i. Par exemple, si $\Lambda = [x_1, \{x_2, x_3\}]$ alors x_1 doit nécessairement apparaître avant x_2 et x_3 tandis que x_2 et x_3 peuvent apparaître dans n'importe quel ordre. Les deux ordres totaux possibles sont alors $[x_1, x_2, x_3]$ ou $[x_1, x_3, x_2]$. Aussi, si $\Lambda = [x_1, \{x_2, \{x_3, x_4\}\}]$ alors les ordres totaux possibles sont : $[x_1, x_2, x_3, x_4]$, $[x_1, x_2, x_4, x_3]$, $[x_1, x_3, x_4, x_2]$ ou $[x_1, x_4, x_3, x_2]$.

Les algorithmes de résolution n'exploitant pas une décomposition bénéficient d'une liberté totale quant au choix de la variable suivante à instancier. Au contraire, les méthodes structurelles comme BTD subissent une restriction de la liberté de l'heuristique de choix de variables (cf. la partie 2.2.5.1). L'ordre de choix de variables induit par la décomposition (E, T) découle de l'ordre de visite des clusters qui à son tour est partiellement imposé par la décomposition. En effet, cet ordre est compatible avec la décomposition s'il peut être produit par un parcours en profondeur de l'arbre correspondant T à partir du cluster racine E_r. Étant donné un ordre sur les clusters $<$, un ordre de choix de variables compatible est alors exploité par BTD. Plus précisément, $\forall x \in E_i, \forall y \in E_j$, avec $E_i < E_j$, x doit forcément apparaître avant y dans Λ, c'est-à-dire $\Lambda = [\ldots, x, \ldots, y, \ldots]$ ou $x \preceq_X y$. Dans le but de respecter un ordre de visite des clusters compatible avec un parcours en profondeur de l'arbre T, la liberté de l'heuristique de choix de variables se limite à :

- Choisir librement la variable suivante parmi les variables propres (variables appartenant à un cluster mais pas à son parent) du cluster courant E_i. Toutefois, à chaque fois que le cluster E_i est revisité l'ordre d'instanciation de ses variables propres peut changer.

- Choisir librement un prochain cluster parmi les clusters fils de E_i une fois le cluster E_i entièrement instancié. De même, l'ordre de visite des clusters fils de E_i peut changer à chaque fois que E_i est entièrement instancié.

La figure 4.1 montre la répartition des variables dans les clusters d'une décomposition. Il s'agit d'une décomposition formée de 6 clusters $E = \{E_i, E_j, \ldots, E_n\}$ d'une instance

ayant 11 variables. Supposons que le cluster racine $E_r = E_i$. L'ordre partiel de choix de variables imposé par la décomposition est :
$\Lambda = [\{x_1, x_2\}, \{x_3, x_4, x_5\}, \{x_6, [\{x_7, x_9\}, \{x_8, \{x_{10}, x_{11}\}\}]\}]$. En effet, le cluster E_i est le premier à être instancié, c'est-à-dire les variables x_1 et x_2, selon l'ordre choisi par l'heuristique de choix de variables, suivi par le cluster E_j incluant les variables propres x_3, x_4 et x_5. Puis, un choix entre les deux clusters fils E_k et E_n est réalisé. Lorsque le cluster E_k est choisi et les variables x_7 et x_9 instanciées, de nouveau, un choix est fait entre les clusters E_l et E_m.

Dans [Jégou and Terrioux, 2014a], BTD acquiert un niveau additionnel de liberté (cf. la partie 2.2.5.1). En effet, lors d'un redémarrage, BTD a l'opportunité de choisir un nouveau cluster racine. Le fait de changer le cluster racine résulte en un changement de l'ordre partiel induit par la décomposition employée. Supposons que le nouveau cluster racine de la décomposition dans la figure 4.1 est $E_r = E_j$. Le nouvel ordre partiel induit par la décomposition est alors : $\Lambda = [\{x_2, x_3, x_4, x_5\}, \{x_1, x_6, [\{x_7, x_9\}, \{x_8, \{x_{10}, x_{11}\}\}]\}]$. Le fait de changer le cluster racine et ainsi l'ordre imposé par la décomposition permet d'exploiter potentiellement un ordre plus pertinent à l'égard de la résolution. En particulier, le choix du nouveau cluster racine peut contribuer à exploiter plus tôt certaines variables jugées intéressantes pour la résolution.

Bien que BTD dispose ainsi d'une liberté plus grande pour le choix de la prochaine variable, l'exploitation de la décomposition demeure très contraignante vis-à-vis de la résolution :

- La même décomposition est utilisée tout au long de la résolution. En effet, l'ensemble E de clusters de la décomposition reste inchangé pendant la résolution même si l'ordre exploité sur ces clusters peut éventuellement changer.
- Les clusters de E sont calculés uniquement sur la base de critères structurels avant le début de la résolution. Les ordres pouvant être induits par la décomposition ne sont pas étudiés au préalable. En outre, la compatibilité entre ces ordres et un ordre souhaité à l'égard de la résolution n'est jamais évoquée. En tous cas, beaucoup de caractéristiques de l'instance (comme la difficulté à satisfaire les contraintes ou l'impact des variables) ne sont dévoilées que pendant la résolution.
- Toutes ces contraintes imposées sur l'heuristique de choix de variables empêchent BTD de profiter pleinement des heuristiques de choix de variables adaptatives qui puisent leur intérêt dans leur capacité à offrir un choix plus libre pour la prochaine variable selon les critères définis par cette heuristique.
- Les contraintes imposées sur l'heuristique de choix de variables pourront également empêcher BTD de bénéficier de tous les avantages des redémarrages. En effet, lors d'un redémarrage certaines variables peuvent être jugées très importantes. Un bon algorithme de recherche souhaiterait ainsi les affecter le plus tôt possible dans l'arbre de recherche. Malheureusement, l'emploi d'une décomposition arborescente peut empêcher une telle affectation.

Tous ces éléments pourront détériorer significativement l'efficacité de la résolution.

4.2.2.2 Vers une exploitation plus opportune de la décomposition

L'idée que nous proposons dans ce chapitre s'inspire des travaux réalisés dans [Jégou et al., 2007]. Plus précisément, ces travaux peuvent être considérés comme étant les fondements théoriques de notre travail. Le but de [Jégou et al., 2007] est de trouver un compromis entre les bonnes bornes théoriques induites par l'utilisation des décompositions et

la nécessité absolue d'exploiter des heuristiques de choix de variables efficaces en pratique. Les auteurs proposent une extension de BTD, appelé BDH (pour *Backtracking on Dynamic covering by acyclic Hypergraphs*). Elle est basée sur une exploitation dynamique du recouvrement par un hypergraphe acyclique (CAH). L'approche vise à intégrer des heuristiques de choix de variables plus dynamiques, un critère jugé essentiel pour une résolution efficace en pratique. Ils se focalisent sur des recouvrements acycliques déduits d'un recouvrement dit *de référence*. À partir de ce recouvrement *de référence*, une grande variété de classes d'hypergraphes acycliques peut être définie. Ces classes peuvent être définies selon des critères liés à la nature du recouvrement ou aux relations existantes avec la méthode de résolution. Nous pouvons par exemple citer le recouvrement visant à borner la valeur de certains paramètres comme la largeur ou la taille des séparateurs, la préservation des séparateurs du recouvrement de référence ou la capacité d'implémenter des heuristiques efficaces comme celles qui sont dynamiques. En particulier, les auteurs mettent en avant la classe qui se focalise sur le concept de séparateur. Ce choix est simplement justifié par l'intérêt de ce paramètre vis-à-vis de la complexité spatiale. Ainsi, les recouvrements acycliques qui peuvent être exploités en pratique peuvent être déduits du recouvrement *de référence* en se limitant à un sous-ensemble des séparateurs de ce dernier. En d'autres termes, aucun nouveau séparateur ne peut effectivement être créé puisque ces recouvrements consistent à mettre en commun plusieurs hyperarêtes. Cette extension de BTD a essentiellement permis de proposer un grand nombre d'heuristiques de choix de variables du fait qu'elle se libère de la structure initiale. En effet, BDH n'a pas besoin de déterminer une décomposition unique à exploiter tout au long de la résolution. Au contraire, pendant la recherche, l'hypergraphe courant peut être modifié dans le but de prendre en compte l'évolution du problème. En plus, BDH est capable d'exploiter tous les (no)goods structurels enregistrés pour tous les séparateurs du recouvrement *de référence* incluant des séparateurs qui sont actuellement dans de plus grandes hyperarêtes du recouvrement courant exploité. Les résultats reportés dans [Jégou et al., 2007] se situent principalement sur un plan théorique notamment en ce qui concerne les changements induits vis-à-vis de la complexité temporelle.

Ce que nous proposons dans ce chapitre est d'offrir aux méthodes structurelles telles que BTD l'opportunité d'exploiter - dans le même esprit que BDH - différemment les décompositions. Notre objectif consiste à alléger les contraintes qui handicapent l'heuristique de choix de variables et qui sont à l'origine de la non-compétitivité des méthodes structurelles par rapport aux méthodes non structurelles dans la plupart des cas. Cependant, nous avons choisi de changer dynamiquement la décomposition employée pendant la résolution de façon à suivre ce qui serait souhaitable à l'égard de l'heuristique de choix de variables. En d'autres termes, l'extension de BTD que nous proposons se distingue par des fusions déclenchées dynamiquement sur la base d'informations relatives à la résolution en cours alors que BDH se contente de fusionner des clusters en ne faisant pas croître la taille des clusters au-delà d'une certaine limite fixée au préalable. La démarche que nous proposons s'inscrit dans la lignée des méthodes adaptatives parce qu'elle vise à adapter la décomposition, au sens de l'ensemble de clusters qui la constituent, au contexte de la résolution en se basant sur des informations concernant les états précédents considérés durant la résolution de l'instance ainsi que son état courant.

Par la suite, la section 4.3 présente une exploitation dynamique de la décomposition dans le cadre du problème CSP.

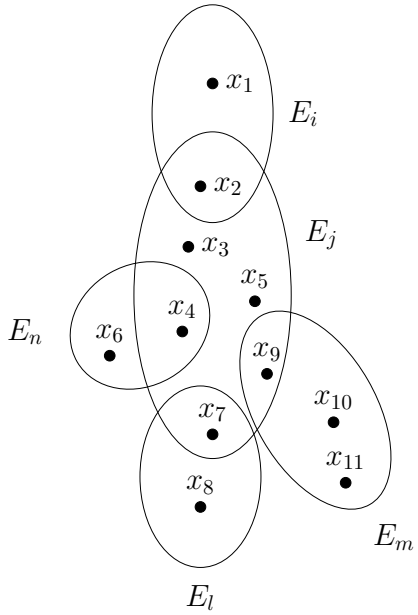

FIGURE 4.2 – Ensemble de clusters de la décomposition de la figure 4.1 après la fusion de E_j et de E_k.

4.3 Modification dynamique de la décomposition via la fusion pour le problème CSP : BTD-MAC+RST+Fusion

Nous visons à offrir à BTD plus de liberté quant à l'heuristique de choix de variables et à ne pas se contenter du niveau de liberté actuellement existant. La thèse que nous défendons ici est que la dynamicité de la décomposition i.e. sa modification pendant la résolution, permet d'adapter la décomposition à la nature de l'instance à résoudre. Par sa modification, nous visons la modification de l'ensemble de clusters E exploités. L'opération permettant de changer la décomposition dans ce contexte est la *fusion* des clusters. Dans ce qui suit, nous proposons l'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ permettant d'intégrer la modification dynamique de la décomposition via la *fusion* des clusters. Ces travaux ont fait l'objet des publications [Jégou et al., 2016b,c].

L'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ (voir l'algorithme 4.2) représente une adaptation de l'algorithme $BTD\text{-}MAC{+}RST$ [Jégou and Terrioux, 2014a] (cf. la partie 2.2.5.1) afin de prendre en compte la fusion dynamique. Pour ces deux algorithmes, la décomposition arborescente est calculée en amont de la résolution. Comme décrit dans la section 4.2.2, l'emploi de cette décomposition enracinée en un cluster E_r induit un ordre partiel sur les variables. La différence principale entre eux réside dans le fait que $BTD\text{-}MAC{+}RST$ utilise cette décomposition initiale durant toute la résolution (au sens de l'ensemble des clusters qui la définissent puisque la racine peut changer), tandis que $BTD\text{-}MAC{+}RST{+}Fusion$ va la faire évoluer dynamiquement. Ainsi, l'ordre partiel imposé par la décomposition change pendant la résolution.

Nous donnons tout d'abord des détails plus amples sur la fusion dynamique.

4.3.1 Fusion dynamique

La fusion consiste à mettre en commun les variables de deux clusters voisins pour former ainsi un seul cluster.

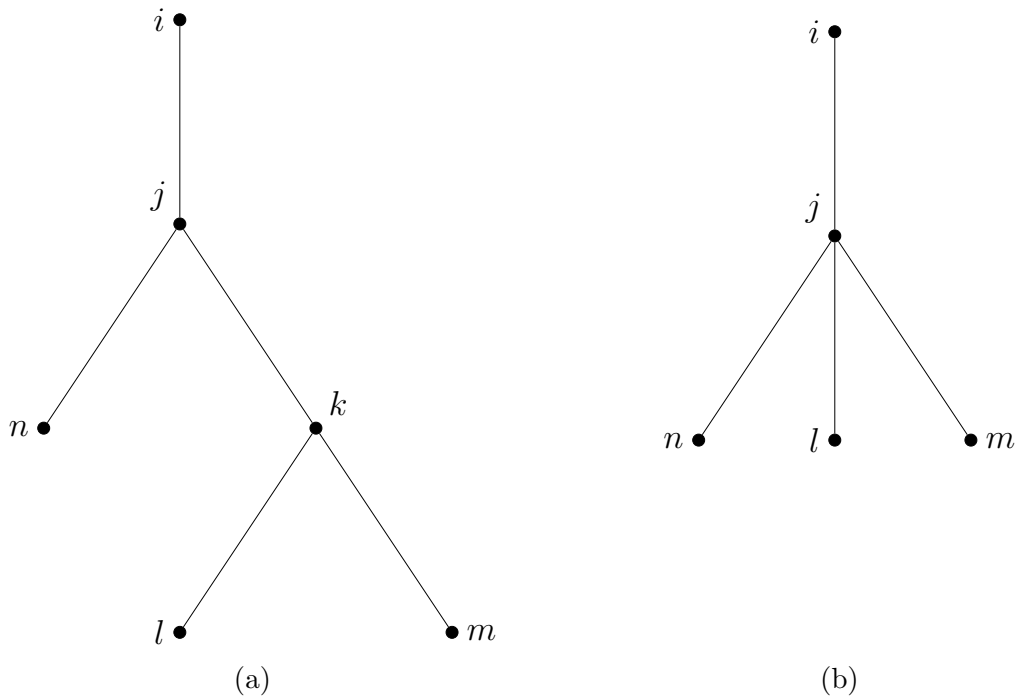

FIGURE 4.3 – L'arbre correspondant à la décomposition avant la fusion (a) et après la fusion (b).

La figure 4.2 montre la fusion des clusters E_j et E_k de la décomposition de la figure 4.1. Le cluster résultant de la fusion du cluster E_j et du cluster E_k est un nouveau cluster E_j. Il est à noter que, les fils du cluster fusionné deviennent les fils du cluster résultant de la fusion. Par exemple, dans la figure 4.1, E_l et E_m, les fils de E_k, deviennent les fils de E_j qui est le cluster résultant de la fusion des clusters E_j et E_k dans la figure 4.2. La figure 4.3 montre à son tour l'arbre T correspondant aux décompositions avant et après la réalisation de la fusion. Soit $\mathcal{D}$ la décomposition initiale et $\mathcal{D}'$ la décomposition après la fusion. Pour le même cluster racine, tout ordre sur les variables permis par $\mathcal{D}$ reste permis par $\mathcal{D}'$. Cependant, en exploitant $\mathcal{D}'$ nous pouvons obtenir plus d'ordres possibles qu'en exploitant $\mathcal{D}$. Nous en déduisons que la fusion préserve les ordres de choix de variables initialement permis tout en offrant plus de liberté. Par exemple, pour $E_r = E_i$, si l'ordre partiel induit par la décomposition dans la figure 4.1 est : $[\{x_1, x_2\}, \{x_3, x_4, x_5\}, \{x_6, [\{x_7, x_9\}, \{x_8, \{x_{10}, x_{11}\}\}]\}]$, l'ordre partiel induit par la décomposition dans la figure 4.2 est : $[\{x_1, x_2\}, \{x_3, x_4, x_5, x_7, x_9\}, \{x_6, x_8, \{x_{10}, x_{11}\}\}]$. Ainsi, l'ordre $[x_1, x_2, x_7, x_9, x_3, x_4, x_5, x_6, x_8, x_{10}, x_{11}]$ est par exemple désormais permis par la décomposition de la figure 4.2 alors qu'il ne l'était pas auparavant par la décomposition de la figure 4.1. Le choix de fusionner ou non des clusters est conditionné par des informations apprises durant la résolution. Aussi, le comportement de $BTD\text{-}MAC{+}RST{+}Fusion$ se situe entre celui de $BTD\text{-}MAC{+}RST$ avec une liberté partielle pour l'ordre sur les variables (si aucune fusion n'est effectuée) et celui de $MAC{+}RST$ avec une liberté totale (si, après une série de fusions, la décomposition ne contient plus qu'un seul cluster). L'intérêt de ce nouvel algorithme est de pouvoir trouver dynamiquement le bon compromis à partir d'informations recueillies durant la résolution. En d'autres termes, son but consiste à exploiter les informations fournies durant la résolution pour adapter en permanence la décomposition et ainsi la liberté de choix de variables au contexte courant de la résolution.

La différence primordiale entre la fusion dite *statique* [Jégou et al., 2005] et la fusion

dynamique réside dans la raison du déclenchement de la fusion des clusters. La fusion dynamique est réalisée pendant la résolution en se basant sur des informations récoltées depuis le début de la résolution jusqu'au moment de la fusion. La fusion statique est faite au préalable de la résolution. Elle se fait habituellement sur la base de critères purement structurels avec tous les inconvénients que cela peut comporter. Par exemple, le but de la fusion statique pourrait consister à limiter la taille des séparateurs en supprimant ceux dans la taille dépasse la limite choisie via la fusion des deux clusters impliqués. D'autres objectifs peuvent être visés comme la création de clusters connexes ou l'augmentation du nombre de variables propres de chaque cluster.

4.3.2 Description de l'algorithme BTD-MAC+RST+Fusion

4.3.2.1 Similitudes avec BTD-MAC+RST

L'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ exploite l'algorithme $BTD\text{-}MAC{+}Fusion$ (voir l'algorithme 4.1). Ce dernier ne se différencie de $BTD\text{-}MAC$ (cf. la partie 2.2.5.1) que par les lignes 17 à 21. Initialement, la suite de décisions Σ ainsi que les ensembles de goods G^d et de nogoods N^d sont vides. $BTD\text{-}MAC{+}RST{+}Fusion$ (à l'instar de $BTD{+}MAC{+}RST$) commence la résolution en assignant les variables du cluster E_r avant de passer à un cluster fils. En exploitant le nouveau cluster E_i, seules les variables non assignées du cluster E_i seront instanciées. En d'autres termes, seules les variables de E_i n'appartenant pas à $E_i \cap E_{p(i)}$ sont instanciées. Pour résoudre chaque cluster les deux algorithmes s'appuient sur MAC (lignes 24-29 et 35-37). Durant la résolution MAC peut prendre deux types de décisions :

- Décisions positives $x_i = v_i$ qui assignent la valeur v_i à la variable x_i (ligne 27),
- Décisions négatives $x_i \neq v_i$ qui assurent que v_i ne peut être assignée à x_i (ligne 35).

Supposons que $\Sigma =< \delta_1, \ldots, \delta_i >$ est la suite de décisions courante où chaque δ_j peut être une décision soit positive, soit négative. Une nouvelle décision positive $x_{i+1} = v_{i+1}$ est prise et un filtrage par cohérence d'arc (AC) est accompli (ligne 27). Si aucune incohérence n'est détectée, la recherche continue normalement (ligne 28). Sinon la valeur v_{i+1} est supprimée de $D_{x_{i+1}}$ et un nouveau filtrage par AC est effectué (ligne 35). Si une incohérence est détectée, nous procédons à un retour-arrière et la dernière décision positive $x_l = v_l$ est changée en $x_l \neq v_l$. Lorsque le cluster E_i est choisi comme cluster suivant, nous savons que la prochaine décision positive implique une variable de E_i. Étant donné que le séparateur $E_i \cap E_{p(i)}$ est instancié, le filtrage par AC impacte uniquement les variables des clusters de $Desc(E_i)$. Une fois le cluster E_i complètement instancié de façon cohérente (ligne 1), chaque sous-problème enraciné en un cluster E_j, fils de E_i, sera résolu (ligne 11). Plus précisément, pour un cluster fils E_j et une suite de décisions Σ, nous résolvons le problème enraciné en E_j et induit par $Pos(\Sigma)[E_i \cap E_j]$ où $Pos(\Sigma)[E_i \cap E_j]$ est l'ensemble des décisions positives impliquant les variables de $E_i \cap E_j$ dans Σ. Si nous trouvons une extension cohérente de Σ sur $Desc(E_j)$, $Pos(\Sigma)[E_i \cap E_j]$ est enregistré comme *good structurel* (ligne 12-13). Si, au contraire, la résolution montre qu'il n'existe aucune extension cohérente de Σ sur $Desc(E_j)$, $Pos(\Sigma)[E_i \cap E_j]$ est enregistré comme un *nogood structurel* (lignes 15-16). Ces (no)good structurels sont utilisés ultérieurement dans la recherche pour éviter certaines redondances (lignes 7-8 et 10). Si un redémarrage est déclenché (ligne 31), la résolution est interrompue. La gestion des redémarrages est faite comme dans [Jégou and Terrioux, 2014a] et s'accompagne de l'enregistrement de *nld-nogoods réduits* [Lecoutre et al., 2007e] qui permettent de ne pas réexplorer des parties de l'espace de recherche déjà

Algorithme 4.1 : BTD-MAC+Fusion $(P, \Sigma, E_i, V_{E_i}, G^d, N^d)$

Entrées-Sorties : $P = (X, D, C)$: une instance CSP
Entrées : Σ : suite de décisions ; E_i : cluster ; V_{E_i} : ensemble de variables
Entrées-Sorties : G^d : ensemble de goods ; N^d : ensemble de nogoods
Sorties : *vrai* si une solution au sous-problème de P enraciné en E_i et induit par Σ a été trouvée, $faux$ s'il est prouvé qu'il n'en possède pas, *inconnu* sinon

```
1  si V_Ei = ∅ alors
2      résultat ← vrai
3      Q_Ei ← Fils(E_i)                    /* Fils(E_i) : ensemble des clusters fils de E_i */
4      tant que résultat ∉ {faux, inconnu} et Q_Ei ≠ ∅ faire
5          Choisir un cluster E_j ∈ Q_Ei
6          Q_Ei ← Q_Ei \ {E_j}
7          si Pos(Σ)[E_i ∩ E_j] est un nogood dans N^d alors
8              résultat ← faux
9          sinon
10             si Pos(Σ)[E_i ∩ E_j] n'est pas un good de E_i par rapport à E_j dans G^d alors
11                 résultat ← BTD-MAC+Fusion(P,Σ,E_j, E_j\(E_i ∩ E_j),G^d,N^d)
12                 si résultat = vrai alors
13                     Enregistrer Pos(Σ)[E_i ∩ E_j] comme good de E_i par rapport à E_j dans G^d
14                 sinon
15                     si résultat = faux alors
16                         Enregistrer Pos(Σ)[E_i ∩ E_j] comme nogood de E_i par rapport à E_j dans N^d
17                     sinon
18                         si fusion alors Fusionner E_j avec un de ses fils
19                         si non redémarrage alors
20                             Q_Ei ← Q_Ei ∪ {E_j}
21                             résultat ← vrai
22     retourner résultat
23 sinon
24     Choisir une variable x ∈ V_Ei
25     Choisir une valeur v ∈ D_x
26     D_x ← D_x \ {v}
27     si AC (P,Σ ∪ ⟨x = v⟩) alors
28         résultat ← BTD-MAC+Fusion(P,Σ∪ ⟨x = v⟩,E_i, V_Ei\{x},G^d,N^d)
29     sinon résultat ← faux
30     si résultat = faux alors
31         si redémarrage ou fusion alors
32             Enregistrer nld-nogoods par rapport à la suite de décisions Σ[E_i]
33             retourner inconnu
34         sinon
35             si AC (P,Σ ∪ ⟨x ≠ v⟩) alors
36                 retourner BTD-MAC+Fusion(P,Σ ∪ ⟨x ≠ v⟩,E_i, V_Ei,G^d,N^d)
37             sinon retourner faux
38     sinon  retourner résultat
```

visitées. L'intérêt du redémarrage réside dans l'exploitation des connaissances acquises auparavant via les (no)good structurels et les nld-nogoods réduits [Lecoutre et al., 2007e]. Le déclenchement d'un redémarrage peut être conditionné par des paramètres globaux

Algorithme 4.2 : BTD-MAC+RST+Fusion (P)

Entrées : $P = (X, D, C)$: une instance CSP
Sorties : *vrai* si P possède une solution, $faux$ sinon

```
1 G^d ← ∅ ; N^d ← ∅
2 répéter
3 |   Choisir un cluster racine E_r
4 |   résultat ← BTD-MAC+Fusion (P,∅,E_r,E_r,G^d,N^d)
5 jusqu'à résultat ≠ inconnu
6 retourner résultat
```

(portant sur l'ensemble du problème) ou locaux (relatifs au cluster courant) ou aussi une combinaison des deux. Pour plus de détails sur les redémarrages sous BTD, nous invitons les lecteurs à se référer à la partie 2.2.5.1 ou aux travaux cités ci-dessus.

4.3.2.2 Modifications réalisées pour BTD-MAC+RST+Fusion

Les lignes 17-21 concernent uniquement la fusion dynamique donc $BTD\text{-}MAC{+}Fusion$. La fusion dynamique vise, comme expliqué ci-dessus, à donner plus de liberté à l'heuristique de choix de variables, et en même temps, à limiter l'impact des défauts potentiels mis en avant dans la partie 4.2.2. Le choix de fusionner ou non des clusters va se faire sur la base des critères s'appuyant sur l'état courant du problème, mais aussi sur tout ou partie des états précédemment rencontrés durant la résolution. Ce choix est implémenté via la fonction $fusion$ qui retourne *vrai* si une fusion est nécessaire et $faux$ sinon. Cette fonction peut être librement implémentée par l'utilisateur qui pourrait intégrer les critères qu'il souhaite faire entrer en jeu dans la décision de la fusion ou de la non-fusion des clusters. Si aucune fusion n'est requise ($fusion$ renvoie $faux$), la résolution se poursuit normalement. Au contraire, si ce critère considère que fusionner le cluster courant avec un de ses fils permettrait de rendre la suite de la résolution plus efficace (par exemple en permettant d'instancier plus tôt certaines variables jouant un rôle clé), $fusion$ renvoie *vrai* et $BTD\text{-}MAC{+}Fusion$ va modifier la décomposition courante en fusionnant ces deux clusters (ligne 18). La figure 4.4 montre l'affectation des variables du cluster E_j (les variables affectées sont colorées en rouge). Supposons, par exemple, qu'après l'affectation de la variable x_4, la fonction $fusion$ retourne *vrai* pour le cluster E_k. La décomposition va alors être modifiée et les clusters E_j et E_k fusionnés. Pour cela, nous commençons par désaffecter les variables instanciées du cluster courant et par enregistrer des nld-nogoods réduits (ligne 32) comme le ferait un redémarrage classique. Dans la figure 4.5, les variables x_3 et x_4 sont alors désaffectées et la recherche retourne au cluster E_i. Une fois revenu dans le cluster parent, nous procédons à la fusion. À ce stade (ligne 19), soit nous continuons à revenir en arrière si *redémarrage* est vrai, soit la recherche se poursuit avec l'exploration d'un fils du cluster parent. Par exemple, la figure 4.6 montre que $BTD\text{-}MAC{+}Fusion$ explore le nouveau cluster E_j et est désormais capable d'affecter d'abord les variables x_7 et x_9. Notons que désaffecter les variables du cluster courant avant de le fusionner avec un de ses fils n'est pas une nécessité. Toutefois, ce choix devrait permettre d'exploiter au plus tôt les variables nouvellement ajoutées dans ce cluster.

L'algorithme $BTD\text{-}MAC{+}Fusion$ est paramétrable notamment par l'heuristique de fusion (nous en proposons une dans la partie expérimentale). Un bon choix pour cette heuristique devrait permettre d'améliorer considérablement la résolution en la rendant plus efficace.

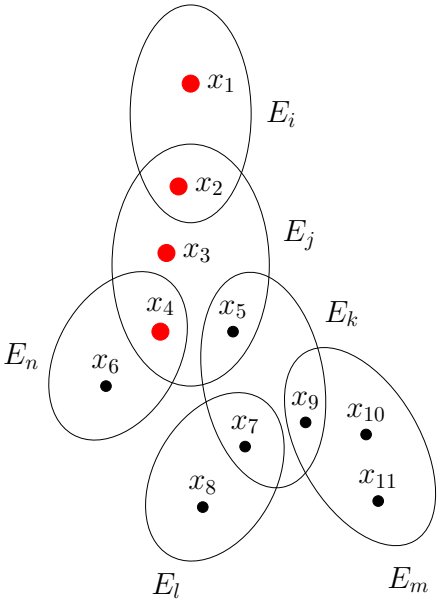

FIGURE 4.4 – Illustration de l'affectation du cluster E_j.

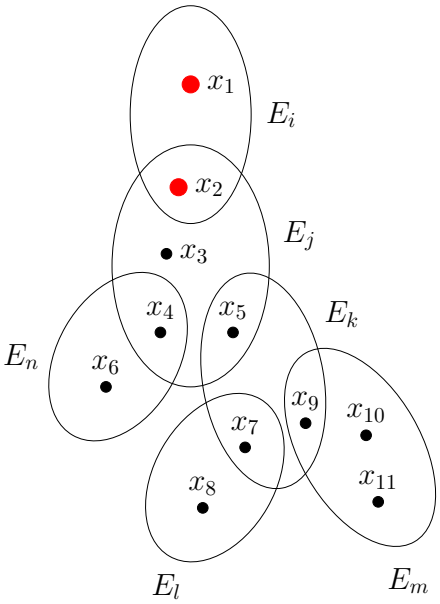

FIGURE 4.5 – Illustration du retour-arrière vers le cluster E_i.

4.3.3 Fondements théoriques

Nous montrons dans cette partie la validité de notre approche. Tout d'abord, nous prouvons que le fait de fusionner deux clusters ne remet pas en cause la validité des (no)goods structurels et des nld-nogoods.

Propriété 5 *Soit (E', T') la décomposition arborescente d'un graphe G obtenue à partir de la décomposition (E, T) de G en fusionnant le cluster E_y avec le cluster E_x (avec E_y fils de E_x dans (E, T)).*
Les (no)good structurels de tout cluster E_i par rapport à un cluster fils E_j (avec $E_j \neq E_y$) et les nld-nogoods réduits enregistrés vis-à-vis de (E, T) restent valides vis-à-vis de (E', T'),

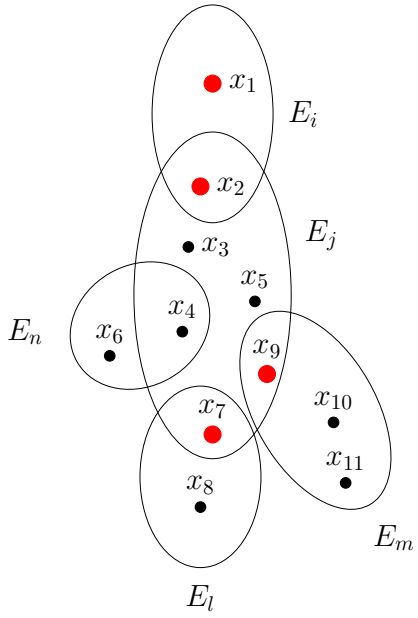

FIGURE 4.6 – Illustration de l'affectation du nouveau cluster E_j.

c'est-à-dire que leur utilisation lors de l'exploitation de (E', T') *ne nuit pas à la validité de BTD-MAC+RST+Fusion.*

Preuve : Soit Δ un good structurel de E_i par rapport à son fils E_j enregistré pour (E, T). Sachant que E_j et E_y sont différents, le sous-problème de P enraciné en E_j dans (E, T) est identique au sous-problème de P enraciné en E_j dans (E', T'). Ainsi, si Δ peut être étendu d'une façon cohérente sur le premier sous-problème, il peut également l'être sur le deuxième sous-problème. En conséquence, Δ est un good structurel valide de E_i par rapport à E_j dans (E', T'). Le raisonnement est similaire pour un nogood structurel.

Un nld-nogood réduit est un nogood quelle que soit la décomposition considérée. Nous devons uniquement vérifier qu'un nld-nogood Δ est valide pour la décomposition (E', T'). En d'autres termes, nous devons vérifier qu'il existe un cluster de E' incluant toutes les variables de Δ. Par construction, il existe nécessairement un cluster E_p de (E, T) couvrant Δ. Si $E_p \neq E_x$ et $E_p \neq E_y$, alors $E_p \in E'$. Sinon, après fusion, nous avons $E_p \subset E_x$ et $E_x \in E'$. Par conséquent, dans les deux cas, les variables de Δ sont toutes recouvertes par un seul cluster de E' et ainsi Δ est valide pour (E', T').□

Nous prouvons maintenant la validité de notre algorithme.

Théorème 9 *BTD-MAC+RST+Fusion est correct, complet et termine.*

Preuve : Considérons premièrement *BTD-MAC+Fusion* qui diffère de *BTD-MAC* par l'exploitation de la fusion. Supposons que nous obtenons (E', T') à partir de la décomposition (E, T) après la fusion de d'un cluster E_i et de son fils E_j. Soit Σ_f la suite de décisions pour laquelle la fonction *fusion* renvoie *vrai* lors de l'affectation des variables de E_i. Certains nld-nogoods réduits sont ainsi enregistrés et la recherche opère un retour-arrière au cluster $E_{p(i)}$ parent du cluster courant E_i. Ainsi, nous obtenons la suite Σ'_f qui correspond à la suite de décisions Σ_f restreinte aux variables des clusters présents dans la branche allant du cluster racine E_r au cluster $E_{p(i)}$. *BTD-MAC+Fusion* continue sa recherche à partir de $E_{p(i)}$ avec Σ'_f en exploitant la décomposition (E', T'). Le cluster

résultant de la fusion (le nouveau cluster E_i) peut éventuellement être le cluster suivant visité ou être visité ultérieurement.

Nous constatons que :

- L'arbre de recherche exploré par $BTD\text{-}MAC{+}Fusion$ depuis son premier appel avec une suite de décisions vide jusqu'à la suite de décisions Σ_f est le même que celui développé par $BTD\text{-}MAC$ dans les mêmes circonstances pour la décomposition (E, T).
- En plus, après la fusion, l'arbre de recherche développé par $BTD\text{-}MAC{+}Fusion$ depuis la suite de décisions Σ'_f jusqu'à sa terminaison est identique à celui développé par $BTD\text{-}MAC$ dans les mêmes circonstances pour la décomposition (E', T').

Nous savons que $BTD\text{-}MAC$ est complet (si aucun redémarrage n'a eu lieu), correct et termine [Jégou and Terrioux, 2014a]. En outre, selon la proposition 5, les (no)goods structurels et les nld-nogoods réduits enregistrés pour (E, T) restent valides pour la nouvelle décomposition (E', T'). Ainsi, la correction, la terminaison et la complétude de l'algorithme ne sont pas mises en danger. Quant aux redémarrages, l'enregistrement de nld-nogoods réduits à chaque redémarrage empêche l'exploration d'une partie de l'espace de recherche déjà explorée. Ainsi, $BTD\text{-}MAC{+}Fusion$ est complet (si aucun redémarrage n'a eu lieu), correct et termine.

En plus, lorsque plusieurs opérations de fusion sont réalisées, le même raisonnement peut être appliqué pour chaque fusion en partitionnant l'arbre de recherche.

Les redémarrages arrêtent la recherche sans changer le fait que si une solution existait dans l'espace de recherche visité par $BTD\text{-}MAC{+}Fusion$ alors $BTD\text{-}MAC{+}Fusion$ l'aurait trouvée. Comme $BTD\text{-}MAC{+}RST{+}Fusion$ fait seulement plusieurs appels à $BTD\text{-}MAC{+}Fusion$, il est correct.

L'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ est complet, correct et termine :

- En ce qui concerne la complétude, si l'appel à $BTD\text{-}MAC{+}Fusion$ n'est pas arrêté par un redémarrage (ce qui est nécessairement le cas du dernier appel à $BTD\text{-}MAC{+}Fusion$ si $BTD\text{-}MAC{+}RST{+}Fusion$ termine), la complétude de $BTD\text{-}MAC{+}Fusion$ implique celle de $BTD\text{-}MAC{+}RST{+}Fusion$.
- En outre, l'enregistrement des nld-nogoods réduits à chaque redémarrage évite l'exploration d'une partie de l'espace de recherche déjà explorée par un appel antérieur à $BTD\text{-}MAC{+}Fusion$. Il en découle que, suite aux appels successifs à l'algorithme $BTD\text{-}MAC{+}Fusion$, l'exploration de l'espace de recherche se fait sur une partie de plus en plus strictement réduite. Ainsi, la terminaison et la complétude de $BTD\text{-}MAC{+}RST{+}Fusion$ sont garanties par l'enregistrement non limité des nogoods réalisé suite aux différents appels à $BTD\text{-}MAC{+}Fusion$ et aussi grâce à la terminaison et la complétude de $BTD\text{-}MAC{+}Fusion$.

□

Nous nous intéressons maintenant aux complexités en temps et en espace et nous énonçons le théorème suivant :

Théorème 10 *L'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ a une complexité en temps en $O(R.((n.s^2.m.\log(d) + w'^{+}.|N_r|).d^{w'^{+}+2} + n.(w'^{+})^2.d))$ et une complexité en espace en $O(n.s.d^s + w'^{+}.(d + |N_r|))$ avec w'^{+} la largeur de la décomposition arborescente finale, s la taille de la plus grande intersection $E_i \cap E_j$ de la décomposition initiale, R le nombre de redémarrages et $|N_r|$ le nombre de nld-nogoods réduits mémorisés.*

Preuve : L'algorithme $BTD\text{-}MAC{+}RST$ a une complexité en temps en $O(((n.s^2.m.\log(d)+ w^+.|N_r|).d^{w^+ +2} + n.(w^+)^2.d).R)$ et une complexité spatiale en $O(n.s.d^s + w^+.(d + |N_r|))$ [Jégou and Terrioux, 2014a] (cf. la partie 2.2.5.1). En ce qui concerne l'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$, l'application des opérations de fusion implique que la taille des clusters peut éventuellement augmenter. Ainsi, les complexités théoriques sont exprimées en fonction de w'^+ au lieu de w^+. Les opérations de fusion ne créent pas de nouveaux séparateurs, mais au contraire, en suppriment quelques-uns. Ainsi, la taille maximale des séparateurs de la décomposition initiale représente une borne supérieure sur la taille des séparateurs utilisés durant la recherche. C'est pourquoi, les éléments des complexités spatiales et temporelles relatifs à la taille des séparateurs ne sont pas modifiés. Pour ce qui est des nld-nogoods réduits enregistrés après une opération de fusion, bien qu'ils induisent des coûts additionnels en espace et en temps, ces coûts sont déjà pris en compte par les coûts des nld-nogoods réduits enregistrés lors des redémarrages. En conséquence, la complexité en temps est de $O(R.((n.s^2.m.\log(d) + w'^+.|N_r|).d^{w'^+ +2} + n.(w'^+)^2.d))$ et celle en espace est de $O(n.s.d^s + w'^+.(d + |N_r|))$.□

Notons qu'il est possible de limiter l'augmentation de la largeur de la décomposition finale par rapport à la décomposition initiale en utilisant une heuristique de fusion prenant en compte ce paramètre.

4.4 Étude expérimentale

Dans cette section, nous évaluons l'intérêt pratique des décompositions dynamiques. Nous présentons tout d'abord le protocole expérimental.

4.4.1 Protocole expérimental

Nous reprenons pour ces expérimentations le protocole expérimental de la partie 3.4.2 du chapitre 3. Au niveau des décompositions exploitées, nous retenons $Min\text{-}Fill$, H_2, H_3 et H_5 qui a montré plus d'intérêt que H_4 dans le cadre de l'exploitation dynamique de la décomposition. Pour H_5, la décomposition est exploitée sans limite sur la taille des séparateurs pour laisser plus de latitude à la fusion. Ainsi, tout séparateur trouvé lors du calcul des clusters de H_5 est conservé.

La décomposition dynamique exploite une heuristique de fusion. Cette dernière se base sur les conseils de l'heuristique de choix de variables pour évaluer le besoin de la fusion des clusters. Plus précisément, étant donné un cluster courant E_i, à chaque fois que nous choisissons la variable suivante à affecter dans E_i, on teste si l'heuristique de choix de variables aurait choisi une autre variable si elle avait l'opportunité de choisir parmi les variables non affectées de $(\bigcup_{E_j \in Fils(E_i)} E_j) \cup E_i$. Si une variable d'un fils E_j de E_i est préférée à une variable de E_i, un compteur relatif à E_j est incrémenté. Lorsque le compteur relatif à E_j atteint une limite L (à savoir 100 dans nos expérimentations), le cluster E_j est fusionné avec son parent E_i. Ainsi, les variables appartenant auparavant à E_j appartiennent désormais à E_i. Elles peuvent ainsi être assignées avant toute variable propre de E_i d'origine. La valeur de 100 utilisée dans les expérimentations n'est sûrement pas garantie d'être la valeur optimale pour toutes les instances, même pas pour une instance donnée. Il s'agit en revanche d'une valeur qui a permis une résolution efficace pour un grand nombre d'instances du benchmark considéré. Elle assure un compromis entre une valeur très élevée qui diminuerait fortement la fréquence des fusions en les rendant très contraintes et entre une valeur très basse qui induirait une fusion de E_i avec E_j sans que

Algorithme	*Min-Fill*		H_2		H_3		H_5^∞	
	#rés	temps	#rés	temps	#rés	temps	#rés	temps
`BTD-MAC`	1 348	34 416	1 424	21 860	1 487	28 792	1 430	23 408
`BTD-MAC+RST`	1 507	33 632	1 536	22 854	1 558	26 418	1 538	25 649
`BTD-MAC+Fusion`	1 497	32 206	1 533	25 475	1 547	28 630	1 550	22 828
`BTD-MAC+RST+Fusion`	1 558	35 444	1 572	27 291	1 584	27 809	1 592	27 185

TABLE 4.1 – Nombre d'instances résolues et temps de résolution pour *BTD-MAC*, *BTD-MAC+RST*, *BTD-MAC+Fusion* et *BTD-MAC+RST+Fusion* selon les décompositions exploitées.

la nécessité de cette opération ne soit suffisamment constatée.

4.4.2 Observations et analyse des résultats

Idée générale sur l'intérêt de la fusion dynamique Le tableau 4.1 fournit le nombre d'instances résolues et le temps de résolution pour chaque algorithme selon chaque décomposition considérée. La comparaison des décompositions avec *BTD-MAC* et *BTD-MAC+RST* a été déjà faite dans la partie 3.4.2 du chapitre 3. Elle a montré l'intérêt des décompositions H_2, H_3 et H_5 vis-à-vis de *Min-Fill*, notamment celui de H_5 qui limite la taille des séparateurs. Nous considérons cependant maintenant une version de H_5 qui ne limite pas la taille des séparateurs puisque cela est plus intéressant pour l'exploitation dynamique de la décomposition même si son utilisation est moins efficace que H_3 avec *BTD-MAC* et *BTD-MAC+RST*. Elle est notée H_5^∞.

Nous constatons que quelle que soit la décomposition utilisée *BTD-MAC+Fusion* résout plus d'instances que *BTD-MAC*. En occurrence, *BTD-MAC+Fusion* résout 149 instances de plus avec *Min-Fill* par rapport à *BTD-MAC*, 109 instances de plus avec H_2, 60 instances de plus avec H_3 et finalement 120 instances de plus avec H_5^∞. La figure 4.7

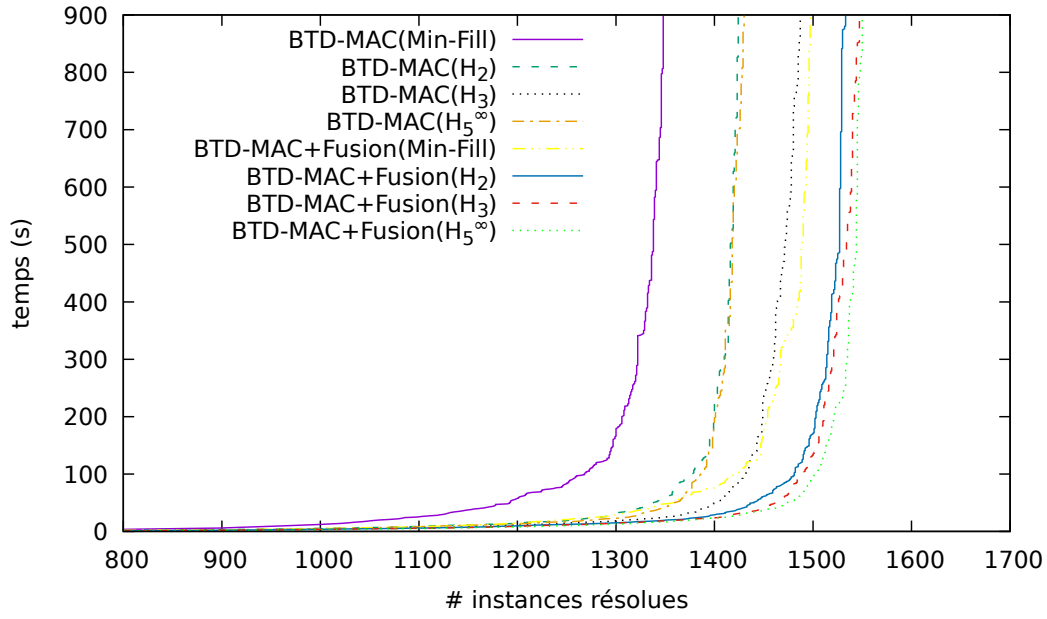

FIGURE 4.7 – Le nombre cumulé d'instances résolues pour les algorithmes *BTD-MAC* et *BTD-MAC+Fusion* selon la décomposition utilisée.

montre le nombre cumulé d'instances résolues pour $BTD\text{-}MAC$ et $BTD\text{-}MAC{+}Fusion$ avec les décompositions $Min\text{-}Fill$, H_2, H_3 et H_5^∞. La figure montre clairement l'amélioration apportée par $BTD\text{-}MAC{+}Fusion$ par rapport à $BTD\text{-}MAC$. Ceci reste valide avec l'exploitation des redémarrages. La figure 4.8 montre également l'intérêt de la fusion dyna-

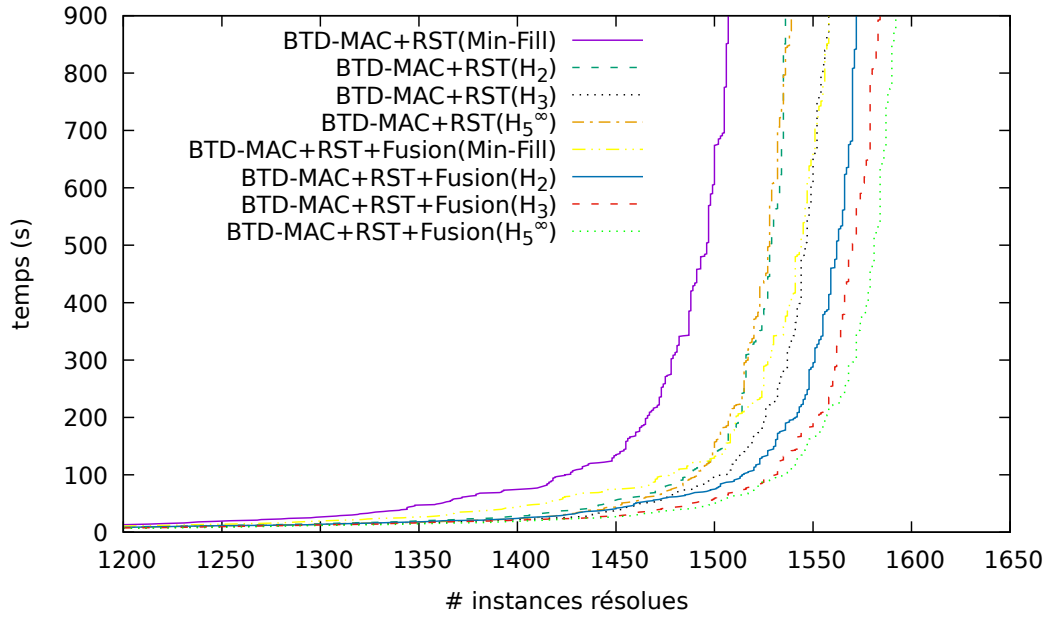

FIGURE 4.8 – Le nombre cumulé d'instances résolues pour $BTD\text{-}MAC{+}RST$ et $BTD\text{-}MAC{+}RST{+}Fusion$ selon la décomposition utilisée.

mique utilisée conjointement avec les redémarrages. En effet, $BTD\text{-}MAC{+}RST{+}Fusion$ résout 51 instances de plus avec $Min\text{-}Fill$ par rapport à $BTD\text{-}MAC{+}RST$, 36 instances de plus avec H_2, 26 instances de plus avec H_3 et 54 instances en plus avec H_5^∞. Les temps de résolution sont soit comparables, soit meilleurs avec la fusion dynamique que sans la fusion dynamique. D'ailleurs, même lorsque le nombre d'instances résolues augmentent considérablement, le temps de résolution cumulé correspondant à la fusion dynamique reste comparable à celui de l'algorithme opérant sans fusion. Il peut même diminuer comme dans le cas de $Min\text{-}Fill$ (34 416 s avec $BTD\text{-}MAC$ et 32 206 s avec $BTD\text{-}MAC{+}Fusion$). L'exploitation de la fusion dynamique avec la décomposition H_5^∞ permet de résoudre le plus grand nombre d'instances sans et avec redémarrage. Vu que $Min\text{-}Fill$ est considéré comme étant l'heuristique de l'état de l'art et comme $BTD\text{-}MAC{+}RST$ est une méthode référence parmi les méthodes structurelles, nous comparons dans la figure 4.9 les deux combinaisons $BTD\text{-}MAC{+}RST$ avec $Min\text{-}Fill$ et $BTD\text{-}MAC{+}RST{+}Fusion$ avec H_5^∞. La figure montre que pour la plupart d'instances du benchmark, soit l'instance est résolue nettement plus rapidement avec $BTD\text{-}MAC{+}RST{+}Fusion$ qu'avec $BTD\text{-}MAC{+}RST$, soit les temps de résolution sont comparables pour les deux méthodes. L'intérêt de l'exploitation dynamique de la décomposition par rapport à son exploitation statique est ainsi mis en évidence.

Apport de la fusion dynamique selon la décomposition ou l'algorithme utilisé L'apport de la fusion dynamique semble d'autant plus important que la qualité de la décomposition vis-à-vis de la résolution est notoirement moins bonne. Par exemple, $BTD\text{-}MAC{+}Fusion$ améliore $BTD\text{-}MAC$ d'une façon plus remarquable avec $Min\text{-}Fill$

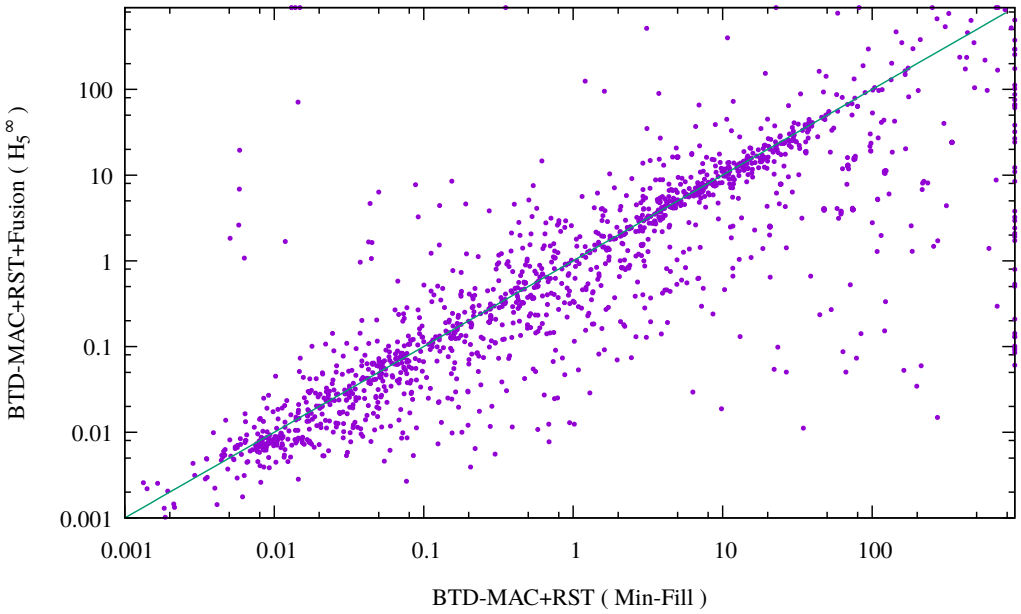

FIGURE 4.9 – Comparaison des temps d'exécution de $BTD\text{-}MAC{+}RST$ avec $Min\text{-}Fill$ et de $BTD\text{-}MAC{+}RST{+}Fusion$ avec H_5^∞.

qu'avec une autre décomposition comme H_3. La raison principale est que l'ordre de choix de variables induit par $Min\text{-}Fill$ est souvent plus contraignant que celui pouvant être induit par une décomposition telle que H_3. Ainsi, l'exploitation de la fusion dynamique peut plus facilement améliorer $BTD\text{-}MAC$ lorsque $Min\text{-}Fill$ est employée que lors de l'utilisation de H_3. En plus, l'amélioration produite par la fusion est plus perceptible lorsque les redémarrages ne sont pas exploitées. En effet, l'exploitation des redémarrages compense en partie les limitations imposées par la décomposition sur l'heuristique de choix de variables, ce qui augmente l'efficacité de $BTD\text{-}MAC{+}RST$ par rapport à $BTD\text{-}MAC$. Plus précisément, lorsque $BTD\text{-}MAC{+}RST$ fait un redémarrage un nouveau cluster racine est choisi. Cela peut être vu comme une forme de dynamicité pour la décomposition. Le même raisonnement réalisé pour la comparaison entre l'apport de la fusion dynamique avec $Min\text{-}Fill$ et celui avec H_3 peut être appliqué pour la comparaison entre l'apport de la fusion dynamique avec redémarrages et celui sans redémarrage. Nous pouvons remarquer aussi que $BTD\text{-}MAC{+}Fusion$ et $BTD\text{-}MAC{+}RST$ se rapprochent vis-à-vis du nombre d'instances résolues ou du temps de résolution. En plus, l'exploitation des redémarrages conjointement avec la décomposition dynamique est vraiment pertinente vu que $BTD\text{-}MAC{+}RST{+}Fusion$ surpasse $BTD\text{-}MAC{+}RST$ et $BTD\text{-}MAC{+}Fusion$.

Comparaison des temps cumulés d'exécution sur les instances résolues par toutes les combinaisons algorithme/décomposition Concernant les temps de résolution, pour une comparaison plus équitable, nous considérons dans le tableau 4.2, les 1 232 instances résolues par tous les algorithmes. Encore une fois, nous obtenons les meilleurs temps de résolution avec H_5^∞ et les pires avec $Min\text{-}Fill$. Plus précisément, le meilleur temps d'exécution cumulé est réalisé par H_5^∞ avec $BTD\text{-}MAC{+}RST$, $BTD\text{-}MAC{+}Fusion$ ou $BTD\text{-}MAC{+}RST{+}Fusion$ (la différence entre les trois méthodes est négligeable) tandis que le pire est celui réalisé par le couple $BTD\text{-}MAC$ et $Min\text{-}Fill$. L'exploitation de la fusion et/ou des redémarrages améliore le temps d'exécution cumulé

Algorithme	*Min-Fill*	H_2	H_3	H_5^∞
BTD-MAC	25 141	12 277	11 371	9 993
BTD-MAC+RST	16 042	10 157	9 478	9 088
BTD-MAC+Fusion	16 579	10 153	9 763	9 039
BTD-MAC+RST+Fusion	16 151	10 166	9 794	9 197

TABLE 4.2 – Le temps de résolution de *BTD-MAC*, *BTD-MAC+RST*, *BTD-MAC+Fusion* et *BTD-MAC+RST+Fusion* selon les décompositions exploitées pour les 1 232 instances résolues par tous les algorithmes.

Algorithme	*Min-Fill*$^{5\%}$		$H_2^{5\%}$		$H_3^{5\%}$		$H_5^{5\%}$	
	#rés	temps	#rés	temps	#rés	temps	#rés	temps
BTD-MAC	1 483	32 998	1 514	25 413	1 519	27 251	1 524	26 143
BTD-MAC+RST	1 561	32 640	1 578	28 396	1 576	24 993	1 586	24 520

TABLE 4.3 – Nombre d'instances résolues et temps de résolution pour *BTD-MAC* et *BTD-MAC+RST* selon les décompositions exploitées avec une fusion statique limitant la taille des séparateurs à 5% du nombre de variables du problème (taille limitée entre 4 et 50).

pouvant être obtenu avec *Min-Fill* tout en restant supérieur aux temps réalisés sous d'autres décompositions.

Fusion dynamique vs fusion statique Nous comparons maintenant la fusion dynamique avec le concept de la fusion statique proposé dans [Jégou et al., 2005], qui peut être réalisée après le calcul d'une décomposition arborescente indépendamment de l'algorithme utilisé pour la calculer (par exemple, *Min-Fill*, H_2, H_3 ou même H_5). La fusion statique effectue des fusions en se basant sur la taille des séparateurs qui est limitée à 5% de n. D'après le tableau 4.3, nous pouvons noter que, concernant le nombre d'instances résolues, *BTD-MAC+Fusion* est significativement meilleur que *BTD-MAC* tandis que *BTD-MAC+RST+Fusion* et *BTD-MAC+RST* sont plus comparables. Plus précisément, *BTD-MAC+RST+Fusion* est plus efficace pour H_3 et H_5 que *BTD-MAC+RST* et est légèrement moins efficace que *BTD-MAC+RST* pour *Min-Fill* et H_2. Ainsi, la fusion dynamique permet d'obtenir de résultats nettement meilleurs par rapport à la fusion statique si les redémarrages ne sont pas exploités et des résultats comparables si les redémarrages sont utilisés. Au-delà, en fusionnant les clusters pendant la résolution, nous adaptons la décomposition selon des connaissances sémantiques de l'instance tandis que la fusion statique se base uniquement sur des critères structurels et requiert de choisir une limite convenable sur la taille des séparateurs ce qui pourrait être particulièrement difficile.
En effet, cette taille dépend en grande partie de l'instance considérée.

Finalement, nous retenons le couple *BTD-MAC+RST+Fusion* et H_5^∞ qui permet d'obtenir les meilleurs résultats en termes du nombre d'instances résolues parmi toutes les combinaisons d'un algorithme de résolution et d'une méthode de calcul de décomposition arborescente.

BTD-MAC+RST+Fusion vs MAC+RST Nous comparons à présent l'algorithme *BTD-MAC+RST+Fusion* contre *MAC+RST* vis-à-vis de l'efficacité de la résolution. La figure 4.10 représente le nombre cumulé d'instances résolues pour l'algorithme *BTD-*

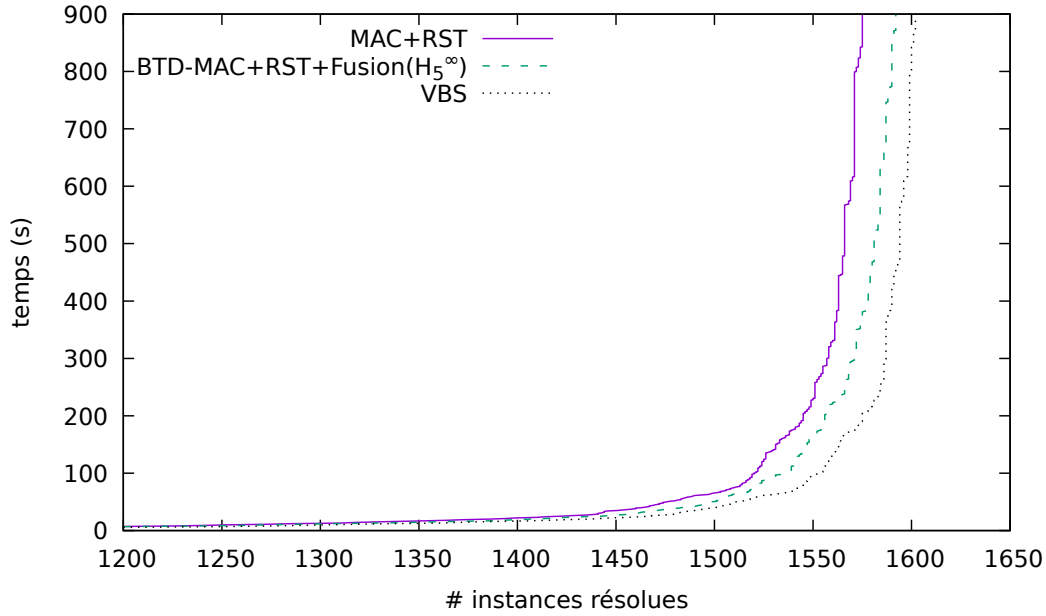

FIGURE 4.10 – Le nombre cumulé d'instances résolues pour $BTD\text{-}MAC{+}RST{+}Fusion$ avec H_5^∞, $MAC{+}RST$ et VBS.

$MAC{+}RST{+}Fusion$, $MAC{+}RST$ et VBS (pour Virtual Best Solver parmi les deux algorithmes). Premièrement, l'algorithme $BTD\text{-}MAC{+}RST{+}Fusion$ résout plus d'instances que $MAC{+}RST$ (1 592 instances contre 1 575). Aussi, nous pouvons noter que le comportement de $BTD\text{-}MAC{+}RST{+}Fusion$ est plus proche de celui du VBS que $MAC{+}RST$, ce qui montre clairement que $BTD\text{-}MAC{+}RST{+}Fusion$ a une meilleure performance que $MAC{+}RST$. Notons finalement que sur ce benchmark, dans 46% des cas, $BTD\text{-}MAC{+}RST{+}Fusion$ ne réalise aucune fusion tandis que dans 6% des cas il fusionne au contraire tous les clusters de la décomposition conduisant à une décomposition à un seul cluster.

Nous focalisons maintenant nos observations sur les instances les plus difficiles. Parmi les 1 859 instances considérées, quelques unes d'entre elles sont facilement résolues par $MAC{+}RST$ (par exemple 286 instances sont résolues sans aucun retour-arrière). L'exploitation des méthodes structurelles comme BTD ou ses variantes pour résoudre de telles instances n'est pas nécessairement pertinente. Ainsi, nous nous basons ici sur le nombre de nœuds développés par $MAC{+}RST$ comme critère de difficulté. Une instance est considérée comme difficile si le nombre de nœuds développés par $MAC{+}RST$ est plus grand que $100 \times n$. Ce faisant, nous disposons de 675 instances considérées comme difficiles. La figure 4.11 fournit la comparaison de temps de résolution pour $BTD\text{-}MAC{+}RST{+}Fusion$ et $MAC{+}RST$ sur ces instances. Globalement, nous observons que les deux algorithmes $BTD\text{-}MAC{+}RST{+}Fusion$ et $MAC{+}RST$ ont un comportement similaire sur une grande partie de ces instances. En effet, pour 66% des instances, la différence du temps de résolution entre les deux méthodes est inférieure à 10%. Cependant, pour le reste des instances, $BTD\text{-}MAC{+}RST{+}Fusion$ est généralement plus rapide que $MAC{+}RST$. D'ailleurs, $BTD\text{-}MAC{+}RST{+}Fusion$ cumule un temps de résolution de 18 741 s sur ces instances tandis que $MAC{+}RST$ requiert 35 116 s. Pour 16% des instances, $BTD\text{-}MAC{+}RST{+}Fusion$ est au moins 10 fois plus rapide que $MAC{+}RST$ tandis que ce dernier est 10 fois plus rapide pour seulement 1% des instances. Finalement, l'exploita-

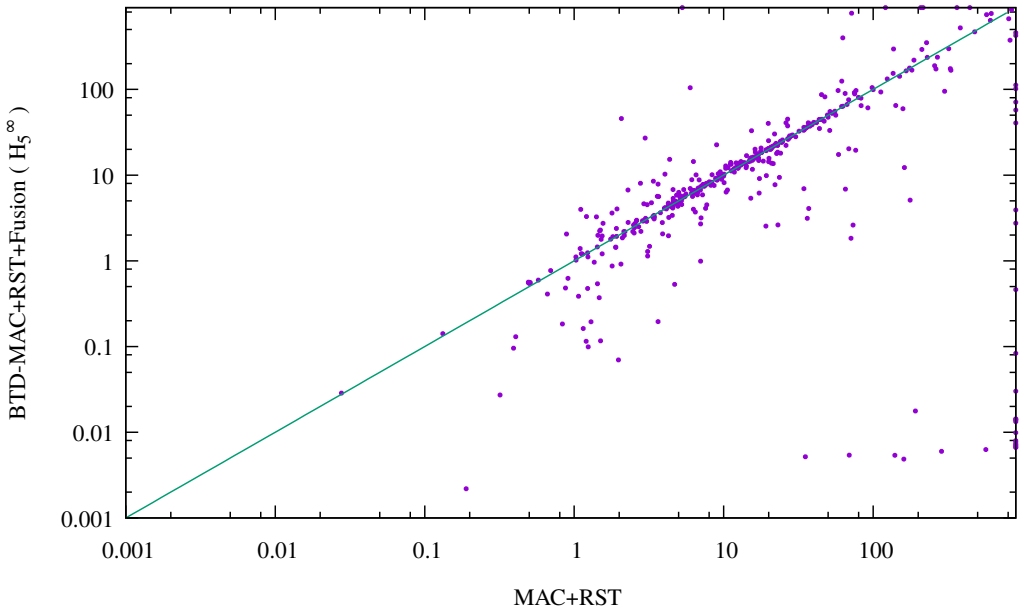

FIGURE 4.11 – La comparaison des temps de résolution de $BTD\text{-}MAC{+}RST{+}Fusion$ et de $MAC{+}RST$ pour les 675 instances.

tion de la structure joue un rôle central. En effet, nous pouvons noter que 100% (respectivement 67%) des instances non résolues par $MAC{+}RST$ mais résolues par $BTD\text{-}MAC{+}RST{+}Fusion$ ont un ratio $n/(w^+ + 1)$ plus grand que 2 (respectivement 5).

Récemment, nous avons participé à la compétition XCSP3 2017 [XC1, 2017]. Nous avons participé au track standard (solveur BTD) et au track mini-solver (solveur miniBTD) pour la résolution séquentielle d'instances CSP. Le temps de résolution par instance est limité à 40 minutes et l'utilisation mémoire à 15500 Mio. Nous exploitons la décomposition $H_5\text{-}TD$, l'heuristique $dom/wdeg$, des redémarrages géométriques avec un seuil initial de 100 retours en arrière et un facteur de 1.1. Le premier cluster racine choisi est celui ayant le ratio maximum du nombre de contraintes par rapport à sa taille moins un. Ensuite, lors de chaque redémarrage, le cluster racine choisi est celui qui maximise la somme des poids des contraintes dont la portée intersecte ce dernier. Finalement, la cohérence d'arc est maintenue en prétraitement via $AC\text{-}3^{rm}$ et pendant la résolution via $AC\text{-}8^{rm}$. Les instances choisies pour le track standard contiennent des contraintes globales non manipulées par notre solveur. Nous avons tout de même réussi à résoudre 47% des 510 instances considérées et 57% des instances résolues par le VBS. Au niveau du track mini-solver, nous avons occupé le deuxième rang et nous avons pu résoudre 67% des 242 instances considérées et 86% des instances du VBS. Le solveur qui a occupé le premier rang résout 95% des instances du VBS.

Bilan En conclusion, nous avons montré l'intérêt pratique de l'exploitation dynamique de la décomposition vis-à-vis de son exploitation statique standard. Nous avons aussi montré que l'exploitation simultanée des redémarrages et de la fusion dynamique augmente l'efficacité des algorithmes bénéficiant des redémarrages. Ainsi, l'intérêt de la fusion dynamique s'ajoute à celui des redémarrages. En outre, nous avons comparé la fusion dynamique à la fusion statique réalisée en amont de la résolution une fois la décomposition calculée. La fusion dynamique met en avant la pertinence des critères sémantiques qu'elle

intègre par rapport à la fusion statique qui se base uniquement sur des critères structurels. Cet aspect est notamment souligné lorsque les redémarrages ne sont pas exploités. En effet, ces techniques compensent dans certains cas l'absence de la fusion dynamique. Finalement, nous avons comparé $BTD\text{-}MAC{+}RST{+}Fusion$ avec H_5^∞, qui permet d'obtenir les meilleurs résultats, à $MAC{+}RST$. $BTD\text{-}MAC{+}RST{+}Fusion$ s'est principalement démarqué sur les instances difficiles. D'ailleurs, il ne faut pas oublier que l'exploitation de la fusion dynamique met en place des techniques sophistiquées dont l'utilisation n'a pas de sens dans le cas où les instances sont faciles, voire triviales.

4.5 Conclusion

Les décompositions arborescentes ont déjà été exploitées avec succès pour résoudre des instances du problème CSP. Pour autant, leur potentiel n'a pas été pleinement exploité. L'amélioration de la qualité des décompositions calculées grâce au cadre général de calcul de décomposition $H\text{-}TD$ (cf. chapitre 3) a permis d'augmenter l'efficacité de la résolution. En effet, ce cadre offre l'opportunité d'intégrer à la décomposition calculée plusieurs critères qui semblent avoir plus d'intérêt à l'égard de la résolution que le paramètre classique qui est la largeur de la décomposition. Ainsi, l'exploitation de ce cadre a permis de calculer, par exemple, des décompositions ayant des clusters connexes, des clusters avec plusieurs fils et aussi des décompositions avec des séparateurs de taille bornée. Ce dernier semble surpasser par son intérêt les autres paramètres vis-à-vis de la résolution des instances CSP. Cependant, cette décomposition est calculée en amont de la résolution et sera utilisée tout au long de la résolution. Elle est calculée en se basant uniquement sur des critères structurels ce qui ne garantit pas que cette décomposition soit la plus appropriée à l'égard du contexte de la résolution. En effet, cette décomposition ne permet pas d'intégrer des éléments relevant de la sémantique du problème.

Quant aux méthodes non structurelles comme MAC, elles profitent librement des approches adaptatives. Ces approches sont connues pour être capables de s'adapter au contexte de la résolution en faisant des choix qui prennent en considération l'état actuel de la résolution mais aussi ses états précédents. En intégrant ces approches, les méthodes de résolution ont témoigné une avancée remarquable. Ainsi, les heuristiques de choix de variables utilisées sont désormais le plus souvent adaptatives. Par exemple, en utilisant l'heuristique de choix de variables $dom/wdeg$, la recherche est dirigée vers les parties les plus difficiles et les plus conflictuelles du problème. Dans le même esprit, la recherche a pu être guidée suivant l'impact de la variable ou son activité. Les méthodes de résolution comme MAC peuvent profiter pleinement de ces méthodes vu qu'il n'y a aucune restriction imposée sur l'heuristique de choix de variables. D'ailleurs, celle-ci peut potentiellement choisir consécutivement des variables qui ne semblent pas avoir de lien structurel et qui sont relativement éloignées dans l'hypergraphe de contraintes. Au contraire, les méthodes structurelles sont désavantagées sur ce point. En effet, la décomposition sur laquelle elle se base induit un ordre partiel de choix de variables. C'est ainsi que ces méthodes ne peuvent pas se laisser complètement guider par une heuristique adaptative tel que $dom/wdeg$. En conséquence, les méthodes à base d'une décomposition voient leur efficacité se dégrader par rapport aux méthodes non structurelles.

Nous avons alors proposé dans ce chapitre de relâcher les contraintes imposées par la décomposition pour permettre plus de liberté aux méthodes structurelles. L'idée est de pouvoir changer de décomposition (au sens de l'ensemble de clusters qui la constituent) pendant la résolution pour pouvoir, par voie de conséquence, changer l'ordre partiel imposé par la décomposition. Le changement de décomposition ne se fait pas d'une

manière aléatoire mais plutôt d'une façon adaptative. En effet, nous nous inspirons des approches adaptatives pour guider le changement de la décomposition selon le contexte de la résolution. La modification de la décomposition se fait en se basant sur des informations recueillies pendant la résolution dans le but d'adapter la décomposition au contexte de celle-ci. Ainsi, nous proposons de modifier la décomposition via *la fusion* des clusters. Deux clusters fusionnés offrent plus de liberté quant aux choix de l'heuristique de choix de variables. Cette fusion se fait grâce aux recommandations de l'heuristique de choix de variables. La fusion dynamique permet aussi à un algorithme tel que BTD de mieux s'adapter aux instances peu structurées qui induisent souvent une « mauvaise » décomposition. En effet, ces instances sont généralement mieux résolues par une approche qui a toute la liberté quant au choix de la prochaine variable. L'utilisation de la fusion dynamique convergerait vers une décomposition qui permettrait de choisir plus librement la prochaine variable voire vers une décomposition ne contenant qu'un seul cluster.

La fusion dynamique de la décomposition a permis d'augmenter l'efficacité pratique de la résolution par rapport à son utilisation statique en termes de nombre d'instances résolues et du temps de résolution. En particulier, l'exploitation de la fusion dynamique avec la décomposition H_5^∞ a sensiblement amélioré la performance de $BTD\text{-}MAC{+}RST$. Nous avons aussi montré l'intérêt de la fusion dynamique par rapport à la fusion statique qui se fait avant la résolution et uniquement à la base de critères structurels. Non seulement, la fusion dynamique a amélioré l'exploitation de la décomposition mais a permis aussi de mettre en avant la compétitivité des méthodes structurelles vis-à-vis des méthodes non structurelles, voire les surpasser dans certains cas notamment pour les instances difficiles.

Dans le chapitre suivant, nous nous intéressons à une autre forme de dynamicité appliquée à la résolution des instances WCSP.