Exploitation dynamique de la décomposition dans le cas du problème WCSP

Sommaire

5.1 Introduction 188
5.2 Adaptation au contexte de la résolution dans le cadre WCSP . 188
5.3 Exploitation de la décomposition « si besoin » pour le problème WCSP : BTD-DFS+DYN 190
5.3.1 Décomposition si besoin 190
5.3.2 Description de l'algorithme BTD-DFS+DYN 192
5.3.2.1 Similitudes avec BTD-DFS 192
5.3.2.2 Modifications réalisées pour BTD-DFS+DYN 194
5.3.3 Fondements théoriques 195
5.4 Étude expérimentale 196
5.4.1 Protocole expérimental 196
5.4.2 Observations et analyse des résultats 197
5.5 Conclusion 204

5.1 Introduction

Nous nous sommes intéressés dans le chapitre précédent à l'amélioration de la prise en compte du contexte de la résolution tout en exploitant une décomposition arborescente pour résoudre des instances CSP. Nous avons montré qu'à travers la fusion dynamique des clusters, nous libérons davantage l'heuristique de choix de variables en nous laissant guider par cette dernière pour décider de la fusion ou non des clusters. Nous pouvons ainsi profiter plus intensément des approches adaptatives dont l'emploi est notoirement d'une très grande importance pour assurer une résolution efficace. La fusion dynamique a permis de modifier le comportement des algorithmes basés sur BTD en assurant un compromis entre un algorithme non structurel tel que MAC et un algorithme comme BTD. Plus précisément, selon les instances, le nouvel algorithme pourra avoir un comportement allant d'un simple $BTD\text{-}MAC$ jusqu'à MAC. Cette approche permettrait de faciliter l'emploi du solveur par des utilisateurs non experts. En effet, les solveurs auxquels nous aspirons ne requiert pas l'intervention de l'utilisateur pour sélectionner, par exemple, la décomposition convenable qui sera utilisée pendant la résolution. Cette tâche n'est pas aisée surtout que ce choix peut porter préjudice à l'efficacité du solveur. Notre approche va permettre d'avoir une décomposition choisie par défaut et qui va être remise en cause pendant la résolution. Le solveur peut éventuellement déduire que l'utilisation d'une décomposition n'est pas avantageuse. Dans le même esprit, nous proposons dans ce chapitre une approche différente basée également sur une exploitation dynamique de la décomposition pour la résolution d'instances WCSP. Elle vise à tirer profit à la fois des avantages des méthodes structurelles et des méthodes non structurelles. En d'autres termes, son objectif consiste à imiter les méthodes non structurelles au sens de la liberté du choix de la prochaine variable à affecter tout en bénéficiant des enregistrements réalisés par les méthodes structurelles. La méthode résultante pourrait avoir le comportement de l'un comme le comportement de l'autre ou aussi un comportement intermédiaire permettant de résoudre des instances ne pouvant être résolues ni par l'un, ni par l'autre. Afin de décider du comportement du nouvel algorithme, nous avons recours à une approche adaptative. Elle rassemble des constatations faites jusqu'à un instant t de la résolution et décide tout en se basant sur son état à cet instant du comportement ultérieur de l'algorithme. Nous proposons ainsi dans ce chapitre un nouvel algorithme basé sur $BTD\text{-}DFS$ (cf. la partie 2.4.4.2) qui exploite la décomposition d'une façon plus avancée vis-à-vis de son exploitation classique. Cet algorithme sera décrit en détails dans la section 5.3. Son intérêt pratique est mis en avant dans la section 5.4 avant de conclure.

Tout d'abord, nous nous focalisons dans la section suivante sur les objectifs de la gestion dynamique de la décomposition dans le cadre WCSP.

5.2 Adaptation au contexte de la résolution dans le cadre WCSP

Nous présentons dans cette partie les objectifs de l'utilisation dynamique de la décomposition pour la résolution d'instances WCSP.

Tout d'abord, l'utilisation dynamique de la décomposition pour la résolution d'instances WCSP rejoint le but de la fusion dynamique employée dans le cadre CSP qui consiste à améliorer l'exploitation des approches adaptatives dans les méthodes structurelles. D'ailleurs, l'emploi d'une décomposition arborescente pour la résolution des instances WCSP pose le même problème que son exploitation pour la résolution des instances CSP en termes de liberté de l'heuristique de choix de variables. En permettant des ordres

partiels non compatibles avec la décomposition d'origine, tout en étant moins contraignants, nous augmentons la chance de l'algorithme de trouver un ordre de choix de variables plus opportun à l'égard de l'instance à résoudre. Par exemple, en ayant le maximum de liberté, une heuristique de choix de variables telle que $dom/wdeg$ est plus susceptible de localiser plus rapidement les parties les plus difficiles de l'espace de recherche.

En plus de leur association avec les approches adaptatives, les algorithmes de résolution des instances WCSP ont connu une amélioration remarquable au niveau de leur stratégie de recherche. Plus précisément, avec $HBFS$, la combinaison des deux stratégies de recherche, à savoir le parcours en profondeur (DFS) et le parcours du meilleur d'abord (BFS) a amélioré significativement la résolution des problèmes d'optimisation sous contraintes [Allouche et al., 2015] (cf. la partie 2.4.4.1). Naturellement, si les limitations imposées par la décomposition au niveau de l'heuristique de choix de variables sont drastiques, l'efficacité pratique de cette hybridation peut s'en retrouver détériorée. Ainsi, relâcher les contraintes imposées par la décomposition offre l'opportunité d'avoir une plus grande sélection de variables lors du choix de la prochaine variable à instancier. Par conséquent, une variable de meilleure borne inférieure peut être trouvée lorsque BFS est exploité.

Ensuite, l'utilisation dynamique de la décomposition pour la résolution d'instances WCSP vise à capturer des instances résolues facilement, voire trivialement, par des méthodes non structurelles. En effet, selon la nature de l'instance à résoudre, il se peut que l'instance soit résolue en quelques fractions de secondes par une méthode comme DFS ou $HBFS$ tandis qu'elle nécessiterait des heures d'exécution avant d'être résolue par une méthode basée sur BTD. Pour y parvenir, la méthode proposée devrait pouvoir supprimer dans certains cas toute contrainte imposée sur l'heuristique de choix de variables afin de lui permettre d'acquérir une liberté totale.

L'exploitation dynamique de la décomposition vise également à permettre la résolution de certaines instances plus difficiles, habituellement résolues par une exploitation classique de la décomposition. Plus précisément, elle vise à exploiter les indépendances détectées entre les sous-problèmes ainsi que les enregistrements pouvant être réalisés. Afin d'atteindre cet objectif, la méthode proposée devrait pouvoir se comporter dans certains cas comme une méthode classique basée sur BTD.

La combinaison des avantages des méthodes structurelles et non structurelles a aussi pour objectif de résoudre de nouvelles instances en exploitant conjointement les avantages des méthodes non structurelles et ceux des méthodes structurelles. À cette fin, l'exploitation dynamique de la décomposition devrait permettre d'exploiter la décomposition d'une façon plus légère que son exploitation classique sans toutefois perdre tout l'intérêt des méthodes à base de décomposition.

En outre, au vu de l'efficacité des approches adaptatives et de la grande variété d'instances ciblées par l'exploitation dynamique de la décomposition, cette dernière vise à s'adapter à la nature de l'instance afin que la décomposition couramment employée soit la plus opportune possible. Il peut s'agir de la décomposition d'origine, d'une décomposition formée d'un seul cluster ou d'une décomposition intermédiaire. Son but est alors de se baser sur l'historique de l'instance et sur son état actuel afin de décider quelle décomposition utiliser.

Finalement, l'utilisation dynamique de la décomposition permet d'attribuer moins d'importance à la décomposition originale calculée. En effet, dans le chapitre 3, nous nous sommes intéressés à la qualité de la décomposition calculée vis-à-vis de la résolution. Nous avons aussi vu que selon la décomposition utilisée pendant la résolution, cette dernière peut être plus ou moins efficace. Permettre de se libérer de cette décomposition initialement calculée atténue les inconvénients de celle-ci s'il s'avère qu'elle est à l'origine de la

détérioration de l'efficacité de la résolution.

5.3 Exploitation de la décomposition « si besoin » pour le problème WCSP : BTD-DFS+DYN

Dans le même esprit des travaux réalisés pour le problème CSP dans le chapitre 4, nous visons dans cette partie à exploiter le concept de dynamicité de la décomposition pour le problème WCSP.

Comme évoqué dans la partie précédente, nous visons à profiter de l'efficacité des heuristiques de choix de variables adaptatives comme l'heuristique *dom/wdeg* et du progrès réalisé par les algorithmes de recherche à parcours hybride comme $HBFS$. Ainsi, nous tentons de libérer les méthodes exploitant une décomposition. En revanche, l'utilisation d'une décomposition peut être parfois très bénéfique pour la résolution des instances WCSP. En particulier, l'enregistrement des informations au niveau des séparateurs entre clusters permet d'élaguer des parties de l'espace de recherche et de rendre la résolution plus efficace. Ces enregistrements permettent de mémoriser, pour chaque sous-problème considéré, les meilleures bornes inférieures et supérieures calculées, ou mieux encore, son optimum.

La question qui se pose légitimement est alors de savoir comment exploiter conjointement les avancées dont ont bénéficié les algorithmes de recherche au niveau des heuristiques de choix de variables et du type de parcours pour le problème WCSP et les avantages offerts par la décomposition arborescente. Afin de concilier ces deux points de vue, nous proposons dans cette partie d'exploiter la décomposition dynamiquement d'une façon plus flexible et plus appropriée et adaptée 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 consiste à utiliser la décomposition correspondant à un sous-problème uniquement lorsque la recherche sans décomposition semble stagner. La décomposition arborescente est alors exploitée d'une façon plus adéquate en évitant de l'utiliser systématiquement que ce soit globalement sur tout le problème ou localement sur un sous-problème. Ainsi, nous proposons dans ce qui suit l'algorithme $BTD\text{-}DFS{+}DYN$ qui intègre ce type de fonctionnement.

Il est très important de noter qu'au vu de l'efficacité pratique de $BTD\text{-}HBFS$ par rapport à celle de $BTD\text{-}DFS$ [Allouche et al., 2015], nous sommes naturellement plus intéréssés par $BTD\text{-}HBFS{+}DYN$ que par $BTD\text{-}DFS{+}DYN$. Toutefois, nous présentons dans cette partie $BTD\text{-}DFS{+}DYN$ par souci de simplicité. Cependant, l'extension de $BTD\text{-}HBFS$ peut être déduite sans difficulté de celle de $BTD\text{-}DFS$.

L'algorithme $BTD\text{-}DFS{+}DYN$ (voir l'algorithme 5.1) se base sur l'algorithme $BTD\text{-}DFS$. 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 E_r. En ce qui concerne $BTD\text{-}DFS$, la décomposition induit classiquement un ordre partiel sur les variables comme tous les algorithmes basés sur BTD et comme nous l'avons déjà vu dans la partie 4.2.2. Cependant, $BTD\text{-}DFS{+}DYN$ exploite la décomposition différemment.

5.3.1 Décomposition si besoin

$BTD\text{-}DFS{+}DYN$ a pour objectif d'exploiter la décomposition à bon escient, c'est-à-dire quand le besoin s'en fait sentir. Plus précisément, la décomposition calculée initialement ne sera exploitée, pour un sous-problème donné, que si la résolution sans décom-

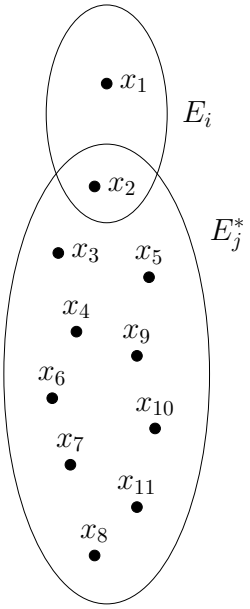


FIGURE 5.1 – Ensemble de clusters de la décomposition de la figure 4.1 lorsque E_j est fusionné avec ses descendants (cluster E_j^*).

position est jugée inefficace. Prenons l'exemple de la décomposition de la figure 4.1 du chapitre 4 représentée par l'ensemble de clusters $E = \{E_i, E_j, E_k, E_l, E_m, E_n\}$ enracinée en E_i. Soit E_j le cluster courant et $\mathcal{A}$ l'affectation courante. Nous nous intéressons au sous-problème $P_j|\mathcal{A}$ enraciné en E_j induit par l'affectation $\mathcal{A}$ du séparateur $E_i \cap E_j$ 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. D'où, formellement $E_j^* = \cup_{E_p \in Desc(E_j)} E_p$. Cette fusion consiste en une mise en commun de toutes les variables des clusters de la descendance de E_j (voir figure 5.1). Ainsi E_j^* contient toutes les variables des clusters descendants de E_j, E_j inclus. En ce qui concerne l'heuristique de choix de variables, elle acquiert une liberté totale quant au choix des variables suivantes sur E_j^*. Plus précisément, l'ordre partiel induit par la décomposition est : $\Lambda = [\{x_1, x_2\}, \{x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}]$.
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 et ne sera pas exploitée.
- 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 les clusters de sa descendance formant le cluster E_k^* comme le montre la figure 5.2. L'ordre partiel ainsi induit par la décomposition est alors :
  $\Lambda = [\{x_1, x_2\}, \{x_3, x_4, x_5\}, \{x_6, \{x_7, x_8, x_9, x_{10}, x_{11}\}\}]$

Quant aux clusters feuilles (n'ayant pas de fils comme E_n), $BTD\text{-}DFS{+}DYN$ se comporte comme $BTD\text{-}DFS$. À noter que ce raisonnement est répété 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, $BTD\text{-}DFS{+}DYN$ se comporte comme l'algorithme

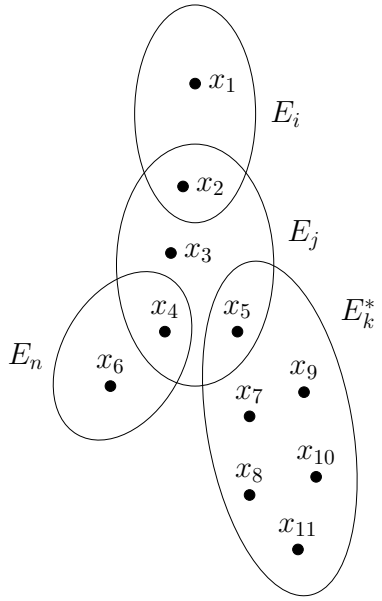


FIGURE 5.2 – Ensemble de clusters de la décomposition de la figure 4.1 lorsque E_j est exploité et E_k fusionné avec ses descendants (cluster E_k^*).

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 $BTD\text{-}DFS{+}DYN$ décide d'exploiter le cluster lui-même plutôt que sa descendance.

5.3.2 Description de l'algorithme BTD-DFS+DYN

Nous décrivons dans cette sous-section l'algorithme $BTD\text{-}DFS{+}DYN$ en mettant en avant les similitudes avec $BTD\text{-}DFS$ ainsi que les modifications qui y sont apportées.

5.3.2.1 Similitudes avec BTD-DFS

L'algorithme $BTD\text{-}DFS{+}DYN$ prend en paramètres l'affectation courante $\mathcal{A}$, le cluster courant E_i, l'ensemble V_{E_i} des variables non affectées de E_i, l'ensemble V_{desc_i} des variables non affectées de la descendance $Desc(E_i)$ de E_i 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 $\mathcal{A}$. Lorsque l'optimum est calculé, la valeur de *cub* retournée correspond à ce dernier. Par souci de clarté, le paramètre *cub* donné en entrée sera noté cub^e et celui en sortie sera désigné par cub^s. Deux cas sont possibles :

- Si la valeur de cub^s est strictement inférieure à celle de cub^e, alors cub^s est l'optimum correspondant à ce sous-problème.
- Sinon, cub^s définit une borne inférieure de l'optimum.

L'appel initial est $BTD\text{-}DFS{+}DYN(\emptyset, E_r, V_{E_r}, V_{desc_r}, lb(P_r|\emptyset), k)$ avec V_{desc_r} 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. Rappelons que tout au long de l'algorithme, $w_\emptyset^i$ désigne la borne inférieure localisée relative au cluster E_i. Comme $BTD\text{-}DFS$, $BTD\text{-}DFS{+}DYN$ suppose que le problème donné en entrée

Algorithme 5.1 : BTD-DFS+DYN $(\mathcal{A}, E_i, V_{E_i}, V_{desc_i}, clb, cub)$

Entrées : L'affectation courante $\mathcal{A}$, le cluster courant E_i, la borne inférieure clb
Entrées-Sorties : L'ensemble V_{E_i} des variables non instanciées de E_i, l'ensemble V_{desc_i} des variables non instanciées de $V_{Desc(E_i)}$, la borne supérieure courante cub

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

est cohérent selon la cohérence locale utilisée et renvoie son optimum. Les lignes 5 à 17 de $BTD\text{-}DFS{+}DYN$ visent à instancier les variables de V' comme le ferait $BTD\text{-}DFS$ pour les variables de V_{E_i}. Un couple (variable, valeur), (x, v), 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 $x = v$ (ligne 9) et $x \neq v$ (ligne 14). Pour chaque branche, la cohérence locale est appliquée au sous-problème enra-

ciné en E_i et induit par l'affectation courante permettant d'obtenir une borne inférieure $lb(P_i|\mathcal{A} \cup \{(x = v)\})$ si une décision positive est faite et $lb(P_i|\mathcal{A} \cup \{(x \neq v)\})$ sinon (cf. la partie 2.4.4.2 et l'algorithme $Lc\text{-}BTD^+$ pour de plus amples détails). Si le maximum clb' entre la borne inférieure $lb(P_i|\mathcal{A} \cup \{(x = v)\})$ (ou $lb(P_i|\mathcal{A} \cup \{(x \neq v)\})$ lorsqu'il s'agit d'une décision négative), 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 $BTD\text{-}DFS{+}DYN$ résolvent les sous-problèmes enracinés en chaque cluster fils E_j de E_i de la même façon que $BTD\text{-}DFS$. Le sous-problème $P_j|\mathcal{A}$ n'est exploré que si son optimum n'est pas déjà connu. D'ailleurs, $BTD\text{-}DFS$ et $BTD\text{-}DFS{+}DYN$ mémorisent pour chaque sous-problème $P_j|\mathcal{A}$ deux valeurs, notées $LB_{P_j|\mathcal{A}}$ et $UB_{P_j|\mathcal{A}}$, représentant 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 [Givry et al., 2006]. Plus précisément, bien que l'utilisation d'un majorant initial trivial (k dans ce cas), garantisse effectivement que l'optimum de $P_j|\mathcal{A}$ sera correctement calculé, cela ne permet pas d'élaguer efficacement l'espace de recherche en n'ayant pas une coupe initiale efficace. En outre, en résolvant le sous-problème $P_j|\mathcal{A}$ indépendamment des autres sous-problèmes, nous pourrons détériorer l'efficacité de la résolution. En effet, l'optimum trouvé pour $P_j|\mathcal{A}$ additionné aux coûts des clusters déjà affectés et aux bornes inférieures calculées pour d'autres, peut éventuellement largement dépasser la borne supérieure courante et ne pas ainsi être capable de participer à une solution globale. Ainsi, $P_j|\mathcal{A}$ exploite une borne supérieure non triviale qui est le minimum entre la meilleure borne supérieure enregistrée pour ce problème $UB_{P_j|\mathcal{A}}$ et une deuxième borne supérieure déduite de la différence entre la borne supérieure courante *cub* et le minorant de $P_i|\mathcal{A}$ duquel nous retranchons le minorant de $P_j|\mathcal{A}$. En exploitant cette nouvelle borne supérieure, si une solution de coût strictement inférieur à cette borne est trouvée, l'optimum est ainsi trouvé. Sinon, nous pouvons uniquement déduire une borne inférieure de l'optimum. L'appel récursif est suivi par une mise à jour de $LB_{P_j|\mathcal{A}}$ et de $UB_{P_j|\mathcal{A}}$ (ligne 26). En effet, si à la ligne 25 l'optimum est trouvé, les valeurs de $LB_{P_j|\mathcal{A}}$ et $UB_{P_j|\mathcal{A}}$ seront toutes les deux égales à l'optimum. Sinon, $LB_{P_j|\mathcal{A}}$ sera mis à jour en fonction de la nouvelle borne inférieure trouvée. L'exploration de tous les clusters fils permet enfin de mettre à jour *cub* (ligne 27). Cette mise à jour se fait en prenant le minimum entre la borne supérieure courante *cub* et celle déduite de la somme du coût de l'affectation courante du cluster E_i représenté par $w^i_\emptyset$ et des meilleures bornes supérieures $UB_{P_j|\mathcal{A}}$ trouvées pour chaque sous-problème $P_j|\mathcal{A}$.

5.3.2.2 Modifications réalisées pour BTD-DFS+DYN

Les similitudes entre $BTD\text{-}DFS$ et $BTD\text{-}DFS{+}DYN$ 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 le cluster 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_{E_i}. L'un des deux paramètres V_{E_i} ou V_{desc_i} est effectivement utilisé selon que nous exploitons E_i ou E^*_i. Le choix entre V_{E_i} et V_{desc_i} est fait dans les lignes 1 à 4 et est retenu dans V'. Ces ensembles sont mis à jour convenablement à la ligne 7. Si $V' = \emptyset$ et que E_i n'est pas fusionné avec sa descendance, $BTD\text{-}DFS{+}DYN$ se comporte comme $BTD\text{-}DFS$ (lignes 20-27). Si $V' = \emptyset$ et que E_i est fusionné avec sa descendance, $BTD\text{-}DFS{+}DYN$ n'a plus de variables à instancier vu que toutes les variables de la descendance de E_i sont déjà affectées. Dans ce cas, $BTD\text{-}DFS{+}DYN$ met à jour *cub* (ligne 29) en se référant à la

borne inférieure localisée $w_\emptyset^j$ de chaque cluster E_j appartenant à $Desc(E_i)$. En effet, la nouvelle borne supérieure *cub* est alors le maximum entre la borne supérieure courante *cub* et la somme des coûts des affectations des clusters de la descendance de E_i, $Desc(E_i)$. Si $V' \neq \emptyset$ (lignes 5-17), $BTD\text{-}DFS{+}DYN$ se comporte de la même façon indépendamment du choix de l'exploitation de E_i ou de E_i^* en essayant d'instancier les variables de V'.

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

5.3.3 Fondements théoriques

Nous nous intéressons à présent à la validité de $BTD\text{-}DFS{+}DYN$. La clé de la validité de $BTD\text{-}DFS{+}DYN$ réside dans la validité des bornes supérieure et inférieure enregistrées pour chaque sous-problème. En effet, pour un cluster E_i, quels que soit les clusters inclus dans $Desc(E_i)$, $LB_{P_i|\mathcal{A}}$ et $UB_{P_i|\mathcal{A}}$ resterons valides puisque le sous-problème $P_i|\mathcal{A}$ ne change pas. Nous pouvons alors énoncer le théorème suivant :

Théorème 11 *$BTD\text{-}DFS{+}DYN$ est correct, complet et termine.*

Preuve : La correction, la complétude et la terminaison de $BTD\text{-}DFS{+}DYN$ se base sur la correction, la complétude et la terminaison de $BTD\text{-}DFS$ [Givry et al., 2006]. Soit (E, T) la décomposition de base enracinée en E_r pouvant être exploitée par $BTD\text{-}DFS{+}DYN$.

Si tout au long de la résolution, la décomposition effectivement employée est celle contenant un seul cluster (le cas où le seul cluster utilisé de la décomposition est E_r^*) la validité de $BTD\text{-}DFS{+}DYN$ est trivialement garantie grâce à la validité de DFS.

Supposons maintenant que la décomposition employée à un instant donné contient au moins deux clusters, E_i et E_j^*. Autrement dit, pour l'affectation $\mathcal{A}$ du séparateur $E_i \cap E_j$, $BTD\text{-}DFS{+}DYN$ commence à exploiter le cluster E_j^*. Tant que pour l'affectation $\mathcal{A}[E_i \cap E_j]$ $BTD\text{-}DFS{+}DYN$ exploite E_j^*, $BTD\text{-}DFS{+}DYN$ se comporte comme $BTD\text{-}DFS$ sur la base d'une décomposition en deux clusters E_i et E_j^*. De ce fait, comme $BTD\text{-}DFS$ est correct, complet et termine, $BTD\text{-}DFS{+}DYN$ l'est aussi. $BTD\text{-}DFS{+}DYN$ peut éventuellement trouver l'optimum de $P_j|\mathcal{A}$ en exploitant E_j^* et l'enregistrer pour le séparateur $E_i \cap E_j$ qui est le même que celui de $E_i \cap E_j^*$.

Supposons maintenant que l'heuristique $Fusion$ décide d'exploiter le cluster E_j au lieu du cluster E_j^* avant que l'optimum ne soit trouvé. Les bornes inférieures et supérieures $LB_{P_j|\mathcal{A}}$ et $UB_{P_j|\mathcal{A}}$ calculées auparavant restent valides quelle que soit la décomposition employée pour le sous-problème P_j. Elles peuvent ainsi être utilisées en toute sécurité pour renseigner la recherche sur la nouvelle décomposition du problème P_j. Désormais, pour la même affectation $\mathcal{A}$ du séparateur $E_i \cap E_j$, $BTD\text{-}DFS{+}DYN$ exploitera définitivement le cluster E_j. De même, $BTD\text{-}DFS{+}DYN$ se comporte maintenant comme $BTD\text{-}DFS$ sur P_j à base d'une décomposition formée du cluster E_j et des clusters correspondants à la fusion de chaque cluster fils de E_j avec ses descendants. En répétant le même raisonnement à tous les niveaux de la décomposition et pour toutes les affectations d'un séparateur, nous en déduisons grâce à la validité de $BTD\text{-}DFS$, la validité de $BTD\text{-}DFS{+}DYN$. □

L'exploitation dynamique de la décomposition induit un changement au niveau des complexités en temps et en espace.

Théorème 12 *$BTD\text{-}DFS{+}DYN$ a une complexité temporelle en $O(exp(w^{*+}{+}1))$ et une complexité spatiale en $O(exp(s^*))$ avec $w^{*+}{+}1$ la taille du plus grand cluster effectivement exploité et s^* (avec $s^* \leq s$) la taille du plus grand séparateur effectivement utilisé.*

Preuve : Bien que $BTD\text{-}DFS{+}DYN$ se base initialement sur une décomposition précalculée en amont de la résolution, la décomposition n'est pas exploitée traditionnellement. En particulier, $BTD\text{-}DFS{+}DYN$ peut ne jamais exploiter la décomposition, l'exploiter partiellement ou aussi utiliser l'ensemble de tous ses clusters. La complexité temporelle dépend ainsi de la taille du plus grand cluster exploité $w^{*+}+1$ tandis que la complexité spatiale dépend de la taille du plus grand séparateur utilisé s^*. De ce fait, les complexités spatiales et temporelles dépendent de l'heuristique de $Fusion$ utilisée. □

Le comportement de $BTD\text{-}DFS{+}DYN$ pouvant aller d'un $BTD\text{-}DFS$ standard à un DFS classique, il en résulte que la complexité temporelle de $BTD\text{-}DFS{+}DYN$ 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 de la fusion des descendants d'un cluster pour tout cluster situé à un profondeur faible par rapport à la racine. En particulier, l'heuristique de fusion pourrait interdire l'exploitation du cluster E_r^* et ainsi empêcher $BTD\text{-}DFS{+}DYN$ de se comporter comme DFS. Au niveau de la complexité en espace, dans le pire des cas, la complexité en espace est la même que celle de $BTD\text{-}DFS$ si le plus grand séparateur de la décomposition est utilisé puisque $s^* \leq s$.

5.4 Étude expérimentale

Dans cette sous-section, nous évaluons la pertinence de l'exploitation dynamique de la décomposition. Nous évoquons tout d'abord le protocole expérimental.

5.4.1 Protocole expérimental

Nous reprenons, pour ces expérimentations, le protocole expérimental de la partie 3.4.3 du chapitre 3. Nous considérons les algorithmes $HBFS$, $BTD\text{-}HBFS$ et $BTD\text{-}HBFS{+}DYN$. Pour les deux premiers algorithmes, nous exploitons leurs implémentations fournies dans $Toulbar2$ [TOU, 2006]. Nous avons également implémenté l'algorithme $BTD\text{-}HBFS{+}DYN$ au sein de $Toulbar2$. Notre implémentation se base sur celle de $BTD\text{-}HBFS$ tout en apportant les modifications nécessaires. Pour tous les algorithmes, $\alpha_{hbfs} = 5\%, \beta_{hbfs} = 10\%$ et $N_{hbfs} = 10\ 000$ comme dans [Allouche et al., 2015]. L'efficacité pratique de $BTD\text{-}HBFS$ nous incite à le considérer au lieu de $BTD\text{-}DFS$. L'extension $BTD\text{-}HBFS{+}DYN$ peut être facilement déduite de $BTD\text{-}HBFS$ d'une façon similaire à la déduction de $BTD\text{-}DFS{+}DYN$ à partir de $BTD\text{-}DFS$. Concernant les décompositions, nous considérons de nouveau les décompositions $Min\text{-}Fill$ et $Min\text{-}Fill^4$ dont l'implémentation est fournie dans $Toulbar2$. En outre, nous considérons les décompositions H_2, H_3 et H_5 avec ses deux variantes H_5^{25} et $H_5^{5\%}$. La configuration de $Toulbar2$ est identique à celle décrite dans la partie 3.4.3 du chapitre 3. Les instances de I_3 sont toutes retenues. Les algorithmes de résolution ont un temps limite de 20 minutes incluant pour les algorithmes basés sur BTD le temps de calcul de la décomposition et 16 Go d'espace mémoire.

Heuristique de fusion $\mathcal{F}$ L'exploitation dynamique de la décomposition se base sur l'heuristique $\mathcal{F}$ de fusion qui est responsable de décider d'exploiter le cluster initial E_i

Algorithme	*Min-Fill*		*Min-Fill*^{4}	
	#rés.	temps	#rés.	temps
BTD-HBFS	1 712	26 291	1 995	91 232
BTD-HBFS+DYN	1 905	45 450	2 019	74 159

TABLE 5.1 – 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	1 946	76 018	1 989	57 348	2 006	58 826	2 028	58 043
BTD-HBFS+DYN	2 023	50 445	2 023	56 978	2 039	52 304	2 038	60 674

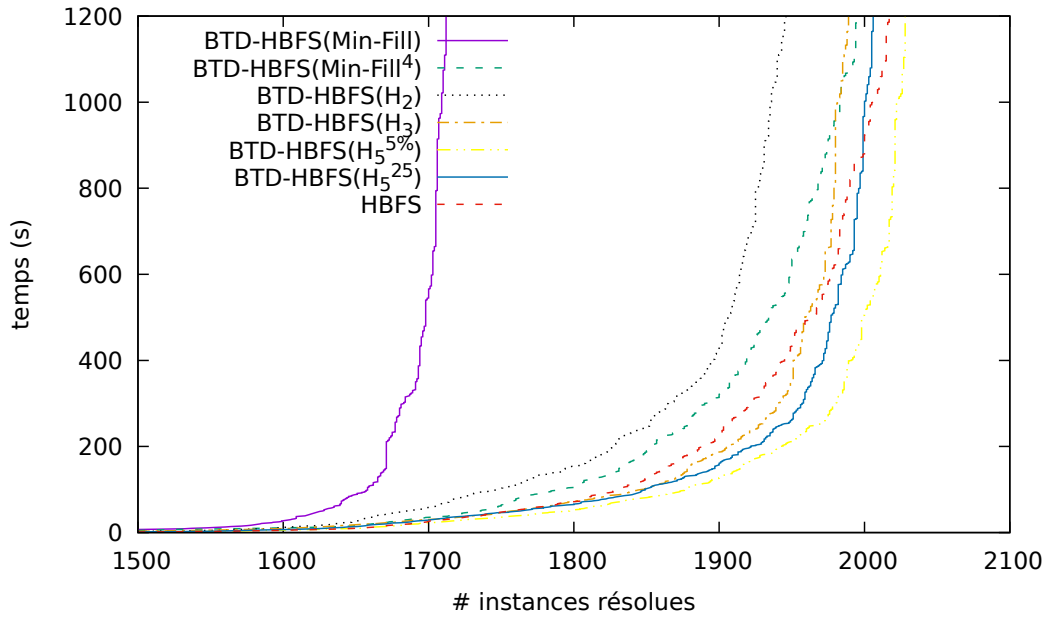
TABLE 5.2 – Nombre d'instances résolues et temps d'exécution en secondes pour *BTD-HBFS* et *BTD-HBFS+DYN* selon les décompositions de *H-TD*.

ou le cluster fusionné E_i^*. L'heuristique $\mathcal{F}$ se base sur le retour d'informations fait par *BTD-HBFS*. Elle exploite notamment son comportement *anytime* et les mises à jour permanentes des bornes inférieures et supérieures reportées pour chaque sous-problème. En effet, chaque appel à *BTD-HBFS* sur un sous-problème $P_i|\mathcal{A}$ prend en entrées une borne inférieure *clb* et une borne supérieure *cub*. Si, dans la limite du nombre de retours en arrière autorisé, *BTD-HBFS* ne réussit pas à améliorer une des 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, nous exploitons par la suite E_i en stoppant l'exploitation de E_i^*. Ce faisant, *BTD-HBFS+DYN* accumule des informations relatives aux états précédents de la résolution. Grâce à ces enregistrements, *BTD-HBFS+DYN* est capable de s'adapter au contexte de la résolution et aux particularités de l'instance à résoudre. La limite que nous choisissons dans nos expérimentations est 5. Ce choix est le fruit des expérimentations intensives qui ont permis d'évaluer l'intérêt des différentes valeurs pour cette limite. Une limite trop élevée s'avère contre-productive puisque *BTD-HBFS+DYN* se comporterait souvent comme *HBFS*. Au contraire, une limite inférieure à 5 empêcherait *BTD-HBFS+DYN* quand il le faut de bénéficier de la liberté totale offerte à l'heuristique de choix de variables sur un sous-problème. La limite de 5 est alors capable de donner de bons résultats en moyenne sur l'ensemble des instances du benchmark même si bien sûr, rien ne garantit que cette limite soit la plus adéquate pour chaque instance.

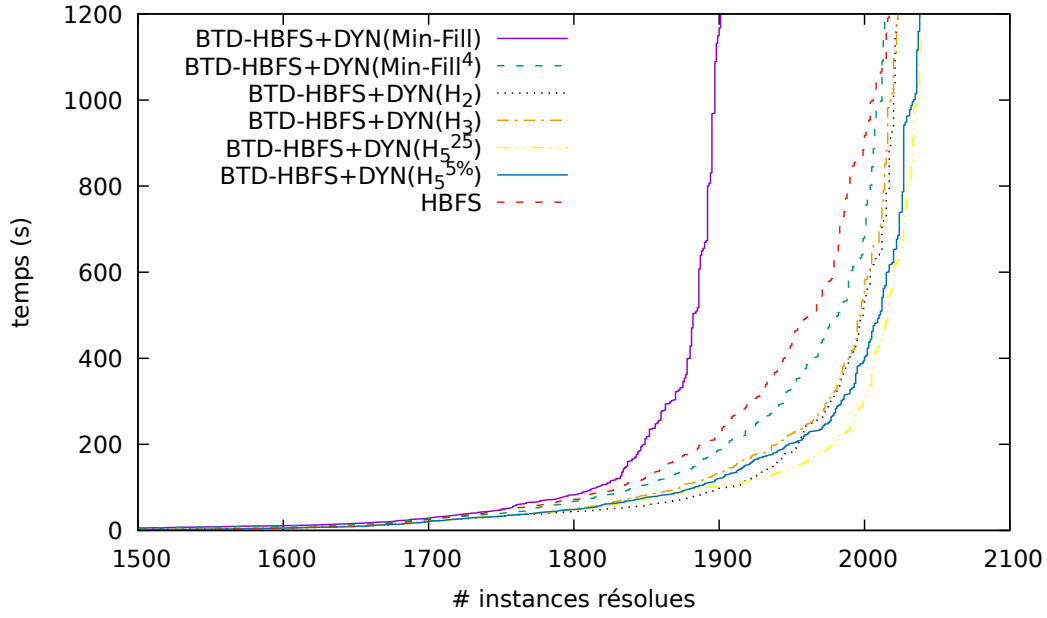
D'autres heuristiques peuvent être également proposées selon le besoin de l'utilisateur. Notons finalement que l'heuristique $\mathcal{F}$ proposée est fondée sur les retours d'informations de *BTD-HBFS*. L'utilisation d'un algorithme différent de *HBFS* nécessiterait un changement de l'heuristique de fusion exploitée.

5.4.2 Observations et analyse des résultats

Apport de l'utilisation si besoin de la décomposition par rapport à son utilisation traditionnelle Nous évaluons à présent *BTD-HBFS+DYN*. Les deux tables 5.1 et 5.2 ainsi que la figure 5.3 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 comme c'est le cas pour *Min-Fill* qui permet de résoudre 1 905 instances au lieu de 1 712 instances. L'utilisation de H_2 et H_3 avec *BTD-HBFS+DYN* permet de résoudre 2 023 instances contre 1 946 et 1 989 résolues par *BTD-HBFS*. L'exploitation de *Min-Fill*^{4} et H_5 est aussi améliorée avec



(a)



(b)

FIGURE 5.3 – Le nombre cumulative d'instances résolues pour $BTD\text{-}HBFS$ et $HBFS$ (a) et pour $BTD\text{-}HBFS{+}DYN$ et $HBFS$ (b) pour les instances de I_3.

$BTD\text{-}HBFS{+}DYN$ par rapport à $BTD\text{-}HBFS$. 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'algorithmes 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 $Min\text{-}Fill$, mais ce dernier point s'explique naturellement par le nombre d'instances supplémentaires résolues par $BTD\text{-}HBFS{+}DYN$ par rapport à $BTD\text{-}HBFS$. C'est ainsi que $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ résout 2 019 instances en 74 159 s tandis que $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ requiert 91 232 s pour résoudre 1 995

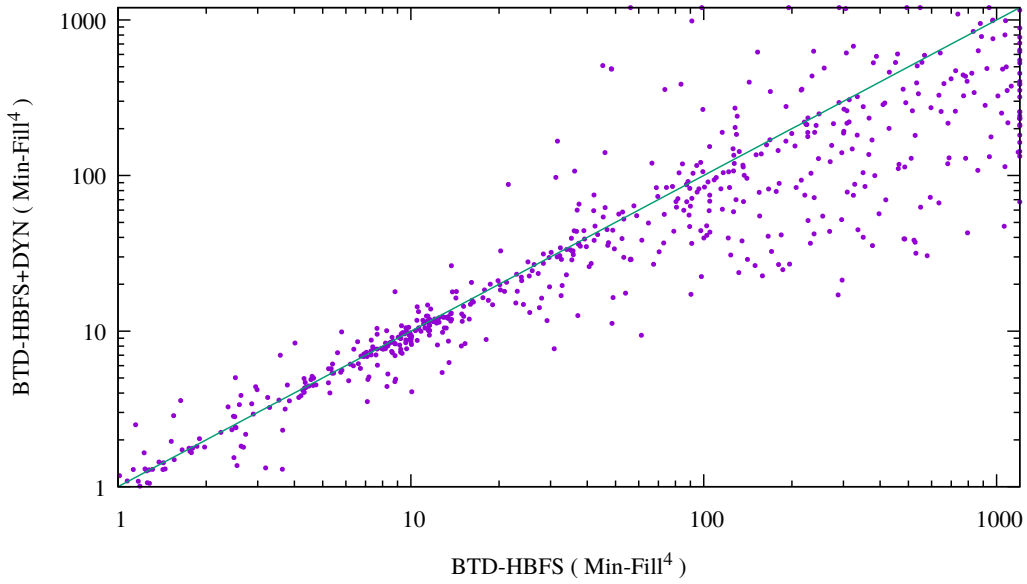


FIGURE 5.4 – Comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ pour les 2 444 instances.

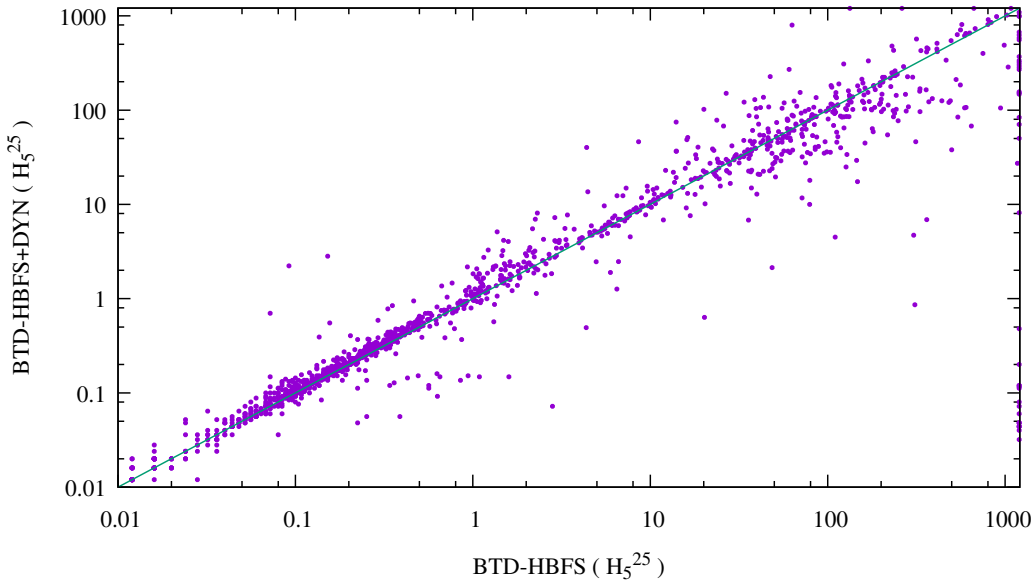


FIGURE 5.5 – Comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $BTD\text{-}HBFS$ avec H_5^{25} pour les 2 444 instances.

instances. La figure 5.4 compare les temps de d'exécution de $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^{r_4}$. Elle montre que l'efficacité de la résolution a sensiblement augmenté avec l'exploitation dynamique de la décomposition. Concernant H_5^{25}, $BTD\text{-}HBFS{+}DYN$ utilisant H_5^{25} requiert 52 304 s pour résoudre 2 039 instances tandis que $BTD\text{-}HBFS$ nécessite 58 826 s pour résoudre 2 006 instances. La figure 5.5 qui compare les temps de d'exécution $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $BTD\text{-}HBFS$ avec

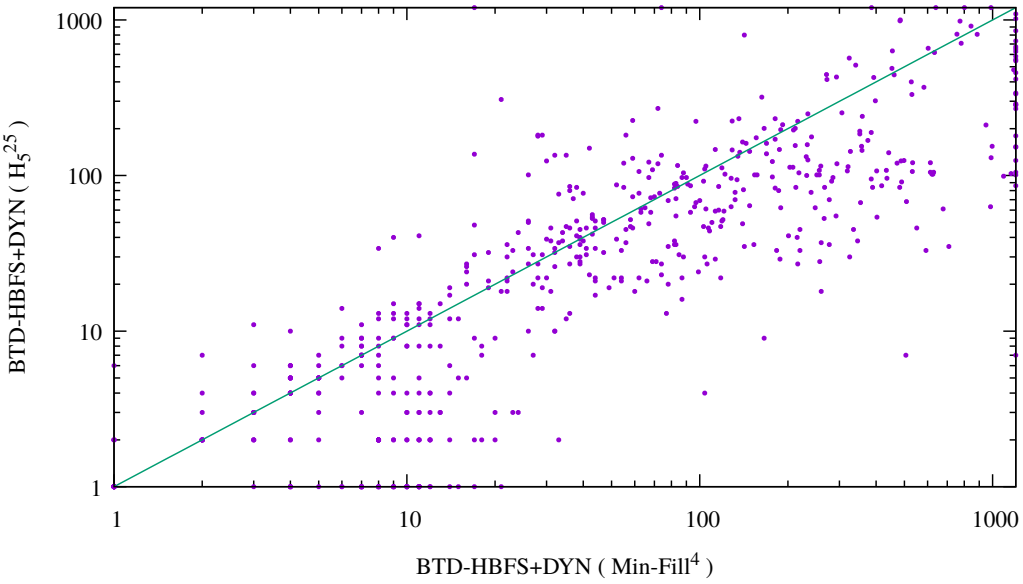


FIGURE 5.6 – Comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ pour les 2 444 instances.

H_5^{25} indique que les temps d'exécution sont plus souvent comparables que dans le cas de $Min\text{-}Fill^4$. Nous nous intéressons finalement à la comparaison de H_5^{25} à l'heuristique de l'état de l'art $Min\text{-}Fill^4$ avec $BTD\text{-}HBFS{+}DYN$. La figure 5.6 présente une comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$. $BTD\text{-}HBFS{+}DYN$ associé à H_5^{25} prouve clairement son intérêt pratique. Pour une comparaison plus équitable des temps cumulés d'exécution, nous nous appuions sur le benchmark des instances résolues à la fois par $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ et par $BTD\text{-}HBFS{+}DYN$ avec H_5^{25}. Il compte 2 013 instances résolues par $BTD\text{-}HBFS{+}DYN$ avec $Min\text{-}Fill^4$ en 71 249 s contre seulement 41 372 s par $BTD\text{-}HBFS{+}DYN$ avec H_5^{25}. Nous retenons pour la suite la décomposition $Min\text{-}Fill^4$ vu que son exploitation avec $BTD\text{-}HBFS$ est considérée comme la méthode structurelle de référence pour la résolution d'instances WCSP. Nous retenons également la décomposition H_5^{25} en raison de son intérêt vis-à-vis de la résolution avec $BTD\text{-}HBFS(+DYN)$.

$HBFS$ et $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ sont considérés les méthodes de l'état de l'art respectivement sans et avec exploitation de la structure.

BTD-HBFS+DYN avec H_5^{25} vs BTD-HBFS avec $Min\text{-}Fill^4$ Nous comparons tout d'abord l'algorithme $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ à $BTD\text{-}HBFS{+}DYN$ avec H_5^{25}. $BTD\text{-}HBFS{+}DYN$ surpasse significativement $BTD\text{-}HBFS$ en nombre d'instances et en temps de résolution. D'ailleurs, $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} résout 2 039 instances en 52 304 s tandis que $BTD\text{-}HBFS$ résout 1 995 instances avec $Min\text{-}Fill^4$ en 91 232 s. La figure 5.7 comparant les temps d'exécution des deux méthodes confirment cette tendance.

BTD-HBFS+DYN avec H_5^{25} vs HBFS Nous comparons à présent l'algorithme $BTD\text{-}HBFS{+}DYN$ basé sur H_5^{25} à $HBFS$. Nous pouvons noter que $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} a une meilleure performance que $HBFS$ (voir la figure 5.3). Plus précisément, $BTD\text{-}HBFS{+}DYN$ résout plus d'instances que $HBFS$ (2 039 instances contre 2 017 ins-

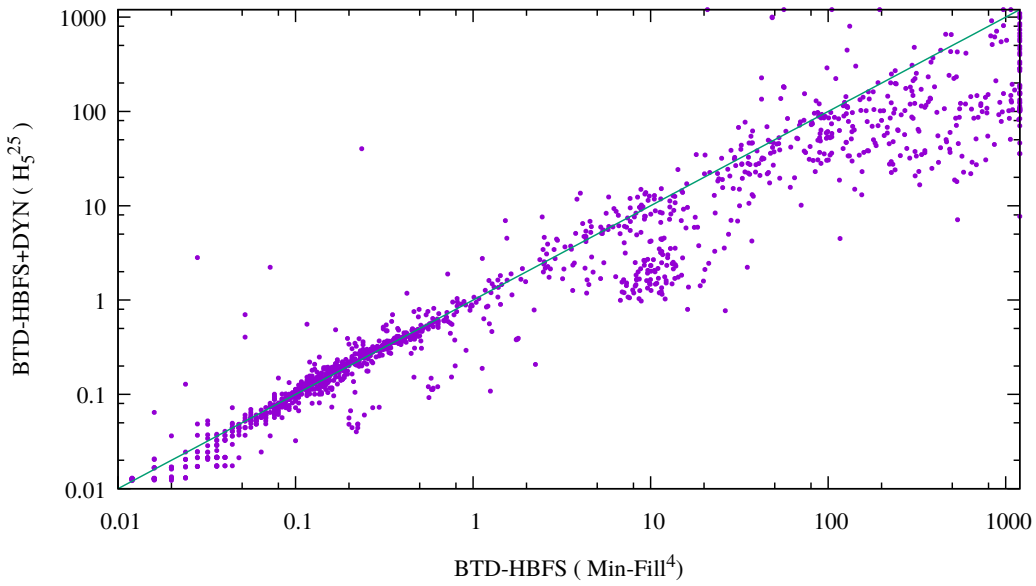


FIGURE 5.7 – Comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ pour les 2 444 instances.

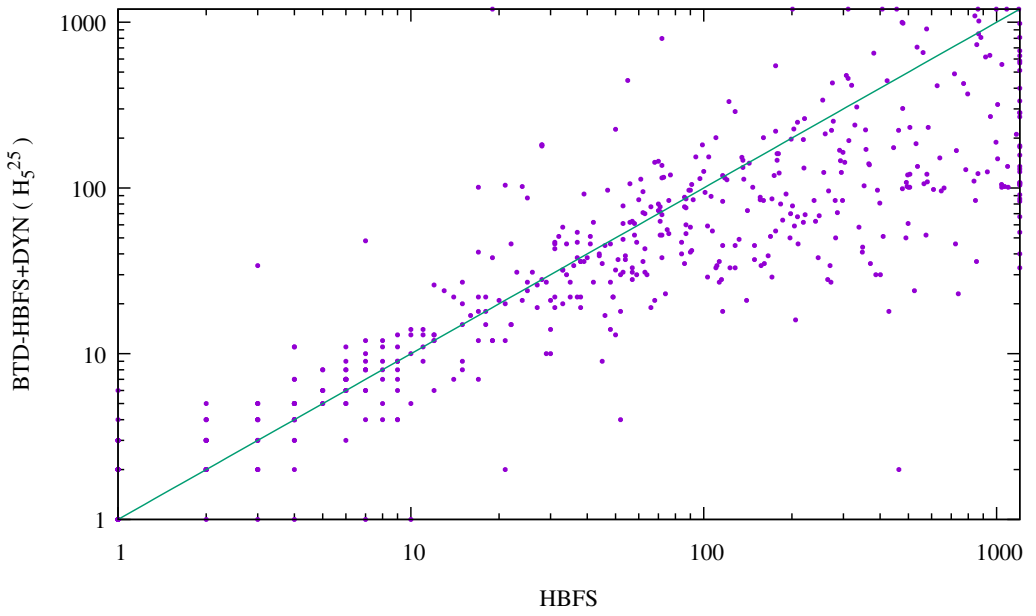


FIGURE 5.8 – Comparaison des temps d'exécution de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} à $HBFS$ pour les 2 444 instances.

tances). En addition, $BTD\text{-}HBFS{+}DYN$ a un meilleur temps de résolution que $HBFS$, à savoir 52 304 s contre 84 657 s pour $HBFS$. La figure 5.8 montre que ce constat reste valide pour le temps de résolution de la plupart d'instances. Afin de comparer de façon équitable leur temps de résolution, nous considérons le benchmark résolu à la fois par $BTD\text{-}HBFS{+}DYN$ et par $HBFS$. Nous obtenons alors un total de 2 008 intances résolues en 79 020 s par $HBFS$ contre seulement 43 914 s pour $BTD\text{-}HBFS{+}DYN$.

Ces résultats peuvent être essentiellement expliqués par les enregistrements réalisés par $BTD\text{-}HBFS{+}DYN$, par la distinction des différentes composantes connexes du problème et la gestion dynamique de la décomposition qui atténue les inconvénients de l'exploitation traditionnelle de la décomposition. Le couplage de $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} assure notamment que les méthodes basées sur la décomposition sont 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 $BTD\text{-}HBFS{+}DYN$ 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.

BTD-HBFS+DYN avec H_5^{25} sur les instances les plus difficiles Nous focalisons maintenant nos observations sur le comportement de $BTD\text{-}HBFS{+}DYN$ 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 par $HBFS$). Notons que ces instances sont aussi facilement résolues par $BTD\text{-}HBFS{+}DYN$ vu qu'au niveau du cluster racine $BTD\text{-}HBFS{+}DYN$ se comporte comme $HBFS$ (lorsque nous exploitons la fusion des descendants relatifs au cluster racine). Nous pouvons distinguer trois partitions sur ce benchmark de 794 instances :

- 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 $BTD\text{-}HBFS{+}DYN$ 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, $BTD\text{-}HBFS{+}DYN$ peut se comporter simplement comme $HBFS$ ou faire des choix d'exploitation des clusters d'origine de la décomposition (au lieu de la fusion de leur descendants) jusqu'à atteindre le comportement de $BTD\text{-}HBFS$.

BTD-HBFS+DYN avec H_5^{25} vs HBFS en cas de dépassement du temps limite Nous comparons les bornes inférieures et supérieures rapportées par $HBFS$ et $BTD\text{-}HBFS{+}DYN$ dans le cas de dépassement du temps limite. Nous disposons de 375 instances qui ne sont pas résolues ni par $HBFS$, ni par $BTD\text{-}HBFS{+}DYN$ avec :

- Pour 252 instances, la borne supérieure calculée par $BTD\text{-}HBFS{+}DYN$ est strictement inférieure à celle de $HBFS$ contre seulement 52 instances pour $HBFS$.
- Pour 229 instances, $BTD\text{-}HBFS{+}DYN$ calcule une borne inférieure strictement supérieure à celle de $HBFS$ contre 146 instances pour $HBFS$.

La figure 5.9 compare l'écart entre les bornes inférieures et les bornes supérieures pour les deux algorithmes. Clairement, $BTD\text{-}HBFS{+}DYN$ offre un écart plus réduit que celui de $HBFS$. Cela montre que même lorsque l'instance n'est pas résolue, $BTD\text{-}HBFS{+}DYN$ est capable de donner des approximations de meilleure qualité que $HBFS$.

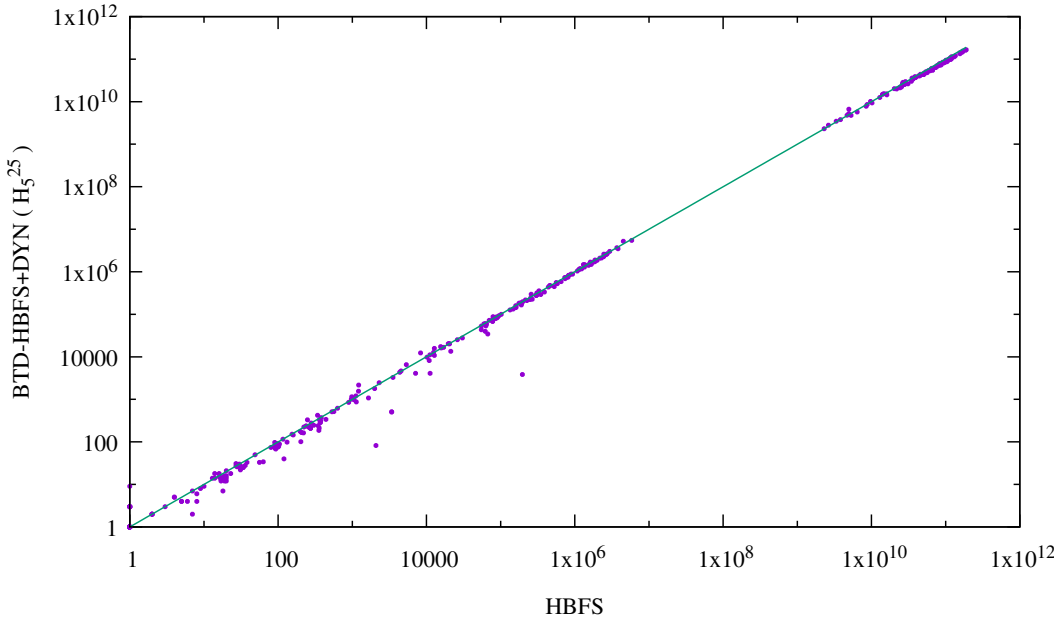


FIGURE 5.9 – Écart entre les bornes inférieures et supérieures pour $BTD\text{-}HBFS{+}DYN$ avec H_5^{25} et $HBFS$.

Bilan Au terme de l'analyse, nous avons démontré que $BTD\text{-}HBFS{+}DYN$ améliore $BTD\text{-}HBFS$, quelle que soit la décomposition utilisée, en nombre total d'instances résolues, en temps, mais également pour la qualité de bornes obtenues pour les cas des instances non résolues. Aussi, grâce à cette extension, les algorithmes basés sur une décomposition présentent davantage d'intérêt par rapport à ceux non basés sur une décomposition comme $HBFS$. Finalement, il semble nécessaire de donner des commentaires additionnelles sur la comparaison entre les temps d'exécution respectifs des différentes approches. En fait, une méthode peut être considérée comme plus efficace si son temps d'exécution est meilleur. Cependant, le nombre d'instances résolues doit être pris en compte. Plus précisément, nous considérons la comparaison des algorithmes ($HBFS$, $BTD\text{-}HBFS$, $BTD\text{-}HBFS{+}DYN$) et des décompositions ($Min\text{-}Fill$, $Min\text{-}Fill^4$, H_2, H_3, H_5^{25}, $H_5^{5\%}$) donnée dans les tableaux 5.1 et 5.2. Lorsque nous rapportons que $BTD\text{-}HBFS$ utilisant $Min\text{-}Fill$ résout 1 712 instances en 26 291 s, tandis que $BTD\text{-}HBFS{+}DYN$ utilisant H_5^{25} résout 2 039 instances en 52 304 s, nous pourrions être intéressés par le benchmark résolu par au moins une méthode parmi les deux méthodes qui incluent 2 049 instances. Nous pouvons constater que $BTD\text{-}HBFS$ exploitant $Min\text{-}Fill$ a résolu seulement 1 712 instances parmi les 2 049 instances en consommant 430 691 s. En effet, nous ajoutons ici le coût des 337 instances non résolues ($2\,049 - 1\,712 = 337$) en 20 minutes, qui est de 404 400 s, qui doit être ajouté aux 26 291 s utilisés pour résoudre les 1 712 instances. En ce qui concerne $BTD\text{-}HBFS{+}DYN$ utilisant H_5^{25}, après l'ajout du coût des 10 instances non résolues parmi les 2 049 instances, nous obtenons un total de 64 304 s. Ce faisant, nous déduisons que $BTD\text{-}HBFS{+}DYN$ (utilisant H_5^{25}) est 6,7 fois plus rapide que $BTD\text{-}HBFS$ (utilisant $Min\text{-}Fill$) tandis qu'il résout 327 instances additionnelles.

5.5 Conclusion

Dans le chapitre 4, nous avons proposé une exploitation dynamique de la décomposition arborescente pour la résolution des instances CSP. Cette dynamicité a été implémentée via la fusion des clusters. La fusion dynamique a permis d'améliorer significativement l'efficacité de la résolution en augmentant le nombre d'instances résolues et en diminuant en général le temps de résolution de chaque instance. Le succès du schéma dynamique dans le cadre du problème CSP nous a incité à proposer une gestion dynamique de la décomposition pour la résolution d'instances WCSP.

Comme pour le problème CSP, la décomposition arborescente a été déjà exploitée pour la résolution d'instances WCSP. La qualité souvent médiocre des décompositions utilisées a été un frein à leur exploitation pour la résolution. La proposition du nouveau cadre généraliste de calcul de décompositions $H\text{-}TD$ a été sensiblement bénéfique pour la résolution. Il n'a cependant pas permis de libérer suffisamment l'heuristique de choix de variables. Or, la liberté de l'heuristique de choix de variables joue un rôle essentiel dans l'augmentation de l'efficacité de la résolution. Les algorithmes de résolution non structurels comme $HBFS$ jouissent d'une liberté totale pour le choix de la prochaine variable. Ceci leur permet notamment de profiter pleinement des approches adaptatives dont l'intérêt peut être majeur.

Pour y remédier, nous avons proposé un nouvel algorithme de résolution d'instances WCSP qui vise à rendre l'exploitation de la décomposition moins contrainte. L'idée consiste à n'utiliser la décomposition sur un sous-problème que lorsque la résolution sans décomposition semble difficile. Elle sera ainsi utilisée « si besoin ». Cela évite de restreindre inutilement la liberté de l'heuristique de choix de variables pour des sous-problèmes « faciles ». En effet, la décomposition est habituellement utilisée pour pouvoir résoudre des instances difficiles du point de vue d'un algorithme tel que $HBFS$.

L'utilisation « si besoin » de la décomposition a rendu $BTD\text{-}HBFS$ plus performant. Elle a augmenté le nombre d'instances résolues et a significativement diminué le temps de résolution pour certaines instances. L'adoption du couplage de $BTD\text{-}HBFS{+}DYN$ et de H_5^{25} a clairement montré son intérêt lors de sa comparaison à $BTD\text{-}HBFS$ avec $Min\text{-}Fill^4$ ou à $HBFS$, les deux méthodes de l'état de l'art. Il a également montré qu'en pratique, l'algorithme peut se comporter comme un simple $HBFS$ notamment pour les instances faciles, avoir un comportement intermédiaire entre $HBFS$ et $BTD\text{-}HBFS$ ou même se comporter comme un $BTD\text{-}HBFS$ classique. Les expérimentations montrent finalement qu'en cas de dépassement du temps limite, $BTD\text{-}HBFS{+}DYN$ est généralement capable de fournir des bornes inférieures et supérieures de meilleure qualité que celles fournies par $HBFS$.

Finalement, toutes les modifications de la décomposition restent complètement transparentes à l'utilisateur et ne demande aucune intervention de sa part. Pour résoudre une instance donnée, il n'a pas désormais à choisir entre $HBFS$ ou $BTD\text{-}HBFS$ par exemple. En effet, si le solveur exploite l'algorithme $BTD\text{-}HBFS{+}DYN$, ce dernier s'occupera de trouver le bon compromis entre profiter pleinement la structure et avoir une liberté totale pour le choix de la prochaine variable.

Dans le prochain chapitre, nous nous intéressons à l'amélioration de l'exploitation de la décomposition arborescente pour la résolution du problème #CSP.