Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre

Philippe Jegou, Cyril Terrioux, Samba Ndiaye

Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre

Philippe Jégou **Samba Ndojh Ndiaye** **Cyril Terrioux**

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

Résumé

Lors des JFPC'2005, les auteurs de la présente contribution, ont montré l'intérêt que recèle l'exploitation des heuristiques pour s'assurer de l'efficacité pratique des méthodes de résolution de CSP par décomposition arborescente. Nous étendons ce travail en le généralisant. Nous montrons qu'au-delà d'heuristiques sur le parcours de l'arborescence associé à la décomposition, une gestion dynamique de l'heuristique de choix des variables s'avère cruciale. Tout en conservant les bornes de complexité ($O(exp(w+1))$ où w est la tree-width du CSP), nous proposons une voie pour s'affranchir en partie du carcan imposé par la structure arborescente. Cette démarche nous conduit à définir de nouvelles bornes de complexité qui offrent un compromis entre bornes théoriques et libération de l'ordre. En particulier, nous introduisons un nouveau paramètre k indiquant le degré de liberté laissé à l'heuristique de choix de variables qui permet de borner la complexité par $O(exp(2(w+k)))$. Les résultats expérimentaux présentés montrent l'intérêt d'une telle démarche.

1 Introduction

Le formalisme CSP (Constraint Satisfaction Problem) permet de représenter et de résoudre efficacement de nombreux problèmes, dont la modélisation se présente par l'expression d'un ensemble de variables à affecter en satisfaisant un ensemble de contraintes. Tester l'existence d'une solution est un problème NP-complet. Pour le résoudre, on a généralement recours à des techniques fondées sur le backtracking, dont l'efficacité pratique s'appuie sur la mise en œuvre de techniques de filtrage couplées à l'emploi d'heuristiques pour le choix des variables et valeurs à privilégier lors des affectations. La NP-complétude de CSP confère à ce type d'approche une complexité de nature exponentielle, précisément en $O(e.d^n)$ où n, e et d notent respectivement le nombre de variables, de contraintes et la taille maximum des domaines.

Une part importante des travaux réalisés par la communauté a donc consisté à développer des méthodes garantissant de meilleures bornes de complexité, ce qui s'avère possible en capturant certaines propriétés des instances. En particulier, il est bien connu que la meilleure borne à ce jour est fournie par la "largeur d'arborescence" (tree-width) d'un CSP, traditionnellement notée w, dont la valeur peut être significativement inférieure à n. Le recours à cet invariant de graphe a permis d'élaborer des méthodes dont la complexité temporelle est $O(n.d^{w^+ +1})$ (notation que nous simplifierons par $O(exp(w^+ + 1))$ où w^+ exprime une approximation de w), et dont la première fut proposée dans [2] sous l'appellation *Tree-Clustering* (arborescence de "clusters" de variables). Cette méthode s'appuie sur la notion de décomposition arborescente de graphe, formalisée par Robertson et Seymour [9]. Plusieurs variantes de cette approche ont été proposées. Le lecteur pourra consulter [3] pour une présentation et la comparaison de plusieurs d'entre elles.

Si le gain espéré par la complexité théorique est indéniable dans de nombreux cas, il a fallu attendre plus d'une décennie pour accéder à des développements permettant d'espérer une exploitation pratique de ce type d'approche. Récemment, une approche appelée BTD [7], exploitant mieux les bornes de complexité théorique et s'appuyant fortement sur les méthodes d'énumération classiques, a permis de montrer l'inté-

rêt pratique de l'exploitation de la structure du réseau.

Ces derniers développements ont ouvert un champ d'investigation nouveau, qui porte sur l'optimisation de la mise en œuvre de ces méthodes, principalement basée sur l'élaboration d'heuristiques. Lors des JFPC'2005 [5], nous nous sommes principalement intéressés à la recherche de "bonnes" décompositions arborescentes, fondées notamment sur des techniques et méthodes de calcul heuristiques [4]. Ce travail préliminaire évoquait aussi la question des stratégies de parcours de la décomposition arborescente considérée, soit de l'arborescence des clusters. La contribution que nous proposons ici a pour objet le développement de cette orientation, d'abord sur un plan théorique, puis heuristique et enfin expérimental. Elle s'appuie sur la méthode BTD [7] qui offre d'une part une approche par décomposition opérationnelle sur le plan pratique, et d'autre part, des degrés de liberté importants qui devraient faciliter l'élaboration et la mise en œuvre d'heuristiques.

Précisément, étant donnée une décomposition arborescente, donc une arborescence des clusters, au-delà de la simple étude des stratégies de son parcours, nous étudions le problème de la recherche d'ordres pour les affectations de variables qui l'exploiterons au mieux. Cette étude est motivée par l'aspect crucial des ordres d'affectation pour les méthodes fondées sur l'énumération. On sait en effet que toute méthode énumérative doit s'appuyer sur des heuristiques de choix de variables et de valeurs pertinentes. Pour être efficaces, ces heuristiques doivent être dynamiques, c'est-à-dire que les choix opérés doivent être élaborés au gré de la recherche et non au préalable de façon statique et définitive. Cette étude sur les ordres passe d'abord par une analyse des propriétés des ordres d'affectation qui garantissent des temps de calcul en $O(exp(w^+ + 1))$. Nous montrons ici que ces ordres d'énumération, tout en demeurant compatibles avec l'arborescence des clusters, peuvent être gérés dynamiquement. Cette étude consistant à délimiter le cadre où ce type de borne de complexité existe, nous permet de les étendre. Nous proposons, également, une nouvelle borne pour le temps, $O(exp(2(w + k)))$ où k est une constante entière. Une telle extension confère une importance accrue à l'apport et aux bénéfices des heuristiques.

Sur la base de ces propriétés, nous élaborons différentes heuristiques. Celles-ci ont donc pour objet d'orienter la recherche dans la décomposition arborescente, à la fois pour ce qui concerne le choix des clusters, mais aussi plus généralement, pour le choix des variables. Les critères sur lesquels sont fondées ces heuristiques sont liés aux propriétés topologiques du réseau de contrainte et de sa décomposition, ainsi qu'aux traits sémantiques de l'instance traitée. Nous montrons notamment comment des heuristiques basées sur l'espérance mathématique du nombre de solutions partielles d'un réseau décomposé peuvent améliorer considérablement les temps d'exécution. D'autres heuristiques proposées, comme celle basée sur la taille des clusters, obtiennent souvent des résultats similaires et qui peuvent être meilleurs sur les problèmes réels.

Cet article est organisé comme suit. Dans la section 2, nous rappelons les notions de base propres aux méthodes exploitant la décomposition arborescente de graphe, en particulier BTD. La section 3 étudie les propriétés sur les ordres d'énumération qui permettent de garantir des bornes de complexité du type $O(exp(w^+ + 1))$ et montre comment cette borne peut être étendue à $O(exp(2(w + k)))$ par l'introduction d'un nouveau paramètre k. La section 4 introduit différentes heuristiques alors que la section 5 présente une évaluation expérimentale des contributions des sections 3 et 4. Enfin, la dernière section trace les perspectives et donne une conclusion à cette contribution.

2 Rappels et contexte de l'étude

Un *problème de satisfaction de contraintes* (CSP) est défini par la donnée d'un triplet (X, D, C). X est un ensemble $\{x_1, \ldots, x_n\}$ de n variables. Chaque variable x_i prend ses valeurs dans un domaine fini donné dans D. Les variables sont soumises à des contraintes définies dans C. Etant donnée une instance (X, D, C), le problème CSP consiste à déterminer s'il existe une affectation de chaque variable qui satisfait chaque contrainte. Ce problème est NP-complet. Dans cet article, et sans manque de généralité, nous considérerons uniquement le cas des contraintes binaires (i.e. contraintes impliquant deux variables). Ainsi, la structure du CSP peut être représentée par le graphe (X, C), appelé *graphe de contraintes*. Les sommets de ce graphe sont les variables de X et une arête connecte deux sommets si les variables associées partagent une contrainte.

Le Tree-Clustering [2] constitue la méthode de référence pour résoudre un CSP en exploitant la structure de ce graphe de contraintes. Elle est basée sur la notion de décomposition arborescente de graphe [9] :

Définition Etant donné un graphe $G = (X, C)$, une *décomposition arborescente* de G est une paire $(E, \mathcal{T})$ avec $\mathcal{T} = (I, F)$ un arbre et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (ou cluster) E_i est un nœud de $\mathcal{T}$ et vérifie :

(i) $\cup_{i \in I} E_i = X$,

(ii) pour toute arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$,

(iii) pour tout $i, j, k \in I$, si k figure sur un chemin de i à j dans $\mathcal{T}$, alors $E_i \cap E_j \subseteq E_k$.

La largeur d'une décomposition arborescente $(E, \mathcal{T})$ est égale à $max_{i \in I}|E_i|-1$. La *largeur d'arborescence* ou *tree-width* w de G est la largeur minimale par rapport à toutes les décompositions arborescentes de G.

La complexité en temps du Tree-Clustering est $O(exp(w^+ + 1))$ et en espace, elle a été réduite à $O(n.s.d^s)$ où s est la taille du plus grand séparateur minimal du graphe [1] (ce séparateur est constitué par l'une des intersections $E_i \cap E_j$). Notons que le Tree-Clustering n'a jamais fourni de résultats en pratique, bien que possédant la meilleure borne de complexité théorique des méthodes CSP. Aussi, une approche alternative, également basée sur la décomposition arborescente de graphes, a été proposée dans [7]. Cette méthode, appelée BTD, semble offrir l'un des meilleurs résultats pratiques observés parmi les techniques de résolution structurelles.

La méthode BTD (pour Backtracking with Tree-Decomposition) procède par une recherche énumérative qui est guidée par un ordre statique des variables. Il est donc pré-établi à partir de la décomposition arborescente du réseau de contraintes. Aussi, la première étape de BTD consiste-t-elle en un calcul de décomposition arborescente. Cette décomposition permet d'exploiter certaines propriétés structurelles du graphe pendant la recherche pour élaguer certaines branches de l'arbre de recherche, ce qui distingue ainsi BTD des autres techniques classiques. Pour cela, BTD s'appuie sur la notion d'ordre compatible pour l'instanciation des variables.

Définition Considérons $(E, \mathcal{T})$ avec $\mathcal{T} = (I, F)$ une décomposition arborescente de CSP. Une numérotation sur E est dite *compatible*, s'il s'agit d'une numérotation préfixe de l'arbre $\mathcal{T} = (I, F)$ dont E_1 est la racine. Supposons que les éléments de $E = \{E_i : i \in I\}$ soient indexés par une telle numérotation. Dans ce cas, un ordre $\preceq_X$ des variables de X tel que $\forall x \in E_i$, $\forall y \in E_j$, avec $i < j$, $x \preceq_X y$ est appelé *ordre d'énumération compatible* (avec l'ordre des clusters).

En utilisant un ordre d'énumération compatible et en mémorisant certaines informations (appelées *goods* ou *nogoods*), il est possible d'éviter la visite de certaines parties de l'espace de recherche, car soit leur consistance sera déjà connue, soit leur inconsistance le sera. Un good est une instanciation consistante sur un séparateur (une intersection entre deux clusters), tel que le sous-arbre situé en dessous de ce séparateur soit consistant et admette une solution compatible avec le good. Ainsi, cette partie ne sera plus visitée parce sa consistance est connue. Un nogood est une instanciation consistante sur un séparateur qui ne peut pas être prolongée de manière consistante au sous-arbre situé en dessous de ce séparateur. De ce fait, toute instanciation partielle contenant ce nogood ne sera pas prolongée car cela conduit à un échec. L'ordre d'énumération donne un ordre partiel sur les variables. Les variables dans E_i sont instanciées avant celles dans E_j si $i < j$, exception faite des variables situées en-dessous d'un good. En fait, ces variables peuvent être affectées de manière consistante avec l'instanciation partielle. Donc, BTD ne les affecte pas en réalité, mais les considère comme tel. Pour les problèmes consistants, un travail supplémentaire sera opéré à la fin pour les instancier si une solution est recherchée. Ces variables sont appelées *variables instanciables* grâce à un good. Donc, les variables dans E_j sont affectées si celles dans E_i le sont déjà ou sont *instanciables* grâce à des goods. Pour compléter cet ordre, il faut une heuristique de choix de variables à l'intérieur de chaque cluster. Donc, un ordre d'énumération compatible sur les variables est donné par une numérotation compatible sur les clusters et un ordre sur les variables de chaque cluster.

Les résultats expérimentaux donnés dans [7] furent obtenus sans exploitation d'heuristiques visant à établir de bonnes stratégies pour un parcours de l'arbre de clusters. La marge laissée à l'heuristique dans ces expérimentations portait uniquement sur l'ordonnancement des variables internes à un cluster, garantissant ainsi un ordre compatible d'affectation des variables. L'ordre d'affectation des variables du problème a une importance fondamentale pour l'efficacité de toute méthode de résolution. Dans la section suivante, nous étendons l'étude préliminaire faite dans [5] sur un plan théorique afin de mieux cerner les propriétés qui doivent être vérifiées par l'ordre pour garantir des bornes de complexité intéressantes.

3 Bornes de complexité et ordres

3.1 Conservation de la borne

Il est bien connu que la dynamicité du choix joue un rôle crucial dans le contexte des méthodes de résolution énumératives. Toutefois, une libéralisation totale de l'ordre BTD ne permettrait plus de disposer des bornes de complexité que nous recherchons. Aussi, s'avère-t-il nécessaire d'étudier les limites qui permettent de garantir ces bornes tout en s'affranchissant au mieux de l'ordre originellement imposé à la méthode BTD.

Dans la version de base de BTD [7] rappelée en section 2, la résolution s'appuyait sur un ordre d'énumération compatible *a priori* statique. Nous montrons qu'il est possible de s'affranchir de cet aspect statique, selon certains degrés de liberté que nous répertorions

en définissant différentes classes d'ordres.

Classe 1. Ordre d'énumération statique (ordre de base décrit dans [7]). Cet ordre considère un ordre de visite statique des variables qui est un ordre d'énumération compatible.

Classe 2. Ordre statique des clusters et Ordre dynamique des variables. L'ordre de visite des clusters est un ordre compatible; il est donc statique. Par contre, à l'intérieur d'un cluster, l'ordre dans lequel sont affectées les variables peut être dynamique. Ainsi, pour toute affectation courante portant sur un ensemble de variables Y, si $x_i \in E_i$ est la dernière variable affectée, alors $\forall E_j \in E$, avec $j < i$, $\forall x_j \in E_j$, on a $x_j \in Y$. En d'autres termes, l'affectation d'une nouvelle variable $x_i \in E_i$ n'est possible que si les variables situées dans les clusters E_j qui précèdent E_i dans un ordre compatible sur les clusters ont toutes été affectées ou sont *instanciables* grâce à des goods.

Classe 3. Ordre dynamique des clusters et Ordre dynamique des variables. Pour toute affectation courante portant sur un ensemble de variables Y, si $x_i \in E_i$ et $x_i \in Y$, alors $\forall E_j \in E$, avec $i \neq j$ et tel que E_j figure sur le chemin allant de la racine E_1 à E_i, $\forall x_j \in E_j$, on a $x_j \in Y$. En d'autres termes, l'affectation d'une nouvelle variable $x_i \in E_i$ n'est possible que si les variables situées entre la racine E_1 et E_i ont toutes été affectées.

Classe 4. Ordre d'énumération dynamique. Dans cet ordre, aucune restriction n'est imposée. Ainsi, l'ordre de choix des variables, complètement dynamique, ne fait pas référence à l'arbre des clusters, excepté pour la première variable affectée qui sera prise dans E_1. En conséquence, l'ordre d'affectation des variables peut ne pas être un ordre d'énumération compatible.

Les expérimentations présentées dans [7] utilisaient déjà un ordre de Classe 2. Formellement, seuls les ordres de la Classe 1 sont des ordres d'énumération compatibles. Toutefois, on constate que les Classes 2 et 3 définissent des ordres assimilables, pour une affectation donnée, à des ordres d'énumération compatibles. En effet, pour toute affectation courante portant sur un ensemble de variables Y pour lequel l'ordre d'affectation appartient à la Classe 3, il est facile de prouver l'existence d'un ordre de Classe 1 identique à l'ordre d'affectation courant. Cette propriété confère aux ordres de Classe 3 la possibilité de produire des goods et des nogoods, puis de les exploiter dans des conditions identiques à celles utilisées pour les ordres de la Classe 1.

De plus, cette collection de classe d'ordre définit en fait une hiérarchie, pour laquelle chaque classe inclut strictement la précédente. Ainsi, la propriété énoncée ci-dessus est donc également valable pour les ordres de la Classe 2.

Si la Classe 4 autorise une totale liberté, elle ne permet pas de garantir de borne de complexité. En effet, supposons qu'à un instant donné, l'ordre d'énumération considéré ait permis d'affecter les variables d'un cluster E_i, excepté celles qui appartiennent à $E_i \cap E_j$, où E_j est un cluster fils de E_i. On supposera que celles qui figurent dans les clusters qui précèdent E_i sur le chemin allant de la racine E_1 à E_i ont été affectées. Supposons maintenant, que la variable suivante, notée x, prise dans E_j, ne figure pas dans $E_i \cap E_j$. L'énumération peut se poursuivre, mais lorsque toute l'arborescence enracinée en E_j aura été traitée, il ne sera possible de déduire aucun nogood sur $E_i \cap E_j$. Cela est dû au fait qu'après l'affectation de x, les affectations qui suivent sont tributaires de cette affectation de x. Par contre, on peut noter que la consistance du sous-problème associé à l'arborescence enracinée en E_j permet la prise en compte de l'affectation obtenue sur $E_i \cap E_j$ sous forme de good. En effet, s'il est possible d'affecter ce sous-problème de manière consistante, et en particulier sur $E_i \cap E_j$, cela signifie que cette sous-affectation possède une extension consistante sur tout le sous-problème : il s'agit là de la définition de good de E_i par rapport à E_j.

Concernant la borne de complexité, puisqu'il n'est pas possible de déduire de nogood sur $E_i \cap E_j$, l'élagage prévu par BTD lors d'une affectation ultérieure de E_i ne pourra s'opérer, et l'argument utilisé pour borner la complexité disparaîtra. En particulier, une affectation identique à celle considérée pourra se reproduire sur $E_i \cap E_j$ et aucun élagage ne pourra être réalisé pour arrêter la poursuite de la recherche sur cette même affectation.

Considérons maintenant la Classe d'ordres 3, et montrons qu'elle vérifie les conditions requises pour obtenir la borne de complexité.

Théorème La complexité temporelle de BTD pour le cas où l'ordre de résolution appliqué appartient à la Classe 3 est $O(exp(w^+ + 1))$.

Preuve Considérons un cluster E_j quelconque de l'arborescence des clusters. En fait, il suffit de s'assurer que toute affectation sur E_j ne peut être rencontrée qu'une seule fois. Considérons une affectation courante, pour laquelle les dernières variables affectées figuraient dans un cluster E_i, père de E_j. Par définition d'un ordre de Classe 3, les variables qui se trouvent dans les clusters figurant sur le chemin menant de la racine à E_i sont déjà affectées. De plus, aucune autre variable figurant dans le sous-problème enraciné en E_j n'a été affectée. Ainsi, toute affectation consistante $\mathcal{A}$ sur $E_i \cap E_j$ sera obtenue lors de l'affectation des variables de E_i. En particulier, aucune variable figurant dans la descendance de E_j ne sera affectée avant que

toutes celles de $E_i \cap E_j$ ne le soient. Une fois traité, le sous-problème enraciné en E_j aura conduit soit à un résultat consistant, soit à un échec. Dans chaque cas, $\mathcal{A}$ sera enregistré, soit dans le premier cas avec le statut de good, soit de nogood dans le second cas. Notons $\mathcal{A}'$ l'affectation produite sur E_j. Ainsi, lors d'une résolution ultérieure, respectant un ordre de classe 3, si l'affectation $\mathcal{A}$ est à nouveau rencontrée après le traitement de E_i, la résolution n'ira pas visiter la descendance de E_j puisque l'exploitation du good ou du nogood déjà mémorisé stoppera immédiatement la recherche sur un échec ou un succès. On en déduit que l'affectation $\mathcal{A}'$ sur E_j ne pourra alors être rencontrée, car seules les variables de $E_i \cap E_j$ auront à nouveau fait l'objet d'une réaffectation. On dispose ainsi, et avec les mêmes arguments fournis dans la preuve de complexité de BTD (page 57 dans [7]), d'une complexité en temps de $O(exp(w^+ + 1))$. □

En termes d'heuristique, cette propriété va nous permettre d'étendre les possibilités au sujet de l'ordre. Il sera ainsi possible, à tout point de la résolution, de choisir n'importe quel cluster, pour autant que ses ancêtres depuis la racine auront tous été affectés. Le choix du cluster est donc libéré. De plus, à l'intérieur, le choix des variables est quelconque (tout en restant compatible avec celui des clusters dans l'énumération courante).

Nous montrons comment, à partir de ce constat, il est possible d'étendre cette complexité.

3.2 Extension de la borne et des ordres

L'extension de la borne que nous proposons repose sur le fait que plutôt que de limiter l'ensemble des variables éligibles par une heuristique aux seules variables d'un seul cluster, nous autorisons l'heuristique à choisir d'affecter k variables supplémentaires prises dans une branche enracinée dans le cluster considéré, et sous certaines réserves. Formellement, cela revient à définir un ordre débordant de la Classe 3.

Définition Soient $G = (X, C)$ un graphe, k un entier strictement positif, on appelle l'ensemble des *décompositions arborescentes k-recouvrantes orientées* d'une *décomposition arborescente* $(E, \mathcal{T})$ de racine E_1 de G, toutes les *décompositions arborescentes* $(E', \mathcal{T}')$ de G qui vérifient :

(i) $E_1 \subset E'_1$, E'_1 étant la racine de $(E', \mathcal{T}')$

(ii) $E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots E_{i_K}$ et $E_{i_1} \cup E_{i_2} \cup \ldots E_{i_{K-1}} \subset E'_i$, avec $E_{i_1} \ldots E_{i_K}$ un chemin sur une branche de $(E, \mathcal{T})$

(iii) $|E'_i| \leq w^+ + k$, avec $w^+ = max_{E_i \in E}|E_i|$.

Définition Soient (X, D, C) un CSP, $(E, \mathcal{T})$ une décomposition arborescente de racine E_1 du graphe (X, C), k un entier strictement positif. Un ordre d'affectation est dit de *Classe 3 - k étendue*, si pour toute instanciation, il existe une *décomposition arborescente k-recouvrante orientée* de $(E, \mathcal{T})$ pour laquelle l'ordre de cette instanciation est de *Classe 3*.

On observe que la définition de la *Classe 3 - k étendue* force uniquement l'ordre d'une instanciation à être de *Classe 3*. Or, sur cette instanciation unique un ordre de *Classe 3* est aussi de *Classe 2*. Cette gestion dynamique de la décomposition arborescente a pour effet une modification de la borne de complexité car il sera parfois impossible d'enregistrer des nogoods. Néanmoins, avec la limitation imposée par le paramètre k, on peut déduire le théorème suivant :

Théorème La complexité temporelle de BTD pour le cas où un ordre de *Classe 3 - k étendue* est utilisé est $O(exp(2(w^+ + k)))$.

Preuve Pour démontrer ce théorème, il suffit de prouver que toute affectation d'un ensemble V de $2(w^+ + k)$ variables situées dans des clusters formant un chemin dans une branche de $\mathcal{T}$, ne sera produite qu'une seule fois, avec $\mathcal{T}$ l'arbre de la décomposition arborescente $(E, \mathcal{T})$ de racine E_1 du graphe (X, C) d'un CSP (X, D, C). Soit A une affectation qui contient les variables de V. Il existe donc une *décomposition arborescente k-recouvrante orientée* $(E', \mathcal{T}')$ de $(E, \mathcal{T})$ pour laquelle l'ordre d'affectation A est de *Classe 3*. Cette décompostion contient des clusters de taille au plus $w + k$, alors il existe un recouvrement de V avec au moins deux clusters. Soient $E'_{i_1} \ldots E'_{i_r}$ un chemin sur une branche de $(E', \mathcal{T}')$ qui recouvrent V. La consistance ou non de l'affectation A sur le problème enraciné en E'_{i_1} conduit à l'enregistrement de (no)goods sur les séparateurs entre les $E'_{i_1} \ldots E'_{i_r}$. On suppose que E'_{i_1} est différente de la racine de $(E', \mathcal{T}')$. Si $r = 2$ alors E'_{i_1} et E'_{i_2} ont pour taille $2(w^+ + k)$, ce qui conduit à l'enregistrement d'un (no)good sur le séparateur entre E'_{i_1} et son père et d'un autre entre E'_{i_1} et E'_{i_2}. En fait les (no)goods portent sur les variables des séparateurs entre clusters de la décomposition $(E, \mathcal{T})$ vu que ces variables suffisent pour diviser le problème en plusieurs composantes connexes. Ces séparateurs sont contenus dans les intersections des clusters des *décompositions arborescentes k-recouvrantes orientées* de $(E, \mathcal{T})$. Si $r > 2$, on enregistre au moins 2 (no)goods sur les séparateurs entre les clusters $E'_{i_1} \ldots E'_{i_r}$. Soit B une affectation partielle qui contient les variables qui précèdent V dans l'ordre de *Classe 3 - k étendue* et qu'on cherche à prolonger sur V avec les mêmes valeurs que A. Ce prolongement va conduire à la génération d'un des (no)goods enregistrés sur V au premier passage avant que toutes les variables soient affectées car V contient au moins 2 (no)goods. L'exploitation de ce

(no)good va permettre d'arrêter la recherche sur les autres variables de V non encore affectées vu que leur consistance ou inconsistance avec l'affectation courante est déjà connue. Alors l'affectation A ne sera produite qu'une fois. Maintenant, si E'_{i_1} est la racine de $(E', \mathcal{T}')$, alors V contient E_1. Il y aura au moins un (no)good enregistré entre E'_{i_1} et E'_{i_2}. Si $r > 2$, alors l'affectation ne sera pas reproduite car V contient au moins 2 (no)goods. Si $r = 2$, un seul (no)good est enregistré. Cependant, pour toute autre *décomposition arborescente k-recouvrante orientée* $(E'', \mathcal{T}'')$ de $(E, \mathcal{T})$, un ordre d'affectation de *Classe 3* va instancier les variables de V qui sont dans E_1 avant au moins $w + k$ autres variables de V. En effet, E_1 sera contenu dans la racine de $(E'', \mathcal{T}'')$ qui est de taille au plus $w + k$ et qui sera instanciée en premier suivant l'ordre de *Classe 3*. Or, dès que ces variables qui sont dans E_1 sont affectées, le (no)good enregistré va permettre de ne pas poursuivre la recherche sur les autres variables de V vu que leur consistance ou inconsistance avec l'affectation courante est déjà connue. On a donc montré qu'une affectation sur V sera produite au plus une fois.□

Plusieurs approches sont possibles pour le choix des variables à regrouper. Un choix pertinent repose sur la maîtrise du paramètre s, qui peut, par ce type d'opération, être diminué. Pour cela, il suffit de regrouper, quand c'est possible, les clusters qui maximisent la taille de leurs intersections. Ainsi, les intersections *a priori* associées aux plus grands séparateurs minimaux ne serviront plus au calcul et ainsi à la mémorisation des goods et nogoods.

4 Heuristiques

Maintenant que nous avons donné les conditions à respecter pour sauvegarder la borne de complexité de BTD, nous allons proposer plusieurs heuristiques qui les respectent et qui permettent d'améliorer de manière plus ou moins significative les performances pratiques de l'algorithme. Pour chaque type d'ordre sur les clusters ou les variables, nous définirons un ordre aléatoire dont la motivation est de donner un point de repère pour évaluer l'apport des autres heuristiques. Mais, tout d'abord, nous allons définir la notion d'espérance du nombre de solutions qui est utilisée dans plusieurs de ces heuristiques.

4.1 Estimation du nombre de solutions partielles

L'espérance mathématique du nombre de solutions est un critère qui prend en compte la densité du problème, la taille des domaines et la dureté des contraintes. Il permet de faire un choix de parcours de l'arbre de clusters en visitant en premier les clusters avec une espérance du nombre de solutions minimale pour une application simple du fameux principe du "first-fail". On peut ainsi espérer que les échecs vont survenir plus tôt et avoir un coût moindre. L'espérance mathématique du nombre de solutions d'un CSP a été définie dans [10]. Soit $P = (X, D, C)$ un CSP et t_{ij} la dureté de la contrainte entre les variables x_i et x_j, avec $1 \leq i < j \leq n$. S'il existe une contrainte entre x_i et x_j, la dureté t_{ij} est définie par le rapport entre le nombre de couples autorisés par la contrainte et le nombre de couples possibles. S'il n'existe pas de contrainte alors $t_{ij} = 0$. L'espérance mathématique du nombre de solutions de P est définie par la formule : $E(P) = d^n \prod_{1 \leq i < j \leq n}(1 - t_{ij})$. Soit $a = (a_1, \ldots, a_n)$ une instanciation totale, a est une solution de P si a_i et a_j sont compatibles pour la contrainte c_{ij}, pour tout $1 \leq i < j \leq n$. La probabilité de cet événement est de $(1 - t_{ij})$. Donc la probabilitéé pour que a soit solution est $\prod_{1 \leq i < j \leq n}(1 - t_{ij})$. Etant donné que le nombre d'instanciations totales est d^n alors l'espérance du nombre de solutions est $E(P) = d^n \prod_{1 \leq i < j \leq n}(1 - t_{ij})$. Ce critère donne une meilleure estimation de la difficulté du problème et du temps de résolution que la densité du graphe de contraintes ou le nombre de contraintes. Dans notre cas, on ne cherche pas à estimer le nombre de solutions du problème en entier, mais le nombre de solutions des différents clusters. Ce qui explique le terme de solutions partielles.

4.2 Ordres sur les clusters

Dans cette partie, nous définissons les heuristiques de choix de clusters pour les classes d'ordres 1, 2 et 3. Elles sont de deux types : statiques pour la classe 1, dynamiques pour les classes 2 et 3. Elles consistent essentiellement à choisir le cluster racine et à ordonner les fils d'un cluster. Un ordre statique est défini avant le début de la résolution. On commence par donner des critères qui permettent de choisir le cluster racine.

- *alea* : La racine est choisie aléatoirement.
- *minesp* : Cette heuristique est basée sur l'espérance du nombre de solutions partielles d'un cluster et de sa taille. Les expériences menées ont montré qu'il était toujours préférable de choisir comme prochain cluster un de taille importante. Cela justifie le choix du cluster qui minimise le ratio entre l'espérance du nombre de solutions partielles et la taille du cluster comme racine. Cette heuristique permet d'explorer en premier un cluster de grande taille avec peu de solutions.
- *taille* : On choisit le cluster de plus grande taille.
- *bary* : On choisit le cluster barycentre comme ra-

cine. Cette notion de barycentre utilise une autre de distance entre deux clusters dans l'arbre de clusters qui est définie par le plus court chemin entre les deux. Le cluster barycentre x est donc celui qui minimise $\Sigma_{y \in X} dist(x, y)$.

Nous proposons de même des heuristiques pour ordonner les clusters fils.

- $alea_f$: Les clusters fils sont ordonnés aléatoirement.
- $minesp_f$: Cette heuristique est similaire à $minesp$ mais elle ordonne les fils suivant la valeur croissante du ratio de ces derniers.
- $minesp_{fd}$: On ordonne les clusters fils suivant la valeur croissante de l'espérance du nombre de solutions des problèmes enracinés en chacun de ces clusters.
- $maxesp_{fd}$: On ordonne les clusters fils suivant la valeur décroissante de l'espérance du nombre de solutions des problèmes enracinés en chacun de ces clusters.
- $mintaille_{fd}$: On ordonne les clusters fils suivant la valeur croissante de la taille des problèmes enracinés en chacun de ces clusters.
- $maxtaille_{fd}$: On ordonne les clusters fils suivant la valeur décroissante de la taille des problèmes enracinés en chacun de ces clusters.
- $minsep_f$: On ordonne les clusters fils suivant la valeur croissante de la taille du séparateur avec le cluster parent.

Un ordre dynamique est calculé durant la résolution. Pour choisir le cluster racine, étant donné qu'il est nécessaire de le faire pour savoir où débuter la recherche, on peut utiliser les heuristiques statiques. Cependant on en propose une nouvelle :
pv : Les heuristiques dynamiques de choix de variables ont permis d'améliorer grandement l'efficacité des algorithmes d'énumération. Pour tirer profit de cela, on choisit une heuristique de choix de variables dynamique et on prend comme cluster racine celui dans lequel se trouve la prochaine variable (les prochaines variables) dans l'ordre.

La partie dynamique de l'ordre se situe en fait dans l'ordonnancement des fils :

- $alea_{fdyn}$: On choisit aléatoirement le prochain cluster fils à visiter.
- pv_f : C'est une déclinaison de pv pour ordonner la filiation. On visite en premier le cluster qui contient la prochaine variable parmi celles qui se trouvent dans les clusters fils non encore instanciés, dans l'ordre de l'heuristique de choix variables dynamique qu'on a choisie.
- $minesp_{fdyn}$: C'est une version dynamique de $minesp_f$ qui estime le nombre de solutions d'un cluster uniquement quand les variables de son père ont toutes été affectées. Elle fait un meilleur choix en tenant compte des modifications éventuelles du problème dues au filtrage des domaines durant la recherche.

4.3 Ordres sur les variables.

Dans cette partie, nous définissons les ordres dans lesquels les variables d'un cluster seront affectées. Les ordres statiques :

- $alea_s$: L'ordre d'instanciation des variables est calculé aléatoirement.
- mdd_s : On ordonne les variables suivant la valeur croissante du ratio taille du domaine sur degré.

Les ordres dynamiques :

- $alea_d$: On choisit la prochaine variable à instancier dans le cluster aléatoirement.
- mdd_d : On choisit comme prochaine variable à affecter celle qui minimise le ratio taille du domaine courant (après un éventuel filtrage) sur degré.

Toutes les heuristiques proposées donnent un éventail très large de combinaisons possibles pour définir des ordres totaux sur les variables de toutes les classes d'ordres définies dans la section 3. En outre, il est de même possible d'étendre les ordres de la *Classe 3* en fusionnant des clusters, à la *Classe 3 - k étendue* moyennant une extension de la borne de complexité.

4.4 Heuristiques pour la fusion des clusters

La fusion des clusters donne une plus grande liberté à l'heuristique de choix de variables et de ce fait élargit la dynamicité de la résolution. Cette dernière améliore significativement la résolution en pratique à condition de trouver la valeur juste pour k, au-delà de laquelle BTD ne tire plus assez profit de la structure et de la borne de complexité pour éviter les redondances. Nous proposons ici, plusieurs critères de fusions de clusters qui constituent un prélude aux choix des heuristiques d'ordonnancement dynamique des clusters et variables pour définir un ordre de la *Classe 3 - k étendue*.

- *sep* : Cette heuristique prend en paramètre la valeur à ne pas dépasser pour la taille des séparateurs. Elle fusionne chaque paire de clusters de type <père, fils> dont la taille du séparateur dépasse cette valeur.
- *vp* : On lui fournit le nombre minimal de variables propres (variables qui n'appartiennent pas au cluster père) que doit contenir un cluster. Elle fusionne tout cluster dont le nombre de variables propres est inférieur ou égal à cette valeur, avec son cluster père.
- *esp* : Ce critère est basé sur l'espérance du nombre de solutions d'un cluster. On fusionne deux clusters si le cluster résultant a moins de solutions

CSP (n,d,w,t,s,ns,p)	w^+	s	Classe 2				Classe 3		
			alea $alea_f$	*taille* $minsep_f$	*minesp* $minsep_f$	*minesp* $minesp_f$	*minesp* $minesp_{fdyn}$	*pv* pv_f	*taille* pv_f
$(a)(150,25,15,215,5,15,10)$	13,00	12,22	4,86	3,41	2,50	2,52	2,45	6,85	5,34
$(b)(150,25,15,237,5,15,20)$	12,54	11,90	16,96	>41,10	2,54	2,69	2,32	>40,07	>41,47
$(c)(150,25,15,257,5,15,30)$	12,16	11,40	4,76	3,38	4,95	5,06	4,97	5,67	3,55
$(d)(150,25,15,285,5,15,40)$	11,52	10,64	4,39	1,12	>36,87	>36,87	>37,27	0,95	1,16
$(e)(250,20,20,107,5,20,10)$	17,82	16,92	50,47	16,03	>56,81	>56,77	>56,44	>99,67	15,26
$(f)(250,20,20,117,5,20,20)$	17,24	16,56	48,64	23,25	14,06	14,03	13,04	>77,14	24,00
$(g)(250,20,20,129,5,20,30)$	16,80	15,80	M	92,52	64,63	64,87	>78,47	>81,60	>107,10
$(h)(250,20,20,146,5,20,40)$	15,92	15,24	M	26,24	3,92	3,91	4,51	10,61	17,99
$(i)(250,25,15,211,5,25,10)$	13,04	12,34	16,57	15,16	43,16	43,41	44,66	>53,53	17,89
$(j)(250,25,15,230,5,25,20)$	12,86	11,98	17,15	8,51	>42,61	>43,12	>50,84	10,93	19,17
$(k)(250,25,15,253,5,25,30)$	12,38	11,82	8,71	7,01	10,91	10,94	5,06	6,01	6,91
$(l)(250,25,15,280,5,25,40)$	11,80	11,16	15,46	3,82	16,84	16,88	18,13	>52,97	5,03
$(m)(250,20,20,99,10,25,10)$	17,92	17,02	M	M	M	M	M	M	M
$(n)(500,20,15,123,5,50,10)$	13,04	12,58	11,67	7,01	7,78	8,08	7,31	8,32	7,54
$(o)(500,20,15,136,5,50,20)$	12,94	12,10	11,68	25,54	24,19	23,49	27,01	7,26	15,11

TAB. 1 – Valeurs des paramètres w^+ et s et temps d'exécution en s de BTD avec mdd_d.

que les deux pris séparément, l'objectif étant de construire des clusters avec plus de variables et moins de solutions en vue de découvrir les inconsistances le plus tôt possible.

5 Résultats expérimentaux

Nous avons mené une étude expérimentale sur les classes d'ordres 1, 2, 3, 3-k étendue et heuristiques proposées pour voir leur intérêt d'un point de vue pratique au niveau chronométrique. Cette comparaison est faite sur des CSP aléatoires structurés partiels. la génération d'une instance d'une classe (n,d,w,t,s,ns,p) débute par la génération aléatoire d'un CSP structuré suivant le modèle donné dans [5]. Cette instance est définie par un ensemble n de variables qui ont des domaines de taille d. Son graphe de contraintes est un arbre de cliques avec ns sommets de taille au plus $w+1$ et des tailles de séparateurs bornées par s. t donne le nombre de couples de valeurs interdits pour chaque contrainte. Dans un second temps, on retire un pourcentage p d'arêtes de l'instance ainsi construite. Les expérimentations sont menées sur un PC sur linux avec un processeur Pentium 4 3,2 GHz et 1 Go de mémoire. Pour chaque classe d'instances, les résultats présentés sont les moyennes sur un ensemble de 50 instances. Le temps de résolution est limité à 30 minutes par instance, au-delà le solveur est arrêté et le problème correspondant est considéré comme non résolu. De ce fait, dans les tableaux, le symbole $>$ signifie qu'au moins une instance est non résolue et donc que le temps de résolution réelle est supérieure à la valeur donnée. Le caractère M veut dire qu'au moins une instance ne peut pas être résolue parce qu'elle requiert plus de 1 Go de mémoire. Dans [5], une étude a été menée sur les algorithmes de triangulations qui sont utilisés pour le calcul des décompositions arborescentes. Il en est ressorti que MCS donnait les meilleures décompositions au niveau de l'efficacité pratique. Cela justifie son utilisation pour le calcul des décompositions dans cette étude.

Les premières observations mettent en lumière les très mauvais résultats des ordres de la classe 1. Cela n'est pas surprenant, car les ordres statiques sur les variables ont un comportement similaire au niveau des méthodes énumératives. Les classes 2 et 3 donnent d'assez bons résultats grâce à l'apport de la dynamicité. Le tableau 1 montre le temps d'exécution des meilleures heuristiques et d'un ordre aléatoire, ainsi que la largeur des décompositions arborescentes calculées et la taille maximale de leurs séparateurs. Il permet de se rendre compte de l'importance d'un choix de racine judicieux. En effet, le symbole $>$ vient d'une instance non résolue à cause d'un choix de racine désastreux. Ce problème non résolu a pour effet d'augmenter de manière substantielle la moyenne des résultats alors que sur les 49 autres instances l'heuristique se comporte très bien par rapport à un choix aléatoire. Il est à noter aussi que les instances non résolues sont différentes pour les heuristiques *taille* et *minesp*. Les problèmes de mémoire des classes 2 et 3 peuvent être résolus par le passage à la *Classe 3 - k étendue* avec comme critère de fusion *sep*. Le tableau 2 montre les temps d'exécution des heuristiques de cette classe avec une taille de séparateur limitée à 5. Par manque de place, nous ne donnons que quelques classes d'instances (le tableau complet est disponible dans [6]). Avec des clusters de taille plus importante, les mauvais

CSP	w^+	s	*alea*	*taille*	*minesp*	*minesp*	*minesp*	*pv*	*taille*
			$alea_f$	$minsep_f$	$minsep_f$	$minesp_f$	$minesp_{fdyn}$	pv_f	pv_f
(g)	19,82	5,00	>220,65	33,93	4,66	4,61	4,41	41,92	34,20
(h)	20,44	5,00	>231,63	11,38	3,17	3,17	3,17	7,58	10,63
(i)	14,00	5,00	>47,60	5,86	7,75	7,71	6,65	8,86	6,44
(m)	57,28	4,88	>315,24	66,94	63,35	63,15	62,99	74,32	66,33
(o)	14,44	5,00	10,56	4,86	4,94	4,92	3,94	5,54	5,24

TAB. 2 – Valeurs des paramètres w^+ et s et temps d'exécution en s de BTD pour un ordre de *Classe 3 - k étendue* basé sur mdd_d et l'heuristique de fusion *sep* (avec une taille de séparateurs d'au plus 5).

choix ont des conséquences plus visibles. Un ordre aléatoire éprouve plus de difficultés, surtout sur les problèmes inconsistants. Les heuristiques *minesp* et *pv* résolvent toutes les instances sauf une dans deux classes à cause d'un mauvais choix de racine alors que *taille* arrive à résoudre une de plus. Hormis ces instances non résolues, *minesp* donne de très bons résultats. Au niveau de la classe (250,20,20,99,10,25,10) il est possible d'observer un apport net d'un choix dynamique des clusters fils. Mais cet apport devrait être plus important sur des problèmes avec un plus grand nombre de clusters fils ce qui augmenterait les choix possibles de même que leur impact. On constate que l'heuristique $minesp{+}minesp_{fdyn}$ donne les meilleurs résultats, mais l'heuristique $taille{+}minsep$ obtient souvent des résultats similaires et réussit même à résoudre toutes les instances dans la *Classe 3 - k étendue*. Le calcul de l'espérance suppose que les contraintes du CSP sont indépendantes, ce qui est le cas pour les instances considérées ici. Il est donc possible que pour des problèmes réels avec des contraintes dépendantes, une heuristique du type $taille{+}(minsep$ ou $pv_f)$ se comporte mieux que $minesp{+}minesp_{fdyn}$. L'extension de l'ordre avec *sep* comme critère de fusion et $s = 5$, permet d'améliorer les résultats des heuristiques. Pour trouver la meilleure valeur de s, on a fait varier cette dernière sur le tableau 3, avec l'heuristique $minesp{+}minesp_{fdyn}$. Quand la valeur de s est grande, la modification de la décomposition est assez limitée, avec une faible augmentation de la taille des clusters. Plus s diminue, plus l'ordre des variables est dynamique et plus la résolution devrait être efficace. Néanmoins, à partir d'une certaine valeur, la taille des clusters devient trop grande et leur nombre trop petit pour permettre une utilisation de la structure pour limiter les redondances et une conservation de bornes de complexité acceptables. A partir de là, le nombre d'instances non résolues se multiplient et le comportement de BTD se rapproche de celui d'algorithmes énumératifs comme FC ou MAC (qui sont souvent incapables de résoudre une bonne partie des instances utilisées ici). Cependant, on observe l'existence d'une instance non résolue pour la classe (250,20,20,107,5,20,10) entre $s = 5$ et $s = 10$, et pour la classe (250,20,20,129,5,20,30) entre $s = 7$ et $s = 10$. Cela est dû à un mauvais choix de racine de l'heuristique $minesp{+}minesp_{fdyn}$. Pour les autres instances, le comportement est celui attendu.

On observe un comportement semblable avec l'heuristique de fusion *vp* (voir le tableau 4). Le temps de résolution va généralement s'améliorer lorsque le nombre de variables propres dans les clusters augmente. Mais, à partir d'un certain seuil (variable selon les classes d'instances considérées), l'apport de l'exploitation de la structure disparaît du fait de la diminution du nombre de clusters et de l'augmentation significative de leur taille. Notons que cette heuristique permet d'éviter le problème de consommation excessive de mémoire bien qu'elle n'agissent pas directement sur le paramètre s. Enfin, concernant l'heuristique de fusion *esp*, elle revient généralement à regrouper une grande partie des clusters ensemble. Elle s'avère donc inutilisable en pratique.

On a pu montrer en pratique que la dynamicité de l'ordre des variables permet de faire des choix plus éclairés et ainsi d'améliorer les performances de la méthode. Néanmoins, il est nécessaire de maintenir des bornes de complexités intéressantes pour tirer profit de la structure des problèmes. Alors, la *Classe 3 - k étendue* avec un k maîtrisé, donne les meilleures stratégies de parcours des décompositions arborescentes.

6 Conclusion

Dans cet article, nous avons étudié les méthodes de résolution de CSP par décomposition arborescente afin de renforcer leur intérêt pratique. Nous avons axé notre étude à la fois sur un plan théorique, mais aussi sur un plan heuristique et expérimental.

L'analyse des ordres d'affectation nous a permis de montrer comment il est possible de conserver la borne de complexité théorique classique, $O(exp(w + 1))$ où w est la tree-width du réseau de contraintes, tout en offrant une liberté dans le choix des variables, et donc dans la conception d'heuristiques dynamiques. Sur cette base, nous avons proposé d'introduire un paramètre nouveau dans ce cadre, qui sert à définir un

CSP	3		4		5		6		7		8		9		10	
(g)	>78,87	103	5,57	44	4,41	17	4,40	8	>41,47	4	>41,38	4	>41,51	4	>41,00	4
(h)	>87,78	103	19,60	44	3,17	18	3,24	13	3,36	10	3,59	6	6,99	6	8,03	5
(i)	19,26	69	6,91	48	6,65	1	14,76	1	18,81	1	19,02	1	20,07	1	19,99	1
(m)	>466,65	152	>191,42	152	62,99	110	67,94	91	44,41	72	44,03	62	35,83	30	53,33	6
(o)	22,75	122	3,41	63	3,94	5	4,70	3	5,80	2	6,40	2	6,63	2	6,45	2

TAB. 3 – Temps d'exécution en s de BTD et valeur de k pour un ordre de *Classe 3 - k étendue* basé sur mdd_d+*minesp*+$minesp_f$ et l'heuristique de fusion *sep* pour différentes tailles maximum de séparateurs.

CSP	1	2	3	4	5	6	7	8
(g)	16,30	17,23	2,85	>43,84	12,03	14,14	2,93	16,26
(h)	8,44	5,37	6,02	4,84	18,84	20,35	20,08	31,05
(i)	39,47	9,37	5,20	5,15	5,14	6,04	8,74	15,12
(m)	47,65	44,92	59,90	61,25	66,37	52,31	57,56	>96,71
(o)	6,99	12,80	9,69	9,70	3,64	7,42	>43,61	>48,23

TAB. 4 – Temps d'exécution en s de BTD selon k pour un ordre de *Classe 3 - k étendue* basé sur mdd_d+*minesp*+$minesp_f$ et l'heuristique de fusion *vp* avec des clusters ayant au moins $k+1$ variables propres.

degré de liberté laissé à l'heuristique de choix de variables. Cela nous a permis d'introduire une nouvelle borne de complexité, $O(exp(2(w+k)))$ où k est ce paramètre. Bien que de moindre qualité par rapport à la borne théorique classique, le compromis ainsi offert nous permet d'améliorer, par le jeu d'heuristiques plus riches, l'efficacité pratique de ce type de méthode. Enfin, nous avons proposé une collection d'heuristiques de choix de variables et de parcours de l'arborescence associée à la décomposition dont nous avons montré expérimentalement l'intérêt.

Ce travail nous semble important car il opère un compromis entre deux démarches souvent observées dans le cadre de la décomposition de CSP. Soit le développement de nouvelles méthodes (avec bornes de complexité) difficiles à justifier sur un plan pratique (cf. [3]), soit l'exploitation de la structure de problèmes pour introduire de nouvelles heuristiques pour les méthodes procédant par backtracking, mais cette fois-ci sans borne de complexité (cf. [8]).

Nous envisageons une poursuite à ces travaux qui consiste à traiter les problèmes de satisfaction de contraintes avec optimisation. Ces problèmes sont bien connus pour être significativement plus difficiles que le CSP de décision. Aussi l'apport de meilleures heuristiques et la plus value potentielle qu'elles peuvent offrir devraient permettre d'améliorer significativement leurs performances. Cette tâche n'est cependant pas simple, car les heuristiques pour la décision ne sont pas toutes aisément généralisables à l'optimisation.

Références

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

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

[3] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[4] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, pages 777–781, 2005.

[5] P. Jégou, S. N. Ndiaye, and C. Terrioux. Sur la génération et l'exploitation de décompositions pour la résolution de réseaux de contraintes. In *Actes des JFPC'2005*, pages 149–158, 2005.

[6] P. Jégou, S. N. Ndiaye, and C. Terrioux. Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre. Technical Report LSIS.RR.2006.004, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2006.

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

[8] W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proceedings of ICTAI*, pages 542–548, 2004.

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

[10] B. Smith. The Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI*, pages 100–104, 1994.