Une technique de décomposition pour Max-CSP

Hachémi Bennaceur, Christophe Lecoutre, Olivier Roussel

Actes JFPC 2008

Une technique de décomposition pour Max-CSP

Hachemi Bennaceur Christophe Lecoutre Olivier Roussel

Université Lille-Nord de France, Artois, F-62307 Lens
CRIL, F-62307 Lens
CNRS UMR 8188, F-62307 Lens
rue de l'université, SP 16, F-62307 Lens, France
{bennaceur,lecoutre,roussel}@cril.fr

Résumé

L'objectif du problème Max-CSP (Maximal Constraint Satisfaction Problem) est de trouver une instanciation qui minimise le nombre de contraintes violées dans un réseau de contraintes. Dans cet article, à partir du concept de contraintes disjonctives inférées introduit par Freuder et Hubbe, nous montrons qu'il est possible d'exploiter les compteurs d'arc-incohérence, associés à chaque valeur du réseau, de manière à éviter l'exploration de parties inutiles de l'espace de recherche. Le principe est de raisonner à partir de la distance entre les deux meilleures valeurs du domaine d'une variable, selon ces compteurs. Sur cette base, on peut mettre en place une technique de décomposition qui peut être utilisée tout au long de la recherche de manière à réduire le problème courant en sous-problèmes plus simples. Il est intéressant de noter que cette approche ne dépend pas de la structure du graphe de contraintes, comme cela est généralement proposé. Comme alternative à la décomposition, il est possible de poster des contraintes dures qui peuvent être utilisées pour élaguer l'espace de recherche. L'intérêt de notre approche est illustré en pratique, avec cette alternative, par une expérimentation basée sur un algorithme classique de séparation et évaluation, à savoir PFC-MRDAC.

Abstract

The objective of the Maximal Constraint Satisfaction Problem (Max-CSP) is to find an instantiation which minimizes the number of constraint violations in a constraint network. In this paper, inspired from the concept of inferred disjunctive constraints introduced by Freuder and Hubbe, we show that it is possible to exploit the arc-inconsistency counts, associated with each value of a network, in order to avoid exploring useless portions of the search space. The principle is to reason from the distance between the two best values in the domain of a variable, according to such counts. From this reasoning, we can build a decomposition technique which can be used throughout search in order to decompose the current problem into easier sub-problems. Interestingly, this approach does not depend on the structure of the constraint graph, as it is usually proposed. Alternatively, we can dynamically post hard constraints that can be used locally to prune the search space. The practical interest of our approach is illustrated, using this alternative, with an experimentation based on a classical branch and bound algorithm, namely PFC-MRDAC.

1 Introduction

Le problème de satisfaction de contraintes (CSP pour Constraint Satisfaction Problem) est la tâche NP-difficile de déterminer si un réseau de contraintes donné est cohérent ou non, i.e. si il est possible d'assigner une valeur à chaque variable de manière à satisfaire toutes les contraintes. Lorsqu'aucune solution ne peut être trouvée, il peut être intéressant d'identifier une instanciation complète qui satisfasse le plus grand nombre de contraintes (ou de manière équivalente, qui minimise le nombre de contraintes violées). Ce nouveau problème est appelé Max-CSP (pour Maximal Constraint Satisfaction Problem).

Durant la dernière décennie, de nombreux travaux ont été entrepris pour résoudre ce problème (ainsi que son extension directe, WCSP). L'approche élémentaire (complète) consiste à employer un mécanisme de séparation et évaluation (branch and bound), explorant l'espace de recherche en profondeur d'abord tout en maintenant une borne supérieure, le coût de la meilleure solution trouvée jusqu'alors, et une borne inférieure, minorant la meilleure extension possible de l'instanciation partielle courante.

Lorsque la valeur de la borne inférieure devient plus grande ou égale à la valeur de borne supérieure, un retour-arrière (ou un processus de filtrage) s'impose. Les calculs de minorants de violations de contraintes ont été améliorés au cours des ans en exploitant des compteurs d'incohérence [11, 17, 1, 16], des ensembles conflictuels disjoints de contraintes [22], ou des transferts de coûts entre les contraintes [18, 6, 7, 5].

Comme alternatives, certaines approches combinent la recherche par séparation et évaluation avec la programmation dynamique ou l'exploitation structurelle. D'un coté, on trouve la recherche basé sur les poupées russes (Russian Doll Search) [23, 21] et l'élimination de variables [15]. Leur principe repose sur la résolution de sous-problèmes successifs, un par variable du problème initial, et peuvent être considérées comme des méthodes de programmation dynamique. D'un autre coté, les méthodes de décomposition structurelle [13, 14, 20, 19, 8] exploitent la structure des problèmes afin d'établir les conditions sous lesquelles une décomposition peut être envisagée. De telles méthodes, basées sur la décomposition arborescente, fournissent des complexités temporelles théoriques intéressantes qui dépendent de la largeur de la décomposition (tree-width) et obtiennent des résultats probants.

Dans [9], Freuder et Hubbe ont proposé d'exploiter, pour la satisfaction de contraintes, le principe des contraintes disjonctives inférées (inferred disjunctive constraints) : étant donné un réseau de contraintes binaires cohérent P, pour toute assignation (X, a) où X est une variable de P et a une valeur dans le domaine de X, si il n'y a pas de solution assignant a à X, alors il y a une solution assignant une valeur (à une variable autre que X) qui est incohérente avec (X, a). En utilisant ce principe, les auteurs montrent qu'il est possible de décomposer dynamiquement un problème sans autre restriction imposée par la structure.

Dans cet article, nous généralisons cette approche à Max-CSP (en incluant le cas des contraintes non binaires) en exploitant les compteurs d'arc-incohérence associés aux valeurs du problème. Le compteur d'arc-incohérence (aic en abrégé) d'un couple (X, a) correspond au nombre de contraintes qui ne supportent pas (X, a). L'écart aic associé à une variable X est la différence (en valeur absolue) entre les deux plus petites valeurs des compteurs d'arc-incohérence associées aux différentes valeurs de X plus un. Nous montrons qu'il est possible de raisonner à partir de l'écart aic pour obtenir une condition sous laquelle nous avons la garantie d'obtenir une solution optimale, tout en évitant de parcourir certaines portions de l'espace de recherche.

À partir de ce raisonnement, nous pouvons construire une technique de décomposition qui peut être utilisée tout au long de la recherche pour décomposer le problème courant en sous-problèmes plus simples, généralisant pour Max-CSP l'approche introduite dans [9]. Il est important de remarquer qu'à la différence des méthodes de décomposition plus classiques, cette approche ne dépend pas de la structure du graphe de contraintes puisque la décomposition peut toujours être appliquée, quelle que soit la structure. De manière alternative, il est possible de poster dynamiquement des contraintes dures qui peuvent être utilisées localement pour élaguer l'espace de recherche. Selon l'implantation, ces contraintes dures peuvent participer à la propagation de contraintes, ou plus simplement imposer des retours-arrières.

Cet article est structuré comme suit. Après quelques rappels techniques, nous introduisons le théorème central de ce papier. Nous présentons ensuite deux approches déclinant deux exploitations possibles de ce théorème : une approche par décomposition et une approche par élagage. Avant de conclure, nous montrons le potentiel de ce travail à l'aide d'une expérimentation basée sur l'algorithme PFC-MRDAC.

2 Préliminaires

Le travail présenté dans ce papier concerne le problème de satisfaction de contraintes (CSP) discret. Chaque instance P du CSP correspond à un réseau de contraintes qui est défini par un ensemble fini de n variables $\{X_1, X_2, \ldots, X_n\}$ et un ensemble fini de e contraintes $\{C_1, C_2, \ldots, C_e\}$. À chaque variable X est associé un domaine fini de valeurs possibles, noté $dom(X)$. Chaque contrainte C porte sur un sous-ensemble ordonné $scp(C)$ de variables de P, appelé la portée de C, et spécifie l'ensemble $rel(C)$ des combinaisons de valeurs autorisées pour les variables de sa portée. $|scp(C)|$ est l'arité de la contrainte C. C est binaire si elle ne porte que sur deux variables (son arité est 2). Un réseau est binaire si il n'est composé que de contraintes binaires ou unaires. Il est normalisé si il ne contient pas deux contraintes ayant la même portée. Deux variables sont dites voisines si toutes les deux appartiennent à la portée d'une même contrainte.

Une instanciation complète est l'affectation d'une valeur à chaque variable du réseau. Soit s une instanciation complète, $s(X, a)$ est l'instanciation complète obtenue à partir de s en remplaçant la valeur de X dans s par a. Une instanciation complète s viole une contrainte C (C est insatisfaite par s) si la projection de s sur $scp(C)$ n'appartient pas à $rel(C)$. Une solution du réseau est une instanciation complète qui satisfait toutes les contraintes du réseau. Un réseau est dit cohérent ssi il possède au moins une solution, sinon le réseau est dit incohérent.

Dans certains cas, une instance CSP peut-être surcontrainte, et n'admet donc pas de solution. Dans ce cas, on est souvent amené à considérer le problème d'optimisation qui consiste à déterminer une instanciation complète optimale selon un certain critère donné. Dans cette présentation, nous considérons le problème Max-CSP qui consiste à

déterminer une solution qui satisfasse le plus grand nombre de contraintes possible. Une instance Max-CSP peut-être aussi représentée par un réseau de contraintes.

Soit la contrainte C avec $scp(C) = \{X_{i_1}, \ldots, X_{i_r}\}$, tout tuple de $dom(X_{i_1}) \times \ldots \times dom(X_{i_r})$ est appelé tuple valide de C. La valeur a pour la variable X sera notée (X, a). Une contrainte C supporte la valeur (X, a) (ou autrement dit, la valeur a possède un support sur C) ssi soit $X \notin scp(C)$, soit il existe un tuple valide de C appartenant à $rel(C)$ et contenant la valeur a pour X. Une valeur est arc-cohérente (généralisée) lorsque elle possède un support sur chacune des contraintes du réseau. Pour les réseaux binaires normalisés, on dit que la variable Y supporte la valeur (X, a) ssi soit il n'existe pas de contraintes liant X et Y soit il existe une telle contrainte supportant (X, a). Pour une contrainte binaire C avec $scp(C) = \{X, Y\}$, une valeur (X, a) est compatible avec une valeur (Y, b) si (a, b) appartient à $rel(C)$. Le compteur d'arc-incohérence, noté $aic(X, a)$, d'une valeur (X, a) est le nombre de contraintes (variables, dans le cas de CSP binaire normalisé) qui ne supportent pas (X, a).

3 Théorème central

Cette section présente le résultat central de cet article qui généralise pour Max-CSP l'approche de [9] introduite pour le problème de satisfaction de contraintes binaires. Nous utilisons les définitions suivantes :

Definition 3.1. *Soit P une instance Max-CSP et X une variable de P.*

- *Une* meilleure valeur aic *de X est une valeur $a \in dom(X)$ telle que $aic(X, a)$ est minimal (i.e. $\forall c \in dom(X), aic(X, a) \leq aic(X, c)$).*
- *Une* deuxième meilleure valeur aic *de X est une valeur $b \in dom(X)$ telle que $b \neq a$ et $\forall c \in dom(X) - \{a, b\}, aic(X, a) \leq aic(X, b) \leq aic(X, c)$.*
- L'écart aic *d'une variable X est défini comme $\delta = aic(X, b) - aic(X, a) + 1$.*

Theorem 3.1. *Soit P une instance Max-CSP, X une variable de P, a l'une des meilleures valeurs aic de X, δ l'écart aic de X et $C_1, \ldots, C_m$ les contraintes sur X qui supportent (X, a). Il existe toujours une solution optimale s^* de P telle que :*

- *soit X prend la valeur a dans s^*,*
- *soit X a une valeur différente de a dans s^*, et au moins δ contraintes parmi $C_1, \ldots, C_m$ sont violées par $s^*(X, a)$.*

Démonstration. Si P a une solution optimale où X vaut a, la première condition est trivialement vraie et le théorème est vérifié. Sinon, s'il n'existe aucune solution optimale avec $X = a$, soit $s^* = (v_1, ..., v, ..., v_n)$ l'une des solutions optimales de P, et soit v la valeur de X dans s^*. Notons par C_X l'ensemble des contraintes de P portant sur X (C_X est un sur-ensemble de $\{C_1, \ldots, C_m\}$) et posons que s^* viole p contraintes de C_X et que $s^*(X, a)$ viole q contraintes de C_X. Puisqu'il n'y a aucune solution optimale avec $X = a$, $s^*(X, a)$ viole forcément plus de contraintes de C_X que s^* et donc $q > p$. De ce fait, nous avons $p \geq aic(X, v)$ et $q \geq aic(X, a)$ puisque les compteurs d'arc-incohérence calculés pour P sont des bornes inférieures des compteurs aic après l'instanciation complète de toutes les variables de P. Donc, $\exists t \geq 0, r \geq 0$ tels que $p = aic(X, v) + r$ et $q = aic(X, a) + t$. Puisque $q > p$, on obtient $aic(X, a) + t > aic(X, v) + r$ ou de manière équivalente $t > aic(X, v) - aic(X, a) + r$. Comme $v \neq a$, nous avons $aic(X, v) \geq aic(X, b)$ (b est une seconde meilleur valeur aic de X) et donc $t > aic(X, b) - aic(X, a) + r$. Comme $r \geq 0$ et $\delta = aic(X, b) - aic(X, a) + 1$, on obtient $t \geq \delta$. Cela signifie que au moins δ contraintes de P qui supportent (X, a) portent sur des variables dont les valeurs assignées par s^* sont incompatible avec (X, a). De ce fait, le théorème est vérifié. □

Ce théorème peut être utilisé de deux manières différentes : soit pour décomposer une instance Max-CSP en sous-problèmes, soit comme une règle d'élagage de l'arbre de recherche.

4 Une approche par décomposition

La décomposition d'une instance Max-CSP P autour d'une variable X est définie comme suit.

Definition 4.1. *Soit P une instance Max-CSP, X une variable de P, a l'une des meilleures valeurs aic de X, δ l'écart aic de X et m le nombre de contraintes portant sur X qui ont un support pour (X, a). La décomposition d'une instance Max-CSP P autour de (X, a) génère les sous-problèmes $P_0, P_1, \ldots P_k$ (avec $k = \binom{m}{\delta}$) définis par :*

- *P_0 est dérivé de P en assignant a à la variable X ;*
- *P_i (avec $i \in 1..k$) est dérivé de P en éliminant a du domaine de X et restreignant les assignations des voisins de X de telle sorte qu'au moins δ contraintes de P parmi celles supportant (X, a) ne supportent plus (X, a) dans P_i.*

Ces sous-problèmes peuvent être résolus indépendamment et le théorème 3.1 garantit qu'au moins l'un d'entre eux contient une solution optimale de P. Il doit être noté que cette décomposition peut éliminer certaines solutions optimales (équivalentes) de P.

Les sous-problèmes introduits dans la définition ne sont pas disjoints, ce qui signifie qu'une assignation peut être solution de plusieurs sous-problèmes simultanément. Il est cependant facile de générer des sous-problèmes disjoints comme cela sera montré en section 4.2. Avec m désignant

le nombre de contraintes supportant $X = a$, cette décomposition génère $1 + \binom{m}{\delta}$ sous-problèmes (avec $\delta = 1$, ce nombre est égal à $1 + m$ et est borné par $n - aic(X, a)$) où n désigne le nombre de variables. Bien que le nombre de sous-problèmes soit exponentiel par rapport à δ, la section 4.3 prouve que l'espace de recherche des différents sous-problèmes $P_0, \ldots P_k$ est exponentiellement plus petit que que l'espace de recherche du problème initial P à condition que des sous-problèmes disjoints soient générés. Cela signifie que la décomposition est toujours avantageuse (en théorie) car, même si de nombreux sous-problèmes peuvent être engendrés, ils sont toujours plus simples à résoudre globalement que le problème initial.

4.1 Exemple

Afin d'illustrer la technique de décomposition, considérons le réseau de contraintes binaires P composé de trois variables $\{X_1, X_2, X_3\}$ avec $dom(X_i) = \{1, 2, 3\}$ pour $i \in 1..3$, et trois contraintes $\{C_{12}, C_{13}, C_{23}\}$ définies par les tables suivantes (tuples autorisés) :

$rel(C_{12})$

X_1	X_2
1	1
1	2
3	1

$rel(C_{13})$

X_1	X_3
1	1
1	2
2	1
2	3
3	2

$rel(C_{23})$

X_2	X_3
1	3
3	1

P est incohérent, et toute solution optimale du Max-CSP associé à P viole une contrainte. Par exemple, $X_1 = 1, X_2 = 1, X_3 = 2$ est une solution optimale qui viole la contrainte C_{23}.

Nous devons sélectionner une variable et l'une de ses meilleures valeurs aic pour appliquer la stratégie de décomposition à P. Par exemple, $(X_1, 1)$ est une meilleure valeur aic de X_1 car $aic(X_1, 1) = 0$, $aic(X_1, 2) = 1$ et $aic(X_1, 3) = 0$. La décomposition de P autour de la valeur $(X_1, 1)$ conduit à la construction des trois sous-problèmes indépendants suivants :

P_0 est formé à partir de P en affectant X_1 à 1. Dans P_0, $dom(X_1^0) = \{1\}, dom(X_2^0) = dom(X_3^0) = \{1, 2, 3\}$.

P_1 est formé à partir de P en imposant que $X_1 \neq 1$ et en réduisant le domaine de X_2 aux valeurs incompatibles avec $(X_1, 1)$. Dans P_1, $dom(X_1^1) = \{2, 3\}, dom(X_2^1) = \{3\}, dom(X_3^1) = \{1, 2, 3\}$.

P_2 est formé à partir de P en imposant que $X_1 \neq 1$ et en réduisant le domaine[1] de X_2 aux valeurs compatibles à $(X_1, 1)$ et celui de X_3 aux valeurs incompatibles à $(X_1, 1)$. Dans P_2, $dom(X_1^2) = \{2, 3\}, dom(X_2^2) = \{1, 2\}, dom(X_3^2) = \{3\}$.

Notant que le sous-problème de P avec $dom(X_1) = \{2, 3\}$, $dom(X_2) = \{1, 2\}$ et $dom(X_3) = \{1, 2\}$ est ignoré et ce sous-problème contient une solution optimale du problème initial P, à savoir $X_1 = 3, X_2 = 1$ et $X_3 = 2$.

Un second exemple est obtenu en modifiant légèrement le problème initial. Supposons que la valeur 3 de X_1 est incompatible avec toutes les valeurs de X_3. La nouvelle relation entre X_1 et X_3 est :

$rel(C_{13})$

X_1	X_3
1	1
1	2
2	1
2	3

Dans ce cas, X_1 possède une seule meilleure valeur, puisque $aic(X_1, 1) = 0, aic(X_1, 2) = 1$ et $aic(X_1, 3) = 1$, et donc $\delta = 2$. Par conséquent, la décomposition conduit à la construction de deux sous-problèmes uniquement. P_0 est inchangé et P_1 est formé à partir de P en imposant que $X_1 \neq 1$ et en réduisant les domaines de X_2 et X_3 aux valeurs incompatibles avec $(X_1, 1)$. Dans P_1, $dom(X_1^1) = \{2, 3\}, dom(X_2^1) = \{3\}, dom(X_3^1) = \{3\}$.

Dans ce cas, les deux sous-problèmes de P suivants : P_2 avec $dom(X_1^2) = \{2, 3\}, dom(X_2^2) = \{3\}$ et $dom(X_3^2) = \{1, 2\}$, et P_3 avec $dom(X_1^3) = \{2, 3\}, dom(X_2^3) = \{1, 2\}$ et $dom(X_3^3) = \{1, 2, 3\}$ sont ignorés. Le sous-problème P_3 contient une solution optimale de P : $X_1 = 3, X_2 = 1$ et $X_3 = 3$.

Pour le problème initial, la décomposition élimine 2^3 parmi les 3^3 instanciations complètes possibles alors que pour le problème modifié elle permet d'éliminer plus de la moitié (exactement 16 instanciations complètes).

4.2 Énumération des sous-problèmes

Pour simplifier la présentation, nous considérons dans la suite de l'article que les contraintes sont toutes binaires et normalisées (i.e. elles ont toutes des portées différentes) mais l'extension au cas général est facile[2]. Quand les contraintes sont binaires, faire en sorte qu'une contrainte C avec $scp(C) = \{X, Y\}$ n'ait pas de support pour (X, a) revient simplement à réduire le domaine de Y aux valeurs qui sont incompatibles avec (X, a).

Énumérer tous les sous-problèmes de la décomposition en garantissant que ces sous-problèmes sont disjoints est en fait aussi simple que d'énumérer les valeurs d'un compteur binaire en garantissant qu'au moins δ de ses bits sont à 0.

Soit $I_Y^{X=a}$ les valeurs du domaine $dom(Y)$ qui sont incompatibles avec (X, a) et $C_Y^{X=a}$ les valeurs $dom(Y)$ qui sont compatibles avec (X, a). Par définition, $dom(Y) = I_Y^{X=a} \cup C_Y^{X=a}$ et $I_Y^{X=a} \cap C_Y^{X=a} = \varnothing$. Donc, les sous-domaines I et C forment une partition de chaque domaine ce qui peut être utilisé pour décomposer la recherche de

[1] Cette réduction permet d'obtenir des sous-problèmes disjoints, voir 4.2

[2] La restriction garantit que la réduction du domaine d'une variable voisine de X n'affectera qu'une seule contrainte sur X. Pour étendre au cas général, il suffit de prendre en compte plusieurs fois l'effet de certaines variables.

Y_1	Y_2	Y_3	Y_4
0	$*$	$*$	$*$
1	0	$*$	$*$
1	1	0	$*$
1	1	1	0

(a) recherche avec $\delta = 1$

Y_1	Y_2	Y_3	Y_4
0	0	0	$*$
0	0	1	0
0	1	0	0
1	0	0	0

(b) recherche avec $\delta = 3$

FIG. 1 – Liste des branches à explorer pour $n = 4$ et différentes valeurs de δ

manière systématique. Une recherche exhaustive sur toutes les valeurs d'une variable Y peut s'effectuer en restreignant d'abord le domaine à $I_Y^{X=a}$ et ensuite à $C_Y^{X=a}$. Il s'agit en fait d'un branchement binaire. que l'on peut effectuer récursivement sur les différentes variables. Chaque branche peut être représentée par un mot binaire $b_{Y_1}, \ldots, b_{Y_m}$ où $b_{Y_i} = 0$ indique que le domaine de Y_i est restreint à $I_{Y_i}^{X=a}$ et $b_{Y_i} = 1$ indique que le domaine de Y_i est restreint à $C_{Y_i}^{X=a}$. Une recherche exhaustive sur toutes les valeurs des variables Y va énumérer les 2^m mots binaires (du mot ayant tous les bits à 0 jusqu'au mot avec tous les bits à 1).

Quand $X = a$ est choisi pour décomposer un problème P, le premier sous-problème est P_0 défini par $X = a$ et les autres sous-problèmes sont ceux où $X \neq a$ et où au moins δ variables parmi les m variables Y_i qui supportent (X, a) ont leur domaine réduit à $I_{Y_i}^{X=a}$. Une solution simple pour éviter de parcourir deux fois le même espace de recherche est de se baser sur le branchement binaire précédemment décrit. La condition forçant au moins δ variables parmi les m variables Y_i à avoir leur domaine réduit à $I_{Y_i}^{X=a}$ se traduit directement par «au moins δ bits dans le mot binaire correspondant à une branche doivent être à 0». Cette condition est triviale à mettre en œvre dans un arbre de recherche binaire.

Par exemple, la figure 1 représente les branches à explorer pour deux valeurs différentes de δ et pour $n = 4$ variables. Pour améliorer la lisibilité, $*$ est utilisé comme joker représentant les deux valeurs 0 et 1.

4.3 Complexité

Le branchement binaire proposé dans la section précédente permet d'obtenir des résultats immédiats de complexité. Soient $Y_1, \ldots, Y_m$ les variables qui supportent (X, a) et soient $Z_1, \ldots, Z_r$ les autres variables non encore instanciées. Si l'on n'utilise pas le théorème, la recherche exhaustive des sous-problèmes avec $X \neq a$ va explorer le produit cartésien des domaines ce qui revient à explorer $\prod_{i=1}^{m} |dom(Y_i)|. \prod_{i=1}^{r} |dom(Z_i)|$ instanciations complètes. En appliquant le théorème, au moins δ variables Y doivent avoir leur domaine réduit à $I_Y^{X=a}$. De ce fait, le nombre d'instanciations complètes qui ne seront pas explorées est égal à :

$$\sum_{\substack{S \in 2^{\{Y_i\}} \\ \text{avec } card(S) < \delta}} \prod_{Y_i \in S} |I_{Y_i}^{X=a}|. \prod_{Y_i \notin S} |C_{Y_i}^{X=a}|. \prod_{i=1}^{r} |dom(Z_i)|$$

Cette expression se simplifie si l'on suppose que tous les $C_{Y_i}^{X=a}$ ont la même taille c et que tous les $I_{Y_i}^{X=a}$ ont la même taille i. Dans ce cas, le nombre d'instanciations complètes inexplorées s'exprime par

$$\sum_{j \leq \delta} \binom{n}{j} i^j c^{m-j} \prod_{i=1}^{r} |dom(Z_i)|$$

Pour $\delta = 1$, le nombre d'instanciations élaguées est simplement $i.c^{m-1} \prod_{i=1}^{m} |dom(Z_i)|$. Cela correspond a peu près à la taille du sous-problème dit *cohérent* identifié dans [9] pour le cas CSP. Dans tous les cas, le nombre d'instanciations complètes à explorer en appliquant le théorème est plus petit que le nombre initial d'instanciations complètes à explorer (d'un facteur exponentiel dans le cas général).

4.4 Travaux connexes

Les méthodes de décomposition structurelles classiques combinent la décomposition arborescente d'un graphe avec la recherche par séparation et évaluation [13, 14, 20, 19, 8]. La décomposition arborescente consiste à calculer un pseudo-arbre [10, 2] couvrant l'ensemble des variables par des clusters. Deux clusters sont adjacents dans cet arbre si et seulement si ils partagent certaines variables. Une propriété importante de la décomposition arborescente est que les sous-problèmes associés aux clusters peuvent être résolus de manière indépendante après l'instanciation des variables communes aux clusters. En pratique, l'efficacité des méthodes de décomposition dépend fortement de la structure du graphe de contraintes.

La stratégie de décomposition proposée dans ce papier, inspirée de [9], procède différemment des autres stratégies puisque son principe consiste à décomposer directement le problème initial en des sous-problèmes indépendants sans le calcul de pseudo-arbre et sans instancier aucune variable du problème initial. Chaque sous-problème peut-être résolu indépendemment tout en ignorant une partie de l'espace de recherche de solutions du problème initial. L'inconvénient de cette méthode est que le nombre de sous-problèmes générés peut être important. Cependant, il est intéressant de noter que la décomposition ne se base pas sur la structure du graphe de contraintes.

5 Une approche par élagage

Une autre manière d'exploiter le théorème 3.1 est de l'interpréter comme une règle d'élagage qui peut être intégrée dans n'importe quelle méthode de recherche arborescente pour la résolution du problème Max-CSP. Dans un algorithme basé sur un arbre de recherche binaire [12], chaque nœud ν de l'arbre a deux fils qui sont construit à partir du choix d'une valeur (X, a) : la branche gauche est étiquetée avec l'assignation $X = a$, et la branche droite est étiquetée avec la réfutation $X \neq a$. En considérant l'instance courante au nœud ν, soit a la meilleure valeur aic de X, δ l'écart aic de X et $\{C_i\}$ l'ensemble de contraintes ayant un support pour (X, a). Dès que la branche gauche a été explorée, il est possible d'ajouter une contrainte *dure* locale $atLeastUnsatisfied(\delta, \{C_i\}, (X, a))$ juste avant d'explorer la branche droite de ν. Cette contrainte est définie de sorte qu'elle soit violée dès qu'il n'est plus possible de trouver dans $\{C_i\}$ au moins δ contraintes n'ayant plus de support pour (X, a). Cette contrainte dure doit être retirée dès que la branche droite a été explorée et que l'algorithme fait un retour arrière vers un ancêtre de ν.

Ces contraintes dures ajoutées dynamiquement à l'instance peuvent être simplement utilisées pour forcer un retour arrière et donc éviter l'exploration de portions de l'espace de recherche n'ayant pas de solution. Après chaque propagation, il suffit de vérifier que toutes les contraintes dures ajoutées sont toujours satisfaites. Si ce n'est pas le cas, l'algorithme effectue un retour arrière. Nous désignerons par A tout algorithme de recherche arborescente et par $A\text{-}PC$ (Pruning Constraints) tout algorithme tirant profit de ces contraintes dures ajoutées. Il est intéressant de noter que, à l'exception de certaines heuristiques (telles que celles pondérant les contraintes, par exemple [4]), nous avons la garantie que $A\text{-}PC$ explorera un arbre de recherche inclus dans celui exploré par A.

Une utilisation plus avancée de ces contraintes dures ajoutées consiste à les utiliser dans la propagation afin d'éviter qu'elles ne deviennent violées. Si pour une contrainte $atLeastUnsatisfied(\delta, \{C_i\}, (X, a))$, il est possible de déterminer qu'au plus δ contraintes de $\{C_i\}$ sont encore en mesure de ne pas avoir de support pour (X, a), il est possible de forcer ces δ contraintes à ne plus avoir de support pour (X, a) et donc d'effectuer de nouvelles propagations. Par exemple, si parmi les δ dernières contraintes pouvant ne plus avoir de support pour (X, a), il existe une contrainte binaire portant sur X et une autre variable Y, alors toute valeur de Y cohérente avec (X, a) peut être retirée du domaine de Y. Des structures de données paresseuses (du type *watched literals*) peuvent être utilisées pour effectuer ces propagations efficacement.

Il est important de noter que cette approche par élagage peut être intégrée dans de nombreux algorithmes Max-CSP, y compris les algorithmes hybrides qui combinent énumération et décomposition arborescente.

6 Résultats expérimentaux

Pour montrer l'intérêt pratique de l'approche décrite dans ce papier, nous avons mené une expérimentation en utilisant un cluster de Xeon 3,0GHz avec 1GiB de RAM sous Linux et les instances retenues pour la compétition 2006 de solveurs Max-CSP comme bancs d'essai (voir `http://www.cril.univ-artois.fr/CPAI06/`). Nous avons utilisé l'algorithme classique de séparation et évaluation PFC-MRDAC [17] qui maintient des compteurs d'arc-incohérence dirigés et réversibles de manière à calculer des bornes inférieures à chaque nœud de l'arbre de recherche, et nous nous sommes intéressés à l'impact de l'emploi de l'approche PC (décrite en section 5 - PC pour Pruning Constraints). Nous avons utilisé ici la variante qui permet de forcer les retours-arrières, et n'avons pas (encore) implémenté celle qui permet de filtrer les domaines. Nous n'avons pas non plus implémenté la méthode de résolution par décomposition.

Deux heuristiques de choix de variables ont été retenues. La première est $dom/ddeg$ [3], habituellement considérée pour Max-CSP, qui sélectionne à chaque nœud la variable avec le plus faible ratio *taille du domaine* sur *degré dynamique*. La seconde, notée $dom * gap/ddeg$, implique en plus l'écart aic des variables. Plus précisément, le ratio $dom/ddeg$ est multiplié par l'écart aic de manière à favoriser les variables pour lesquelles il y a un large écart entre la meilleure valeur et la suivante. Nous pensons que cela peut aider à trouver rapidement de bonnes solutions, et plus spécifiquement, augmenter l'efficacité de notre approche. Pour finir, au niveau des valeurs, celle avec le plus petit aic est toujours sélectionnée. Remarquons que cela peut être vu comme un raffinement des compteurs $ic + dac$ usuellement utilisés pour sélectionner les valeurs.

Le protocole utilisé pour notre expérimentation est le suivant : pour chaque instance, nous démarrons avec une borne supérieure initiale[3] fixé à l'infini, et enregistrons le coût de la meilleure solution trouvée (ainsi que l'heure à laquelle elle a été trouvée) dans un temps limite donné (ici, fixé à 1500 secondes). Même si ce protocole ne nous permet pas d'obtenir des résultats utiles pour certaines instances (par exemple, si la même (meilleure) solution est trouvée par les différents algorithmes après quelques secondes), il a l'avantage d'être facilement reproductible et exploitable, que la valeur optimale soit connue ou non.

Tout d'abord, rappelons que nous avons la garantie que PFC-MRDAC-PC visite toujours un arbre qui est plus petit que celui construit par PFC-MRDAC. Cela rend nos comparaisons expérimentales plus faciles. Nous pouvons

[3] Max-CSP peut être vu comme le problème consistant à minimiser le nombre de contraintes violées.

alors faire une première observation générale sur les résultats que nous avons obtenus. Le sur-coût engendré par la gestion des contraintes dures PC se situe de manière générale entre 5% et 10% du temps CPU global. Comme sur les instances aléatoires, notre approche ne permet de sauver qu'un nombre limité de nœuds (comme nous nous y attendions), nous obtenons un comportement similaire avec PFC-MRDAC et PFC-MRDAC-PC. Nous n'avons pas intégré ici ces résultats. D'autre part, concernant les problèmes structurés, la table 1 fournit les résultats sur des instances représentatives et démontre clairement le potentiel de notre approche. Ces instances appartiennent aux séries de problèmes académiques $maxclique$ ($brock$, $p-hat$,san), $kbtree$ (introduit dans [8]), $dimacs$ (ssa) et $composed$, et aux séries d'instances réelles $celar$ ($scen$, $graph$) et $spot$. Le ratio indiqué dans la table correspond au temps CPU de PFC-MRDAC divisé par le temps CPU de PFC-MRDAC-PC. Il correspond soit à une valeur exacte (lorsque les deux méthodes ont trouvé la même borne supérieure) soit à un minorant (dans ce cas, nous utilisons le temps limite 1500 comme minorant). Par exemple, pour l'instance $spot5$-404, nous obtenons 74 comme borne supérieure avec PFC-MRDAC et 73 avec PFC-MRDAC-PC. Puisque chaque nœud visité par PFC-MRDAC-PC est nécessairement visité par PFC-MRDAC, nous savons qu'au moins 1500 secondes sont nécessaires à PFC-MRDAC pour trouver la borne supérieure 73. Nous obtenons donc un ratio d'accélération qui est plus grand que $1500/99 = 15.1$. Remarquons que, comme attendu, les résultats sont plus importants lorsqu'on utilise l'heuristique $dom * gap/ddeg$ (plus de deux ordres de magnitude sur certaines instances) qui de plus, nous permet d'obtenir de meilleures bornes.

7 Conclusion

Dans ce papier, nous avons généralisé pour Max-CSP le principe d'inférence de contraintes disjonctives introduit dans [9] pour CSP. En exploitant l'écart aic entre les deux meilleures valeurs d'une variable, nous avons montré qu'il était possible de garantir le calcul d'une solution optimale tout en élaguant certaines parties de l'espace de recherche. De manière intéressante, ce résultat peut être exploité en terme de décomposition et de retours-arrières/filtrage (en postant des contraintes dures).

Nous avons montré que notre approche intégrée dans un algorithme classique de type séparation et évaluation, permet d'améliorer les performances de la recherche de solutions sur les problèmes structurés. En effet, en l'intégrant dans PFC-MRDAC, nous avons constaté une accélération parfois supérieure à un ordre de grandeur avec l'heuristique $dom/ddeg$ et à deux ordres de grandeur avec l'heuristique $dom * gap/ddeg$.

Nous tenons à rappeler que les méthodes de programma-

		PFC-MRDAC					
		dom/ddeg			dom*gap/ddeg		
		$\neg PC$	PC	$ratio$	$\neg PC$	PC	$ratio$
Instances académiques							
brock-200-1	ub	184	183		184	183	
	cpu	1,490	706	> 2.1	3	57	$= 26.3$
brock-200-2	ub	191	191		191	190	
	cpu	638	92	> 6.9	85	201	> 7.4
brock-400-1	ub	382	380		381	380	
	cpu	23	1,281	> 1.1	19	410	> 3.6
brock-400-2	ub	385	383		383	380	
	cpu	459	376	> 3.9	4	788	> 1.9
composed-25-1-2-1	ub	3	3		6	3	
	cpu	613	332	$= 1.8$	19	846	> 1.7
composed-25-1-25-1	ub	4	4		6	3	
	cpu	92	72	$= 1.2$	14	1,407	> 1
composed-25-1-40-1	ub	3	3		5	4	
	cpu	1,258	1,001	> 1.4	1,462	176	> 8.5
kbtree-9-2-3-5-20-01	ub	6	0		3	0	
	cpu	996	1,333	> 1.1	0	15	> 100
kbtree-9-2-3-5-30-01	ub	13	13		14	4	
	cpu	1,037	1,009	$= 1.0$	1,177	392	> 3
kbtree-9-7-3-5-40-01	ub	27	27		27	27	
	cpu	37	38	$= 1$	98	49	$= 2$
keller-4	ub	162	160		162	160	
	cpu	36	303	> 4.9	1	149	> 10.0
p-hat300-1	ub	293	293		293	293	
	cpu	396	76	$= 5.2$	1,481	224	$= 6.6$
p-hat500-1	ub	493	493		493	492	
	cpu	1,357	652	$= 2.0$	33	717	> 2.0
san-200-0.9-1	ub	174	173		157	155	
	cpu	1,425	1,287	> 1.1	0	888	> 1.6
sanr-200-0.7	ub	185	185		185	184	
	cpu	426	94	$= 4.5$	808	324	> 4.6
ssa-0432-003	ub	82	73		11	2	
	cpu	0	175	> 8.5	46	19	> 78.9
ssa-2670-130	ub	392	390		52	49	
	cpu	1	56	> 26.7	55	1,126	> 1.3
Instances réelles							
graph6	ub	342	341		366	365	
	cpu	216	935	> 1.6	7	406	> 1.0
graph8-f11	ub	161	159		160	160	
	cpu	5	1,299	> 1.1	644	56	$= 11.5$
graph11	ub	576	576		620	620	
	cpu	5	5	$= 1$	677	70	$= 9.6$
scen6	ub	269	269		211	211	
	cpu	69	27	$= 2.5$	20	14	$= 1.4$
scen10	ub	744	744		741	741	
	cpu	34	34	$= 1$	623	56	$= 11.1$
scen11-f12	ub	81	81		66	66	
	cpu	729	395	$= 1.8$	146	35	$= 4.1$
scenw-06-18	ub	215	214		133	131	
	cpu	8	934	> 1.6	231	442	> 3.3
scenw-06-24	ub	98	98		121	117	
	cpu	686	244	$= 2.8$	741	400	> 3.7
scenw-07	ub	353	353		525	524	
	cpu	1,239	471	$= 2.6$	25	8	> 187.5
spot5-28	ub	207	206		196	196	
	cpu	0	31	> 48.3	1	1	$= 1$
spot5-29	ub	52	51		49	48	
	cpu	25	305	> 4.9	807	29	> 51.7
spot5-42	ub	124	124		122	122	
	cpu	900	64	$= 14.0$	1,157	6	$= 192.8$
spot5-404	ub	74	73		76	73	
	cpu	85	99	> 15.1	0	331	> 4.5

TAB. 1 – Meilleure borne (ub = nombre de contraintes violées) et temps CPU (pour l'atteindre) obtenue avec PFC-MRDAC sur des instances structurées, avec (PC) et sans ($\neg PC$) la technique d'élagage. Le temps limite a été fixé à 1500 secondes par instance.

tion dynamique et de décomposition, qui ont récemment reçu beaucoup d'attention, s'appuient toujours partiellement sur la recherche par séparation et évaluation. Cela signifie qu'elles peuvent aussi bénéficier de l'approche développée dans cet article. Pour finir, une perspective de ce travail est de l'étendre aux cadres des CSP pondérés (WCSP) et CSP valués (VCSP).

Remerciements

Ce travail a été soutenu par l'IUT de Lens, le CNRS et le projet ANR Planevo.

Références

[1] M.S. Affane and H. Bennaceur. A weighted arc-consistency technique for Max-CSP. In *Proceedings of ECAI'98*, pages 209–213, 1998.

[2] R.J. Bayardo and D.P. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *Proceedings of IJCAI'95*, pages 558–562, 1995.

[3] C. Bessiere and J. Régin. MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[5] M.C. Cooper, S. de Givry, and T. Schiex. Optimal Soft Arc Consistency. In *Proceedings of IJCAI'07*, pages 68–73, 2007.

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

[7] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. In *Proceedings of IJCAI'05*, pages 84–89, 2005.

[8] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP. In *Proceedings of AAAI'06*, 2006.

[9] E.C. Freuder and P.D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 254–261, 1993.

[10] E.C. Freuder and M.J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of IJCAI'85*, pages 1076–1078, 1985.

[11] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3) :21–70, 1992.

[12] J. Hwang and D.G. Mitchell. 2-way vs d-way branching for CSP. In *Proceedings of CP'05*, pages 343–357, 2005.

[13] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1) :43–75, 2003.

[14] P. Jégou and C. Terrioux. Decomposition and Good Recording for Solving Max-CSPs. In *Proceedings of ECAI'04*, pages 196–200, 2004.

[15] J. Larrosa and R. Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3) :303–326, 2003.

[16] J. Larrosa and P. Meseguer. Partition-Based lower bound for Max-CSP. *Constraints*, 7 :407–419, 2002.

[17] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1) :149–163, 1999.

[18] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for Weighted CSP. In *Proceedings of IJCAI'03*, pages 363–376, 2003.

[19] R. Marinescu and R. Dechter. Advances in AND/OR Branch-and-Bound Search for Constraint Optimization. In *Proceedings of the CP'05 Workshop on Preferences and Soft Constraints*, 2005.

[20] R. Marinescu and R. Dechter. AND/OR Branch-and-Bound for Graphical Models. In *Proceedings of IJCAI'05*, pages 224–229, 2005.

[21] P. Meseguer and M. Sanchez. Specializing russian doll search. In *Proceedings of CP'01*, pages 464–478, 2001.

[22] J.C. Regin, T. Petit, C. Bessiere, and J.F. Puget. New lower bounds of constraint violations for over-constrained problems. In *Proceedings of CP'01*, pages 332–345, 2001.

[23] G. Verfaillie, M. Lemaitre, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of AAAI'96*, pages 181–187, 1996.