

Étude de la dominance dans les CSPs à contraintes de différence

Belaïd Benhamou Mohamed Réda Saïdi

LSIS UMR CNRS 6168

Centre de Mathématiques et d'Informatique

39, rue Joliot Curie - 13453 Marseille cedex 13, France

{Belaid.Benhamou, saidi}@cmi.univ-mrs.fr

Abstract

La détection dynamique et l'élimination des valeurs symétriques dans les CSPs quelconques est en générale une tâche difficile, mais dans le cas des CSPs à contraintes de différence (NECSPs), les conditions de symétrie peuvent être simplifiées. Dans cet article, nous étendons le principe de la symétrie à la dominance dans le cas des CSPs à contraintes de différence et nous montrons comment les valeurs dominées sont détectées et éliminées efficacement à chaque noeuds de l'arbre de recherche. Nous proposons un algorithme de détection de valeurs dominées de complexité linéaire. Nous avons implémenté cet algorithme dans un Forward Checking adapté au cas des CSPs à contraintes de différence. Nous avons comparé cette méthode à la méthode DSATUR sur deux classes de problèmes de coloration de graphes : les aléatoires ainsi que sur des instances issues du challenge de DIMACS. Les résultats montrent que notre méthode est en générale la plus performante.

1 Introduction

Le CSP (pour Constraint Satisfaction Problem en anglais) est un formalisme très utilisé en intelligence artificielle. Plusieurs méthodes et techniques ont été développées pour la résolution des CSPs. La recherche avec backtracking et ses améliorations [10, 15] sont les plus utilisées pour la résolution des CSPs. Les performances de ces méthodes ont été améliorées de différentes façons grâce à l'exploitation de techniques d'élimination des symétries [8, 2, 5, 7, 6, 11].

Dans cet article nous étudions le principe de la dominance dans les CSPs à contraintes de différences (notation NECSPs). Bien qu'ils constituent un cadre restreint

des CSPs quelconques, les NECSPs font partie des classes de problèmes les plus étudiées par la communauté IA: Les NECSPs font partie de la classe des problèmes NP-Complets. En outre, le problème de coloration de graphes peut être représenté par un NECSP. Or, on sait que ce problème est NP-complet [9]. Par ailleurs, les applications de cette famille de problèmes sont nombreuses en intelligence artificielle, on peut citer l'allocation de registres, la cartographie, l'ordonnancement, la planification [1].

La détection dynamique des valeurs symétriques d'un domaine d'une variable pour un CSP quelconque est en générale difficile. Une méthode générale de détection des valeurs symétriques est proposée dans [2], cependant, sa complexité dans le pire des cas est exponentielle. Dans le cas de NECSPs certaines valeurs symétriques peuvent être détectées en un temps linéaire [4].

Nous montrons dans cet article comment la notion de symétrie est étendue à la dominance dans les NECSPs, et comment la condition de la symétrie donnée dans [4] est affaiblie dans le cas où la tentative d'instancier une variable à une valeur de son domaine durant la recherche, mène à un échec. Nous donnons une faible condition de symétrie/dominance. Ce qui nous donne un algorithme performant pour la détection de la dominance et qui détecte plus de symétries que celles détectées par l'algorithme défini dans [4].

Le reste de l'article est organisé comme suite : Nous rappelons dans la section 2, un certain nombre de notions sur les CSPs. Dans la section 3 nous abordons la notion de la dominance et nous montrons comment la condition de la symétrie donnée dans [4] est affaiblie et étendue à la dominance. Puis nous donnons un algorithme efficace pour la détection de la dominance dans les NECSPs. Nous

montrons dans la section 4 comment la dominance est exploitée dans un Forward Checking Simplifié (SFC). Dans la section 5 nous évaluons et comparons les performances de notre méthode à l'aide d'expérimentations sur des problèmes de coloration de graphes. La section 7 conclut ce travail.

2 Le formalisme CSP

Un CSP est quadruple $P = (X, D, C, R)$ où : $X = \{X_1, \dots, X_n\}$ est un ensemble de n variables; $D = \{D_1, \dots, D_n\}$ est l'ensemble des domaines associés aux variables, D_i inclut l'ensemble des valeurs possibles pour la variable X_i ; $C = \{C_1, \dots, C_m\}$ est l'ensemble de m contraintes reliant deux ou plusieurs variables. Une contrainte binaire est une contrainte qui relie exactement deux variables; $R = \{R_1, \dots, R_m\}$ est l'ensemble des relations correspondant aux contraintes de C , R_i représente la liste des tuples permis par la contrainte C_i . Nous nous intéressons qu'aux CSPs qui font intervenir des contraintes binaires. Un CSP binaire \mathcal{P} peut être représenté par un graphe de contraintes $G(X, E)$ où l'ensemble des sommets X est l'ensemble des variables du CSP et chaque arête E connecte deux variables reliées par la même contrainte $C_i \in C$.

Une contrainte binaire est appelée contrainte de différence (Not-Equal constraint en anglais), si elle force les deux variables X_i et X_j à prendre des valeurs différentes (notée $X_i \neq X_j$). Un CSP à contraintes de différence (NECSP) est un CSP dans lequel toutes les contraintes sont des contraintes de différence. Dans la suite, quand il n'y a pas de confusion, nous mentionnerons CSPs pour désigner NECSPs.

Une instantiation $I = (a_1, a_2, \dots, a_n)$ est l'affectation des variables $\{X_1 = a_1, X_2 = a_2, \dots, X_n = a_n\}$ où chaque variable X_i est affectée à une valeur a_i de son domaine D_i . Une contrainte $C_i \in C$ est satisfaite par I si la projection de I sur les variables reliées dans C_i est un tuple de R_i . L'instanciation I est consistante si elle satisfait toutes les contraintes de C , donc I est une solution du CSP. Une instantiation d'un sous-ensemble de variables est appelée instantiation partielle. Une instantiation est totale si elle est définie sur toutes les variables du CSP. Étant donné un CSP, on peut s'intéresser à savoir si le CSP admet au moins une solution (décider de la consistance du CSP), ou à énumérer toutes ses solutions.

Exemple 2.1 Soit le NECSP dont le graphe de contraintes est donné dans la figure 1. Les variables du CSP sont les sommets X_1, \dots, X_5 et les domaines sont inclus dans des patates. Chaque arête du graphe de contraintes connectant deux sommets X_i et X_j , exprime une contrainte de différence entre les variables correspondantes du CSP.

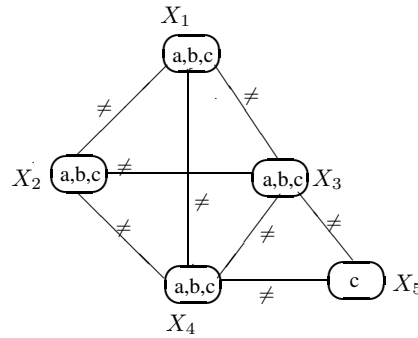


Figure 1: Le graphe de contraintes du NECSP

3 La dominance dans les NECSPs

Les valeurs symétriques d'une variable d'un CSP sont les valeurs qui ont la même pertinence sémantique à participer aux solutions du CSP. Cependant, les valeurs d'un domaine n'ont pas toutes la même pertinence sémantique. Certaines valeurs peuvent avoir probablement plus de chance à participer dans des solutions que d'autres. Les valeurs du premier groupe *dominant* les valeurs du second. Il est important de détecter les valeurs dominantes afin de les considérer en priorité dans l'ordre des affectations. Ce qui permettrait en théorie de réduire l'espace de recherche. Freuder dans [8] introduit la notion de la Substituabilité comme une Interchangeabilité faible. Dans le même esprit, la notion de faible symétrie appelée *Dominance* a été introduite dans [3].

3.0.1 Le principe de la Dominance

Définition 3.1 [Dominance Sémantique] Une valeur a_i domine une autre valeur b_i pour une variable CSP $X_i \in X$ (notation $a_i \succeq b_i$) si et seulement si [Il existe une solution du CSP qui assigne la valeur b_i à la variable $X_i \Rightarrow$ il existe une solution du CSP qui assigne la valeur a_i à X_i].

La valeur b_i participe dans une solution si la valeur a_i participe à une solution; sinon elle n'y participe pas. La valeur b_i peut être alors supprimée de D_i sans affecter la consistance du CSP.

Proposition 3.1 Si $a_i \succeq b_i$ et a_i ne participe à aucune solution du CSP \mathcal{P} , alors b_i ne participe à aucune solution de \mathcal{P} .

Preuve 1 La preuve est une conséquence directe de la définition 3.1

La proposition 3.1 nous permet de supprimer les valeurs dominées $b_i \in D_i$ sans affecter la consistance du CSP, s'il a été montré que la valeur dominante a_i ne participe à aucune solution, sans affecter l'ensemble des solutions du CSP.

La dominance peut être généraliser aux valeurs de différentes variables, mais ici on restreint l'étude à des valeurs d'un même domaine.

Remarque 3.1 Les valeurs symétriques d'un domaine sont les valeurs qui se dominent mutuellement deux à deux. Deux valeurs $a_i \in D_i$ et $b_i \in D_i$ sont symétriques ssi $a_i \succeq b_i$ et $b_i \succeq a_i$. Donc, a_i participe à une solution du CSP ssi b_i participe à une solution du CSP.

3.1 Une condition suffisante pour la dominance

Un algorithme de détection de la symétrie dans les CSPs quelconques est proposé dans [2]. Sa complexité dans le pire des cas est exponentielle. Il est montré dans [4] comment la condition de symétrie peut être simplifiée dans le cas des NECSPs et comment les valeurs symétriques peuvent être éliminées efficacement avec un algorithme simple de complexité linéaire par rapport à la taille du problème. Ce résultat est basé sur la propriété suivante :

Théorème 3.1 Soit a_i et b_i deux valeurs d'un domaine D_i d'un CSP à contraintes de différence \mathcal{P} . Si a_i et b_i apparaissent dans les mêmes domaines des variables non instanciées, alors les deux valeurs sont symétriques.

Preuve 2 Voir [4].

C'est une très simple propriété mais très pratique pour la détection et l'élimination des valeurs symétriques d'un même domaine. En utilisant cette propriété, nous pouvons déduire que les valeurs a et b du domaine de la variable CSP X_1 illustrée dans la Figure 1 sont symétriques. En effet, elles apparaissent toutes les deux dans les domaines de X_2 , X_3 , X_4 et n'apparaissent pas dans le domaine de X_5 . Par un raisonnement similaire, on peut déduire que les valeurs a et b des domaines des variables X_2 , X_3 et X_4 sont symétriques.

Nous donnons maintenant une condition suffisante pour la dominance qui représente la principale contribution de ce travail.

Théorème 3.2 Soient a_i et b_i deux valeurs du domaine D_i correspondant à la variable X_i du CSP à contraintes de différence \mathcal{P} , I une instanciation partielle de \mathcal{P} , et Y l'ensemble des variables non instanciées de \mathcal{P} . Si les deux conditions suivantes:

1. $a_i \in D_j \Rightarrow b_i \in D_j$, pour tout $X_j \in Y$ tel que X_j a une contrainte avec X_i ;
2. $a_i \in D_j \Leftrightarrow b_i \in D_j$ pour chaque variable X_j de Y qui n'a pas de contraintes avec X_i .

sont vérifiées, alors a_i domine b_i dans \mathcal{P} .

Preuve 3 Nous avons à prouver sous les conditions (1) et (2) que si b_i participe à une solution, alors a_i participe à une solution aussi.

Dans ce qui suit, on suppose que si une variable X_i est instanciée à une valeur d_i de son domaine D_i , alors on supprime des domaines des variables voisines à cette variable (variables partageant une contrainte avec X_i) les valeurs incompatibles avec l'instanciation $X_i = d_i$.

Soit $I_{X_i=b_i}$ une solution du CSP \mathcal{P} où la variable X_i est instanciée à b_i , nous avons à montrer l'existence d'une solution où la variable X_i est instanciée à a_i . Soit \mathcal{P}' le CSP obtenu à partir de \mathcal{P} en supprimant la valeur b_i des domaines des variables ayant une contrainte avec X_i et où la valeur a_i n'apparaît pas. Cette opération ne produit pas de domaines vides, puisque $I_{X_i=b_i}$ est une solution du CSP \mathcal{P} . Ce qui veut dire que tous les domaines qui seront réduits contiennent au moins deux valeurs avant la suppression de la valeur b_i . Le CSP \mathcal{P}' est un sous-CSP de \mathcal{P} , il a le même ensemble de variables, le même ensemble de contraintes, mais les domaines réduits deviennent des sous-ensembles des domaines de départ à partir desquels ils résultent. Le CSP \mathcal{P}' "est plus contraint" que \mathcal{P} . Ce qui signifie que, chaque solution de \mathcal{P}' est une solution de \mathcal{P} . Rappelons que, par hypothèse, les conditions (1) et (2) sont vérifiées dans \mathcal{P} , ce qui nous donne que les valeurs a_i et b_i deviennent symétriques dans le CSP \mathcal{P}' . Les conditions du théorème 3.1 sont alors vérifiées, et donc la valeur a_i participe à une solution de \mathcal{P}' ssi la valeur b_i participe à une solution de \mathcal{P}' . Par ailleurs, le CSP $\mathcal{P}_{X_i=b_i}$ obtenu à partir de \mathcal{P} en considérant l'affectation $X_i = b_i$ est identique au CSP $\mathcal{P}'_{X_i=b_i}$ obtenu à partir de \mathcal{P}' en considérant l'affectation $X_i = b_i$. Nous en déduisons que $I_{X_i=b_i}$ est une solution de \mathcal{P}' , et qu'il existe une solution $I'_{X_i=a_i}$ de \mathcal{P}' où X_i est instanciée à a_i , puisque a_i et b_i sont symétriques dans \mathcal{P}' . Comme le CSP \mathcal{P}' est plus contraint que le CSP \mathcal{P} , alors $I'_{X_i=a_i}$ est une solution de \mathcal{P} . Nous avons prouvé l'existence d'une solution où a_i est affectée à X_i , donc a_i domine b_i dans \mathcal{P} . (CQFD)

Remarque 3.2 Ce théorème est vrai pour tout NECSP dont aucune variable n'est instanciée. Cependant, comme indiquée au début de la preuve précédente, lorsque certaines variables du NECSP sont déjà instanciées, il faut au préalable supprimer toutes les valeurs des domaines des variables voisines à celles-ci, qui sont incompatibles avec l'instanciation partielle.

Exemple 3.1 Dans le domaine de la variable X_3 de la Figure 1, la valeur a domine la valeur c .

3.2 Affaiblissement de la condition de dominance

Avant d'introduire la condition affaiblie, on définit la notion d'arbre d'affectation et d'arbre d'échec correspondant

à une méthode de recherche énumérative utilisée pour la preuve de consistance du CSP considéré.

Définition 3.2 On appelle "arbre d'affectation" d'un CSP \mathcal{P} correspondant à une méthode de recherche et un ordre d'instanciation des variables, un arbre qui regroupe l'historique de toutes les affectations de variables effectuées durant la preuve de consistance du CSP, où tous les noeuds représentent les variables du CSP et où les arêtes partant d'un noeud X_i sont étiquetées par les différentes valeurs utilisées pour instancier la variable correspondante X_i .

La racine de l'arbre est la première variable dans l'ordre d'instanciation. Dans cet article, la méthode de résolution considérée est le Forward Checking [10]. On considère uniquement les arbres d'affectations correspondant à cette méthode.

Dans un arbre d'affectation d'un CSP, un chemin reliant la racine de cet arbre à un noeud définit une instanciation partielle du CSP. Les variables de l'instanciation partielle sont les noeuds de ce chemin. Le dernier noeud du chemin correspond à la dernière variable instanciée ou à une variable ayant un domaine vide.

Nous associons à chaque instanciation partielle, correspondant à un chemin dans l'arbre d'affectation, un arbre d'échec défini comme suite :

Définition 3.3 Soit T un arbre d'affectation d'un CSP \mathcal{P} , $I = (a_1, a_2, \dots, a_i)$ une instanciation partielle inconsistante des variables X_1, X_2, \dots, X_i correspondant au chemin $\{X_1, X_2, \dots, X_i\}$ dans T . On appelle arbre d'échec d'une instanciation I , le sous-arbre de T noté $T_{I=(a_1, a_2, \dots, a_i)}$ tel que:

1. La racine de l'arbre T et la racine du sous-arbre $T_{I=(a_1, a_2, \dots, a_i)}$ sont reliées par le chemin correspondant à l'instanciation I ;
2. Toutes les variables CSP correspondant aux feuilles de $T_{I=(a_1, a_2, \dots, a_i)}$ ont des domaines vides.

Exemple 3.2 Considérons le CSP de la Figure 1 et appliquons l'algorithme Forward Checking par rapport à l'ordre d'instanciation des variables $\{X_1, X_2, X_3, X_4, X_5\}$. La Figure 2 illustre l'arbre d'affectation du CSP. Si nous prenons l'instanciation partielle $I = (b, a)$ qui assigne X_1 à la valeur b et X_2 la valeur a , alors l'arbre d'échec $T_{I=(b, a)}$ de l'instanciation I est donnée dans la Figure 2 (partie entourée).

Nous pouvons maintenant donner la condition affaiblie pour la dominance. L'idée est d'affaiblir la condition de dominance du théorème 3.2 quand une instanciation partielle inconsistante est générée durant la recherche. C'est à

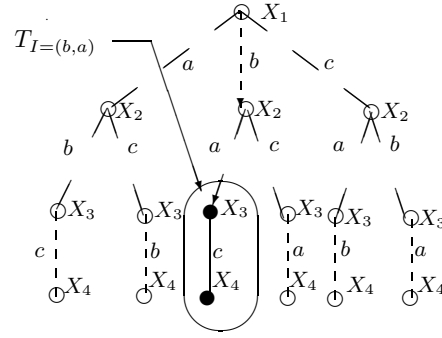


Figure 2: Arbre d'affectation T et arbre d'échec $T_{I=(b,a)}$

dire, la condition du théorème 3.2 est restreinte aux seules variables impliquées dans l'arbre d'échec parmi toutes les variables non instanciées.

Théorème 3.3 Soient $\mathcal{P}(X, D, C, R)$ un CSP, $a_i \in D_i$ et $b_i \in D_i$ deux valeurs du domaine D_i de la variable courante X_i en cours d'instanciation, $I_0 = (a_1, \dots, a_{i-1})$ une instanciation partielle des $i - 1$ variables instanciées avant X_i telle que l'extension $I = I_0 \cup \{a_i\} = (a_1, \dots, a_{i-1}, a_i)$ est inconsistante, $T_{I=(a_1, \dots, a_{i-1}, a_i)}$ est l'arbre d'échec de I et $\text{Var}(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ l'ensemble des variables correspondant aux noeuds de $T_{I=(a_1, \dots, a_{i-1}, a_i)}$. Si les deux conditions suivantes :

1. $b_i \in D_j \Rightarrow a_i \in D_j$, pour tout $X_j \in \text{Var}(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ tel que X_j a une contrainte avec X_i .
2. $a_i \in D_j \Leftrightarrow b_j \in D_i$ pour toute autre variable X_j de $\text{Var}(T_{I=(a_1, \dots, a_{i-1}, a_i)})$ qui n'a pas de contraintes avec X_i

sont vérifiées, alors l'extension $J = I_0 \cup \{b_i\} = (a_1, \dots, a_{i-1}, b_i)$ est inconsistante.

Preuve 4 Soit $\mathcal{P}'(X', D', C', R')$ un sous-CSP du CSP $\mathcal{P}(X, D, C, R)$ tel que $X' = \text{Var}(T_{I=(a_1, \dots, a_{i-1}, a_i)}) \cup X_i$ et $D' \subseteq D$, $C' \subseteq C$ et $R' \subseteq R$ sont les restrictions de D, C et R aux variables de X' . Par hypothèse, $T_{I=(a_1, \dots, a_{i-1}, a_i)}$ est un arbre d'échec de I dans \mathcal{P} . Ce qui implique que l'affectation de X_i à la valeur a_i mène à un échec dans \mathcal{P}' . En d'autres mots, a_i ne participe pas à aucune solution de \mathcal{P}' . Par hypothèse, les valeurs a_i et b_i vérifient les conditions du théorème 3.2 quand on se restreint aux variables de $\text{Var}(T_{I=(a_1, \dots, a_{i-1}, a_i)})$. Ce qui signifie que a_i domine b_i dans le CSP \mathcal{P}' . Par application de la proposition 3.1, on déduit que la valeur b_i ne participe à aucune solution de \mathcal{P}' . Ceci implique que l'instanciation partielle $J = I_0 \cup \{b_i\} = (a_1, \dots, a_{i-1}, b_i)$ est inconsistante dans \mathcal{P} . (CQFD)

Cette dernière propriété est un affaiblissement de la condition de la dominance 3.2 quand l'instanciation partielle courante mène à un échec. Le cas d'instanciations partielles inconsistantes est très important, car il nous permet d'élaguer l'arbre de preuve de la consistance du CSP comme le stipule la Proposition 3.1.

Certaines dominances non capturées par le Théorème 3.2 peuvent l'être grâce à cette condition affaiblie. Considérons par exemple le CSP de la Figure 1. Si nous prenons l'instanciation partielle $I = (b, a)$ des variables X_1 et X_2 . On suppose que X_2 est la variable courante, alors les deux valeurs a et c du domaine de X_2 se dominent mutuellement (symétriques) par application du Théorème 3.3, alors qu'elles ne sont pas symétriques par application des théorèmes 3.1 et 3.2. La branche correspondant à l'affectation X_2 à c ne sera pas explorée dans l'arbre de la preuve de consistance grâce au Théorème 3.3. Ceci définit donc, des coupures de dominance que nous exploitons dans la section 4 pour réduire l'arbre de recherche de consistance du CSP.

3.3 Détection de la Dominance

Maintenant nous abordons le problème de la détection de la dominance. La détection de la dominance est basée sur la condition du théorème 3.3. L'algorithme présenté dans la Figure 3 calcule les valeurs dominées par une valeur a_i d'un domaine D_i donné par application des conditions du théorème 3.3. Ces valeurs forment une classe de dominance de a_i que nous notons $Cl(a_i)$.

```

procedure weak_dominance( $a_i \in D_i, Var(T_{I=(a_1, \dots, a_i)})$ )
, var  $Cl(a_i)$ :classe;
input: une valeur  $a_i \in D_i$ , un ensemble de variables
 $Var(T_{I=(a_1, \dots, a_i)})$ 
Output: la classe  $Cl(a_i)$  des valeurs dominées par  $a_i$ .
begin
   $Cl(a_i) := \{a_i\}$ 
  for each  $d_i \in D_i - \{a_i\}$  do
    for each domaine  $D_k$  des variables de
       $Var(T_{I=(a_1, \dots, a_i)})$ 
      if ( $c_{ik} \in C$  and ( $a_i \in D_k \Rightarrow d_i \in D_k$ ))
      or
      ( $c_{ik} \notin C$  and ( $a_i \in D_k \Leftrightarrow d_i \in D_k$ ))
      then  $Cl(a_i) := Cl(a_i) \cup \{d_i\}$ 
end

```

Figure 3: L'algorithme de détection de la dominance dans les NECSPs

Complexité: Soient n le nombre de variables du NECSP, et d la taille du plus grand domaine. Il est facile de voir que l'algorithme de la Figure 3 peut exécuter au plus d fois la première boucle et au plus n fois la seconde boucle. Donc, il calcule la classe $Cl(d_i)$ des valeurs dominées par a_i avec une complexité $\mathcal{O}(nd)$ dans le pire des cas. En théorie, cet algorithme a la même complexité dans le pire des cas que l'algorithme décrit dans [4]. Cependant, cet algo-

rithme détecte un plus grand nombre de symétries, puisque les symétries déjà détectées par l'ancien algorithme ne représente qu'une partie de celles-ci. De plus, notre algorithme détecte la dominance.

4 Exploitation de la dominance dans les NECSPs

Maintenant, nous montrons comment la propriété de la dominance donnée dans le Théorème 3.3 est exploitée afin d'améliorer les performances d'un algorithme de type backtrack utilisée pour la résolution des NECSPs. Cette propriété peut être exploitée dans toute méthode énumérative de résolution. Nous avons choisi d'implémenter une méthode de Forward Checking simplifié (notée SFC) à laquelle nous ajoutons la propriété de dominance afin d'améliorer les performances.

Le principe du Forward Checking [10] est basée sur le filtrage des domaines des variables non instanciées en tenant compte des valeurs des variables déjà instanciées. Dans le cas des NECSPs, le filtrage est simplifié. Si la variable courante X_i est instanciée à la valeur d_i , le filtrage consiste à enlever la valeur d_i des domaines des variables future (variables non instanciées) ayant une contrainte avec X_i .

Le Théorème 3.3 nous permet d'élaguer $k-1$ branches de l'arbre de recherche s'il y a k valeurs dominées par une valeur dominante, si cette dernière ne participe à aucune solution. Si $Cl(d_i)$ dénote la classe des valeurs du domaine D_i dominées par d_i , alors nous considérons uniquement la valeur d_i , puisque les autres valeurs de $Cl(d_i)$ sont redondantes.

La Figure 4 illustre la procédure SFC combinées avec la propriété de la dominance du théorème 3.3 (notation SFC-weak-dom). Elle permet de tester la consistance d'un NECSP donné. Cependant, elle peut être adaptée facilement pour le calcul de toutes les solutions non symétriques/non dominées. La structure $future(X_i)$ représente l'ensemble des variables non instanciées, et $next-variable$ est une fonction qui code l'heuristique(DomDeg). Elle consiste à minimiser le ratio

$$r = \frac{|D_j|}{Degree(X_j)}$$

où $Degree(X_j)$ dénote le nombre de variables reliées à la variable X_j par une contrainte, pour sélectionner la prochaine variable à instancier. Dans la suite, SFC dénotera la méthode SFC combinée avec l'heuristique (DomDeg).

5 Expérimentations

Nous allons évaluer les performances de notre implémentation. Les testes sont faits sur deux classes

```

Procedure SFC-weak-dom( $D, I, i, \text{var } VFT : \text{liste}$ );
input: un ensemble de domaines  $D$ ,
 $I = (d_1, \dots, d_i)$  une instantiation partielle des variables
 $\{X_1, \dots, X_i\}$ ;  $i$  l'index de la variable courante
et  $VFT$  l'ensemble des variables  $\text{Var}(T_I = (d_1, \dots, d_i))$  de l'arbre
d'échec  $T_I = (d_1, \dots, d_i)$  (au début  $VFT$  est vide).
var empty:boolean;
var  $VFT\_tmp$ :liste;
var  $VFT\_old$ :liste;
begin
  if  $i = n$  then  $[d_1, d_2, \dots, d_i]$  est une solution.printf( $I$ ), stop
  else
    begin
      empty:=false;
       $VFT\_tmp := VFT$ ;
       $VFT\_old := VFT$ ;
      for each  $X_j \in X$ , telle que  $C_{ij} \in C, X_j \in \text{future}(X_i)$  do
        if not(empty) and  $d_i \in D_j$  then
          begin
             $D_j := D_j - \{d_i\}$ ;
            if  $D_j = \emptyset$  then
              begin
                annuler les effets du filtrage;
                ajouter( $X_j, VFT$ );
                empty:=true;
              end
            end
          end
        if not(empty) then
          begin
             $X_{i+1} := \text{next-variable}(X_i)$ 
            repeat
              prendre  $d_{i+1} \in D_{i+1}$ 
               $D_{i+1} := D_{i+1} - d_{i+1}$ 
               $I = [d_1, d_2, \dots, d_i, d_{i+1}]$ ;
               $VFT\_tmp := VFT \cup VFT\_tmp$ ;
               $VFT := VFT\_old$ ;
              SFC-weak-dom( $D, I, i + 1, VFT$ );
              weak_dominance( $d_{i+1} \in D_{i+1}, VFT, Cl(d_{i+1})$ );
               $D_{i+1} := D_{i+1} - Cl(d_{i+1})$ ;
            until  $D_{i+1} = \emptyset$ 
          end
           $VFT := VFT\_tmp$ ;
          ajouter( $X_i, VFT$ );
        end
      end
    end
  end

```

Figure 4: La méthode SFC combinée avec l'algorithme de détection des dominances

de problèmes de coloration de graphes : des graphes aléatoires générer à l'aide d'un programme; des benchmarks issus du second challenge de DIMACS (<http://dimacs.rutgers.edu/Challenges/>). Le problème de coloration de graphes s'exprime de manière triviale sous forme d'un NECSP. Nous testerons et comparons la méthode SFC combinée avec la propriété de symétrie définie dans [4] (SFC-sym), notre méthode SFC combinée avec la propriété de la dominance (SFC-weak-dom) et une version optimisée de la méthode DSATUR [16]. Les indicateurs de complexité sont le nombre de noeuds et le temps CPU. Le code source est écrit en C et compilé sur une machine équipée d'un P4 2.8 GHz - RAM 1 Go.

5.1 Les problèmes de coloration de graphes aléatoires

Les problèmes sont générés en fixant les paramètres suivants : n le nombre de sommets (les variables); $NbColors$ le nombre de couleurs (les valeurs des domaines) et d la densité du graphe des contraintes.

Pour $n, NbColors$ et d fixés, le nombre de noeuds $NbNodes$ ainsi que le temps d'exécution $Time$ sont calculés sur la moyenne des résultats obtenus à partir de 100 instances générées aléatoirement.

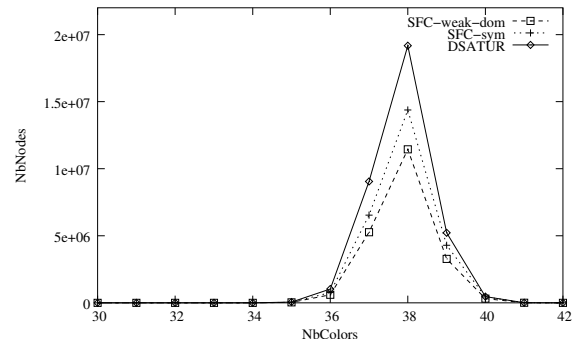


Figure 5: Courbes représentant le nombre de noeuds moyen

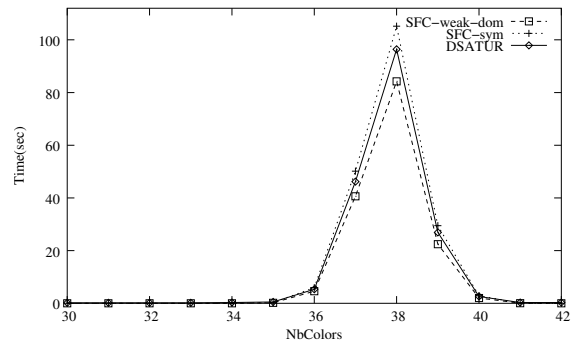


Figure 6: Courbes représentant le temps CPU moyen

Les Figures 5 et 6 donnent les performances des méthodes DSATUR, SFC-sym et SFC-weak-dom en considérant le nombre des noeuds moyen générés, le temps CPU moyen (en secondes) sur les problèmes aléatoires où le nombre de variable est fixé à $n = 100$ et la densité à $d = 0.9$. Nous pouvons voir que la méthode SFC-weak-dom, génère en moyenne moins de noeuds que DSATUR et SFC-sym, et passe moins de temps que les deux autres méthodes pour résoudre les problèmes. Ce qui prouve que SFC-weak-dom détecte et élimine le plus de symétries. DSATUR élimine uniquement des symétries triviales. SFC-sym détecte plus de symétries que DSATUR mais moins que SFC-dom-weak. SFC-weak-dom est la seule méthode

à capturer la dominance, grâce à l'algorithme de la Figure 3. Ces remarques sont aussi confirmées sur les benchmarks de DIMACS où le gain est plus important.

5.2 Les benchmarks de DIMACS

La Table 1 montrent les résultats des différentes méthodes sur quelques problèmes de coloration de graphes issus du challenge DIMACS. On donne le nombre de noeuds ainsi que le temps CPU pour chaque méthode. Nous cherchons pour chaque problème le nombre minimal k de couleurs nécessaires pour la coloration des sommets du graphe correspondant (le nombre chromatique). Déterminer le nombre chromatique consiste à trouver une coloration du graphe avec k couleurs (existence d'une k -coloration du graphe); et à prouver qu'il n'est pas possible de trouver une $k - 1$ coloration du graphe. Le symbole "-" signifie que la méthode correspondante n'a pas retourné de réponse après une heure de calcul.

On peut remarquer que seule la méthode SFC-weak-dom est capable de résoudre le problème "R500.1c". Nous pouvons voir que la méthode SFC-dom-weak est en générale meilleur que DSATUR et SFC-sym, et que SFC-sym est meilleur que DSATUR sur ces problèmes. DSATUR est le moins performant puisque qu'il n'a pas résolu 9 des 22 benchmarks proposés.

Pb instances ($V - E$)	k	DSATUR		SFC-SYM		SFC-dom-weak	
		N	T	N	T	N	T
queen8_8 (64-728)	9	1581661	4.3	1368441	6.1	1353680	6.1
queen8_12 (96-1368)	12	162	0.0	460	0.0	460	0.0
myciel5 (47-236)	6	378310	0.6	72966	0.2	21278	0.0
myciel6 (95-755)	7	-	-	83157279	556.0	29754513	190.2
le450_5a (450-5714)	5	-	-	1408	0.1	1395	0.1
le450_5b (450-5734)	5	-	-	19884	0.6	19763	0.5
le450_25a (450-8260)	25	425	0.1	450	0.1	450	0.1
le450_25b (450-8263)	25	425	0.0	450	0.1	450	0.1
1-FullIns_3 (30-100)	4	37	0.0	51	0.0	50	0.0
1-FullIns_4 (93-593)	5	-	-	8885	0.0	1368	0.0
2-FullIns_3 (52-201)	5	156663424	193.6	678	0.0	359	0.0
qg_order30 (900-26100)	30	1680	0.2	1169	0.2	1162	0.2
qg_order40 (1600-62400)	40	-	-	12089785	302.0	10814593	266.6
school_1 (385-19095)	14	371	0.2	568	0.4	555	0.4
school_nsh (352-14612)	14	338	0.2	352	0.4	352	0.4
wap05a (905-43081)	50	855	1.3	905	1.4	905	1.4
mug88_25 (88-146)	4	-	-	22643	0.0	1631	0.0
mug100_25 (100-166)	4	-	-	99917	0.2	515	0.0
ash608 (1216-7844)	4	10242	0.5	1742	0.5	1707	0.5
ash958 (1916-12506)	4	-	-	10252	2.2	7167	1.6
R125.5 (125-3838)	36	1357573	9.1	55952	0.4	1051	0.0
R500.1c (500-121275)	84	-	-	-	-	28044984	3096.0

Table 1: Les benchmarks de DIMACS

6 Liens avec d'autres travaux

- La méthode *Arbre de Groupes d'Équivalence* (noté GE-tree) [14] élimine toutes les symétries de valeurs à chaque noeud de l'arbre de recherche d'un CSP quelconque. Cette méthode peut, éventuellement, être utilisée pour éliminer les symétries de valeurs dans les NECSPs durant la recherche, mais elle peut être coûteuse, car sa complexité théorique dans le pire des cas à chaque noeud de l'arbre est proche de $O(n^4 d^4)$. Alors que notre méthode permet de détecter les classes de symétries d'un domaine en $O(nd^2)$ et détecte la dominance qui n'est pas considérée par la méthode GE-tree.
- Récemment dans [13, 12], Puget a étudié les symétries de variables liées par une contrainte global *Alldiff*. Cette contrainte est un cas particulier de NECSPs, puisque la contrainte *Alldiff* peut être exprimée par un NECSP dont le graphe des contraintes est complet. La technique d'élimination de symétries utilisée dans ce travail est statique, elle consiste à ajouter un nombre linéaire de contraintes au problème afin de casser toutes les symétries de variables de ce dernier. Notre approche est dynamique et s'applique à une classe plus large de problèmes (les NECSPs). Par ailleurs, notre méthode peut être combinée avec celle de Puget. Il serait donc intéressant d'étudier les conséquences de cette association.

7 Conclusion

Dans ce papier, nous avons étendu le principe de la symétrie à la dominance et nous avons donné une nouvelle condition, plus faible valable pour la symétrie et la dominance dans le cas des CSPs à contraintes de différence, quand une instanciation partielle inconsistante est générée. L'algorithme basé sur cette condition est plus efficace, détecte la dominance et plus de symétries. Nous avons donné un exemple d'implémentation de cet algorithme avec une version simplifié du Forward Checking. Les expérimentations sur des problèmes de coloration de graphes montrent que l'étude de la dominance est d'un grand apport dans la résolution des NECSPs.

Ce travail soulève plusieurs points à discuter qui pourront être développés dans le futur. Certains points nous semblent particulièrement intéressants : le premier consiste à combiner certaines méthodes de décomposition de CSP avec notre méthode afin d'améliorer les performances dans le cas des CSPs à contraintes de différence. Un autre point à voir est d'étendre le principe de la dominance aux valeurs de différentes variables, puis étendre nos résultats à d'autres CSPs plus généraux.

References

- [1] A.Ramani, F.A.Aloul, I.L.Markov, and K.A.Sakallak. Breaking instance-independent symmetries in exact graph coloring. *In DATE*, pages 324–329, 2004.
- [2] B. Benhamou. Study of symmetry in constraint satisfaction problems. *In PPCP'94*, 1994.
- [3] B. Benhamou. Theoretical study of dominance in constraint satisfaction problems. *6th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA-94), Sofia, Bulgaria, september*, pages 91–97, 1994.
- [4] B. Benhamou. Symmetry in not-equals binary constraint networks. *In the working notes*, 2004.
- [5] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. *In KR'96*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.
- [6] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. *In CP'01*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.
- [7] F. Focacci and M. Milano. Global cut framework for removing symmetries. *In CP'01*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.
- [8] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.
- [9] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of np-completeness*, w.h. freeman. Technical report, 1979.
- [10] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, pages 263–313, 1980.
- [11] J.F. Puget. Symmetry breaking revisited. *In CP'02*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2002.
- [12] J.F. Puget. Breaking symmetries in all different problems. *SymCon'04 : 4th International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.
- [13] J.F. Puget. Breaking symmetries in all different problems. *In, proceedings of IJCAI*, pages 272–277, 2005.
- [14] Colva M. Rouney-Dougal, Ian P. Gent, Tom Kesley, and steve A. Linton. Tractable symmetry breaking using restricted search trees. *In proceedings of ECAI-04*, 2004.
- [15] D. Sabin and E. Freuder. Understanding and improving the mac algorithm. *In CP97*, pages 167–181, 1997.
- [16] Edward C. Sewell. An improved algorithm for exact graph coloring. *DIMACS series on Discrete Mathematics and Theoretical Computer Science*, 1995.