Une nouvelle approche pour le test d'inconsistance de CSP

Belaïd Benhamou, Mohamed Saïdi

Une nouvelle approche pour le test d'inconsistance de CSP

Belaïd Benhamou **Mohamed Réda Saïdi**

Laboratoire des Sciences de l'Information et des Systèmes (LSIS)
Centre de Mathématiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France
{Belaid.Benhamou,saidi}@cmi.univ-mrs.fr

Résumé

Tester la consistance de CSP est en théorie un problème NP-Complet. Il existe deux familles de méthodes pour le test de consistance. La première famille regroupe les méthodes complètes qui font un parcours exhaustif de l'espace de recherche de solutions. Ces méthodes ont l'avantage de prouver l'inconsistance de CSP, cependant leur complexité croît exponentiellement avec la taille du problème. La seconde famille inclut les méthodes incomplètes qui font de la recherche local sur l'espace de recherche de solutions. Ces méthodes ont été utilisées efficacement pour trouver des solutions pour des problèmes de grande taille que les méthodes complètes ne peuvent résoudre. Le principal inconvénient des méthodes incomplètes reste tout de même leur incapacité de prouver l'inconsistance d'une instance CSP. L'un des challenges mis en avant par la communauté CP (Selman et al. 1997) est de proposer des méthodes incomplètes efficaces pour la preuve d'inconsistance de CSP. Le travail que nous présentons ici est une contribution à ce difficile challenge. Nous introduisons une nouvelle méthode incomplète pour le test d'inconsistance qui se base sur la notion de dominance entre CSPs et la coloration de la micro-structure du CSP. Nous avons expérimenté la méthode sur des instances CSP générées aléatoirement et les résultats obtenus sont très encourageant.

Abstract

Checking CSP consistency is shown, in theory, to be an NP-complete problem. There is two families of methods for CSP consistency checking. The first family holds the complete methods which make an exhaustive search on the solution space. These methods have the advantage to prove CSP inconsistency, but their complexity grows exponentially when the problem size increases. The second family includes the incomplete methods that make a local search on the solution space. These methods have been efficiently used to find solutions for large size consistent CSPs that complete methods can not solve. One major drawback of the incomplete methods, is their inability to prove CSP inconsistency. One of the challenges that have been put forward by the CP community (Selman et al. 1997) is to provide incomplete methods that can deal with CSP inconsistency efficiently. The work that we present here, is a contribution towards an answer to this hard challenge. We introduce a new incomplete method for CSP inconsistency checking that is based on both a new notion of dominance between CSPs and a coloration of the CSP micro-structure. We experimented the method on randomly generated CSP instances and the results obtained are very promising.

1 Introduction

Le travail que nous présentons ici est une contribution pour résoudre le challenge 5 proposé dans [13]. Ce challenge consiste à exploiter des méthodes incomplètes efficaces pour la preuve d'inconsistance de CSP. Ce challenge est motivé par les grandes avancées dans le domaine de la résolution des problèmes SAT et CSP. En effet, plusieurs problèmes de grande taille réputés difficiles pour les méthodes complètes du type Davis-Putnam-Logemann-Loveland [5], ont été résolu efficacement en utilisant des méthodes incomplètes [14, 8] qui sont basées sur la recherche locale dans l'espace de solutions.

Cependant, les méthodes de recherche locale ne peuvent pas être utilisées pour prouver l'inconsistance de CSP et perdent dans ce cas, une grande partie de leur utilité dans la résolution de CSP.

Bien que connu depuis 1997, très peu de travaux abordent ce challenge, parmi eux : les travaux [10, 7, 11]

ou plus récemment [1, 2] s'intéressent à ce challenge dans le cadre du problème SAT, et principalement deux autres travaux s'intéressent à ce challenge dans le cadre du problème CSP [6, 3].

Nous proposons dans ce travail une nouvelle méthode incomplète pour le test d'inconsistance de CSP binaires. Cette méthode est basée sur une nouvelle notion appelée *dominance* entre CSPs et la coloration de la micro-structure du CSP. Nous allons prouver que l'approche proposée dans [6] est un cas particulier de notre méthode. Nous avons implémenté et expérimenté la nouvelle méthode sur plusieurs instances CSP inconsistantes générées aléatoirement et les résultats obtenus sont très encourageant et montrent que notre méthode a le meilleur taux de détection parmi les méthodes testées.

Ce papier est organisé de la façon suivante : un petit rappel sur les CSPs est donné dans la deuxième section. Nous introduisons dans la troisième section les bases de la nouvelle méthode et nous montrons que l'approche utilisée dans [6] est un cas particulier de notre méthode. Nous évaluons l'efficacité de la nouvelle méthode dans la quatrième section où, à partir des expérimentations faites sur des instances CSP aléatoires, nous comparons plusieurs méthodes à la notre. Nous concluons ce travail dans la dernière section.

2 Préliminaires

Un CSP est une structure $P(n) = (V, D, C)$ où : $V = \{v_1, ..., v_n\}$ est un ensemble de n variables ; $D = \{D_1, ..., D_n\}$ est l'ensemble fini discret des domaines associé aux variables CSP, D_i inclut l'ensemble des valeurs possibles de la variable CSP v_i,

$C = \{C_1, ..., C_m\}$ est l'ensemble de m contraintes, chacune impliquant un sous ensemble de variables CSP. Les contraintes sont données en extension, chaque contrainte $C_i \in C$ est représentée par une liste de tuples de valeurs permises. Une contrainte binaire est une contrainte qui implique au plus deux variables. Un CSP binaire $P(n)$ (un CSP où toutes ses contraintes sont des contraintes binaires) peut être représenté par un graphe de contraintes $G(V, E)$ où l'ensemble des sommets V est l'ensemble des variables CSP et chaque arête $(v_i, v_j) \in E$ connecte les variables v_i et v_j impliquées dans la contrainte $C_{ij} \in C$. La micro-structure [9] d'un CSP binaire $P(n)$ est un graphe $M_{P(n)}(V \times D, \acute{E})$, où chaque arête de $\acute{E}$ correspond soit à un couple permis par une contrainte spécifique ou à un couple permis car il n'y a pas de contrainte entre les deux variables associées.

Une contrainte *Alldifferent* est une contrainte globale qui force toutes les variables impliquées par la contrainte à prendre des valeurs différentes. La contrainte $Alldifferent$ est une contrainte globale très étudiée par la communauté CP, plusieurs algorithmes dédiés existent dans la littérature.

Une instanciation $I = (\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, \ldots, \langle v_n, d_n\rangle)$ est l'affectation des variables $\{v_1 = d_1, v_2 = d_2, \ldots, v_n = d_n\}$ où une valeur d_i du domaine D_i est assignée à la variable v_i.

L'instanciation I est consistante si elle satisfait toutes les contraintes de C, donc I est une solution du CSP.

3 La méthode Dominance-Coloration

Avant de décrire les bases théoriques de notre méthode, nous rappelons le principe de la méthode introduite dans [6].

Le principal résultat théorique sur lequel repose cette méthode est donné dans le corollaire suivant.

Corollaire 1 *[6] Si la micro-structure $\mathcal{M}_{P(n)}$ d'un CSP $P(n)$ ayant n variables, peut être colorée avec $n-1$ couleurs, alors le CSP $P(n)$ est inconsistant.*

Pour illustrer comment ce corollaire peut-être utilisé pour montrer l'inconsistance d'un CSP, prenons une instance du problème des pigeons avec trois pigeons et deux pigeonniers que nous notons $Pigeons(3)$. Rappelons que ce problème consiste à mettre les pigeons dans les pigeonniers, de telle sorte que chaque pigeonnier ne peut contenir tout au plus un pigeon. La figure 1 donne le graphe de contraintes du CSP représentant le problème $Pigeons(3)$ sur la partie gauche de la figure et sa micro-structure sur la partie droite. Les variables CSP v_i, $i \in$ {1,2,3} représentent les pigeons et les valeurs a et b expriment les pigeonniers.

La micro-structure de $Pigeons(3)$ admet une 2-coloration. En effet, il est facile de voir que les sommets $\langle v_1, a\rangle$, $\langle v_2, a\rangle$ et $\langle v_3, a\rangle$ peuvent être colorés avec une couleur, disons couleur 1 par exemple et les autres sommets $\langle v_1, b\rangle$, $\langle v_2, b\rangle$ et $\langle v_3, b\rangle$ sont colorés avec la couleur 2. Maintenant, par application du résultat du corollaire 1, nous pouvons déduire que le problème est inconsistant, puisqu'il n'y a pas de clique de taille 3 (une 3-clique).

Pour un CSP $P(n)$ ayant n variables, les auteurs de [6] tentent de trouver une $(n-1)$-coloration de sa micro-structure pour montrer son inconsistance, nous appelons cette méthode la méthode de coloration.

3.1 Les bases de la méthode Dominance-Coloration

Maintenant, nous définissons la notion de dominance entre CSPs qui est l'une des deux fondamentales notions sur lesquelles la méthode est basée.

Définition 1 *Si $P(n)$ et $P'(n)$ sont deux CSPs ayant n variables, alors $P'(n)$ domine $P(n)$, si et seulement si, il existe une application qui couple toute solution de $P(n)$ avec une solution de $P'(n)$.*

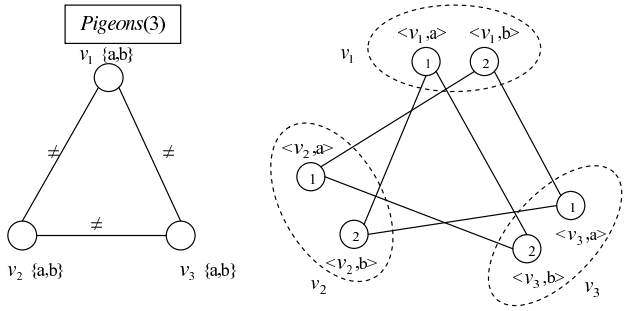

FIG. 1 – $Pigeons(3)$: une instance du problème des pigeons

A partir de la définition précédente on comprend que le CSP $P'(n)$ domine un autre CSP $P(n)$ si et seulement si chaque solution de $P(n)$ peut être transformée en une solution de $P'(n)$. L'interaction entre la dominance et la consistance de CSP est donnée dans la proposition suivante.

Proposition 1 *Si un CSP $P'(n)$ inconsistant domine le CSP $P(n)$, alors $P(n)$ est inconsistant.*

***Preuve* 1** *La preuve peut facilement être dérivée à partir de la définition 1.*

Nous définissons maintenant la notion de k-coloration qui est le second élément de base de notre méthode.

Définition 2 *Soient le CSP $P(n)$ et sa micro-structure $\mathcal{M}_{P(n)}$. Une k-coloration de $\mathcal{M}_{P(n)}$, est une application f_k qui associe à chaque sommet de $\mathcal{M}_{P(n)}$, une couleur représentée par un entier j tel que $1 \leq j \leq k$ et les couleurs des sommets adjacents doivent être différents.*

Maintenant, nous montrons comment associer à $P(n)$ un CSP donné ainsi qu'à f_k une k-coloration de sa micro-structure $\mathcal{M}_{P(n)}$, un CSP Alldiff($P(n)$,f_k) ayant une contrainte globale *Alldifferent* entre ses variables et nous prouvons que le CSP Alldifferent domine le CSP $P(n)$.

Définition 3 *Soient $P(n)$=(V, D, C) un CSP ayant n variables et f_k une k-coloration de sa micro-structure $\mathcal{M}_{P(n)}$. On associe à $P(n)$ et f_k, le CSP Alldiff($P(n), f_k$)=(V', D', C') où :*

- *$V' = \{v'_1, v'_2, ..., v'_n\}$, on associe une variable v'_i à chaque variable v_i de $P(n)$;*
- *$D' = \{D'_1, D'_2, ..., D'_n\}$, où chaque domaine D'_i est définit comme suit : $D'_i = \{f_k(\langle v_i, d_i\rangle)/d_i \in D_i\}$; en d'autres mots D'_i inclut toutes les couleurs de f_k utilisées pour la coloration des sommets $\langle v_i, d_i\rangle$, $\forall d_i \in D_i$;*
- *C'=$\{Alldifferent(v'_1, v'_2, ..., v'_n)\}$, donc, C' est formée par une unique contrainte globale Alldifferent $Alldifferent(v'_1, v'_2, ..., v'_n)$ définie sur les variables de V'.*

Exemple 1 *Reprenons le CSP précédent $Pigeons(3)$ exprimant le problème du pigeons de la figure 1 (partie gauche) et f_2 la 2-coloration de sa micro-structure (partie droite). On associe au CSP $Pigeons(3)$, le CSP Alldiff($Pigeons(3), f_2$)=(V', D', C') où :*

- *V'=$\{v'_1, v'_2, v'_3\}$, représente l'ensemble des variables associées aux variables $\{v_1, v_2, v_3\}$ du CSP initial $Pigeons(3)$;*
- *D'=$\{D'_1, D'_2, D'_3\}$, où D'_i=$\{f_2(\langle v_i, a\rangle), f_2(\langle v_i, b\rangle)\}$= $\{1, 2\}$, $\forall i \in \{1, 2, 3\}$;*
- *C'=$\{Alldifferent(v'_1, v'_2, v'_3)\}$.*

L'hypergraphe de contraintes du CSP Alldiff($Pigeons(3), f_2$), est représenté sur la figure 2.

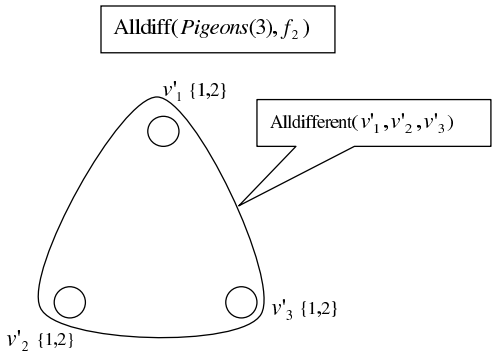

FIG. 2 – Le CSP Alldiff($Pigeons(3), f_2$) associé au CSP $Pigeons(3)$ et la 2-coloration f_2

Maintenant, nous introduisons la propriété clé de cette contribution.

Le résultat principal est dérivé de la relation de dominance existant entre le CSP $P(n)$ et le CSP Alldiff($P(n), f_k$) associé à $P(n)$ et à la k-coloration f_k. Nous avons le théorème suivant.

Théorème 1 *Si $P(n)$ est un CSP ayant n variables et f_k une k-coloration de sa micro-structure $\mathcal{M}_{P(n)}$, alors le CSP associé Alldiff($P(n), f_k$) domine le CSP $P(n)$.*

***Preuve* 2** *Nous allons prouver que, chaque solution I du CSP $P(n)$ peut être transformée en une solution I' du CSP Alldiff($P(n), f_k$). Soit I=$(\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_n, d_n\rangle)$ une instanciation de $P(n)$ et considérons l'application Γ : $\prod_{i=1}^{n}(\{v_i\} \times D_i) \longrightarrow \prod_{i=1}^{n}(\{v'_i\} \times D'_i)$, telle que $\Gamma(I)$=$(\langle v'_1, f_k(\langle v_1, d_1\rangle)\rangle, \langle v'_2, f_k(\langle v_2, d_2\rangle)\rangle, ...,$ $\langle v'_n, f_k(\langle v_n, d_n\rangle)\rangle)$. Maintenant, supposons que I est une solution de $P(n)$, alors nous déduisons que les sommets de l'ensemble $S = \{\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_n, d_n\rangle\}$ forment une clique de taille n dans la micro-structure $\mathcal{M}_{P(n)}$. Comme, f_k est une k-coloration de $\mathcal{M}_{P(n)}$, alors les sommets de la n-clique sont colorés avec des couleurs*

différentes deux à deux. En effet, $\forall \langle v_i, d_i\rangle$, $\langle v_j, d_j\rangle \in S$ *tels que* $i \neq j$, *nous avons* $f_k(\langle v_i, d_i\rangle) \neq f_k(\langle v_j, d_j\rangle)$. *Finalement,* $\Gamma(I)=(\langle v'_1, f_k(\langle v_1, d_1\rangle)\rangle, \langle v'_2, f_k(\langle v_2, d_2\rangle)\rangle, ..., \langle v'_n, f_k(\langle v_n, d_n\rangle)\rangle)=I'$ *est une solution du CSP Alldiff(*$P(n), f_k$*).*

Le théorème 1 met en évidence un résultat important. Nous avons montré que le CSP Alldiff($P(n), f_k$) associé à $P(n)$ et f_k domine le CSP $P(n)$. Par ailleurs, le CSP Alldiff($P(n), f_k$) a une unique contrainte globale *Alldifferent* qui est bien connue et pour laquelle plusieurs algorithmes dédiés existent. Un des algorithmes les plus connus pour cette contrainte est sans aucun doute, l'algorithme de Régin [12] qui est basé sur la recherche des couplages de cardinalité maximale dans des graphes biparties. Cet algorithme permet d'effectuer l'arc consistance généralisée (GAC) sur des contraintes globales du type Alldifferent avec une complexité en temps polynomiale.

L'algorithme GAC est complet pour le test d'inconsistance des CSPs Alldifferent et il a de bonnes performances en pratique. Nous l'utilisons pour le test de consistance du CSP Alldiff($P(n), f_k$) dérivé de $P(n)$ et f_k. Si le CSP Alldiff($P(n), f_k$) est montré inconsistant, alors nous déduisons de la proposition 1 que le CSP $P(n)$ est inconsistant.

Une condition nécessaire pour que le CSP Alldiff($P(n), f_k$) soit consistant est que le nombre de valeurs différentes dans $\cup_{i=1,n} D'_i$ doit être au moins égal à n (i.e. $\mid \cup_{i=1,n} D'_i \mid \geq n$). Sinon, il y aurait moins de valeurs que de variables et dans ce cas la contrainte globale Alldifferent devient inconsistante.

Remarque 1 *Puisque les valeurs des domaines associés aux variables du CSP Alldiff(*$P(n), f_k$*) sont les couleurs utilisées par* f_k *pour colorer la micro-structure* $\mathcal{M}_{P(n)}$*, alors l'inconsistance du CSP détéctée par la méthode donnée dans [6] est un cas particulier de l'inconsistance du CSP Alldiff(*$P(n), f_k$*) correspondant au cas trivial où le CSP Alldiff(*$P(n), f_k$*) contient moins de valeurs que de variables qui est inclus dans notre méthode. D'où le fait que notre méthode inclus la méthode décrite dans [6] et détectera donc, au minimum, les inconsistances détectables par cette dernière.*

Grâce au théorème 1 nous savons que si le CSP Alldiff($P(n), f_k$) est inconsistant, alors $P(n)$ l'est aussi. Cependant, si après application de l'algorithme GAC sur le CSP Alldiff($P(n), f_k$), le CSP reste consistant, alors nous ne pouvons déduire aucune information sur la consistance de $P(n)$, mais, certaines valeurs qui ne participent à aucune solutions dans Alldiff($P(n), f_k$) peuvent être filtrées par GAC durant le test de consistance. Nous montrons dans ce qui suit comment nous pouvons exploiter ces valeurs pour simplifier le CSP $P(n)$ en supprimant un certain nombre de valeurs ne participant à aucune solution dans $P(n)$. Nous avons la proposition suivante.

Proposition 2 *Soient* $P(n)=(V, D, C)$ *un CSP,* f_k *une* k*-coloration de sa micro-structure, Alldiff(*$P(n), f_k$*)=(*V', D', C'*) le CSP Alldiff associé à* $P(n)$ *et* f_k*, et* $v'_i \in V'$ *la variable associée à* $v_i \in V$*. S'il existe une valeur* $d'_i \in D'_i$ *de la variable* v'_i*, telle que* d'_i *ne participe à aucune solution dans Alldiff(*$P(n), f_k$*), alors chaque valeur* $d_i \in D_i$ *de la variable* v_i *qui vérifie* $f_k(\langle v_i, d_i\rangle) = d'_i$ *ne participe à aucune solution dans* $P(n)$.

***Preuve* 3** *Par hypothèses, le CSP Alldiff(*$P(n), f_k$*) domine le CSP* $P(n)$*. Donc, chaque solution* $I=(\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_n, d_n\rangle)$ *peut-être transformée par l'application* $\Gamma : \prod_{i=1}^{n}(\{v_i\} \times D_i) \longrightarrow \prod_{i=1}^{n}(\{v'_i\} \times D'_i)$*, en une solution* $\Gamma(I)=(\langle v'_1, f_k(\langle v_1, d_1\rangle)\rangle, \langle v'_2, f_k(\langle v_2, d_2\rangle)\rangle, ..., \langle v'_n, f_k(\langle v_n, d_n\rangle)\rangle)$ *du CSP Alldiff(*$P(n), f_k$*). Donc, si* d_i *participe à une solution dans* $P(n)$*, alors* $f_k(\langle v_i, d_i\rangle) = d'_i$ *participe à une solution dans Alldiff(*$P(n), f_k$*). D'où en passant à la contraposé, si la valeur* $f_k(\langle v_i, d_i\rangle) = d'_i$ *ne participe à aucune solution de Alldiff(*$P(n), f_k$*), alors la valeur* d_i *de* v_i *ne participe à aucune solution dans* $P(n)$.

Exemple 2 *Considérons le CSP* $P(3)$ *de la figure 3. Sa micro-structure admet une 3-coloration* f_3*. La méthode de coloration de [6] n'est pas pertinente ici, et ne peut prouver l'inconsistance de* $P(3)$*. Par ailleurs, le CSP Alldiff(*$P(3), f_3$*) associé à* $P(3)$ *et à la 3-coloration, obtenu à l'étape (E1) est consistant. On ne peut rien déduire sur la consistance de* $P(3)$*. Cependant, la valeur* 1 *du domaine de la variable* v'_3 *ne participe à aucune solution dans Alldiff(*$P(3), f_3$*), donc elle est supprimée par le filtrage GAC. Maintenant par application de la proposition 2, nous déduisons que la valeur* a *de la variable* v_3 *ne participe à aucune solution dans* $P(3)$*, elle est alors supprimée. Ceci mène à une simplification du CSP* $P(3)$ *(étape E2). En considérant maintenant le CSP simplifié, calculons une nouvelle coloration de la micro-structure simplifiée (étape E3). Nous obtenons une 2-coloration, ce qui signifie que le CSP est inconsistant.*

3.2 L'algorithme de Dominance-Coloration

L'exemple précédent donne l'intuition d'une méthode incrémentale pour le test d'inconsistance par l'exploitation du résultat de la proposition 2 et du théorème 1. En effet, en itérant plusieurs fois simplification et coloration nous pouvons tester l'inconsistance d'un CSP. Les fondements de cette méthode sont données dans l'algorithme 1.

Le principe de cet algorithme consiste à répéter les opérations suivantes jusqu'à montrer l'inconsistance ou que le nombre maximal d'étapes soit atteint.

1. Utiliser un algorithme incomplet de coloration de graphes (ici nous utilisons DSATUR [4]) pour calculer une k-coloration f_k de la micro-structure

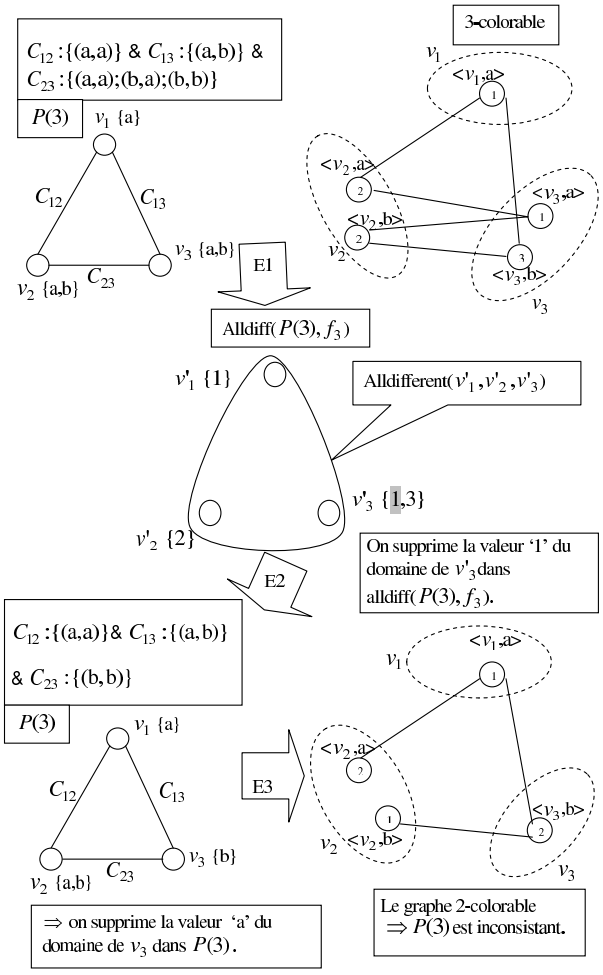

FIG. 3 – Un exemple de détection d'une inconsistance après filtrage de valeur

Algorithme 1 L'algorithme de dominance-coloration

Entrée : $P(n)$ {*n est le nombre de variables*}
Sortie : {"$P(n)$ inconsistant" ;"$P(n)$ simplifié" ;"rien"}

1: $test \leftarrow 1$
2: $Old\_P(n) \leftarrow P(n)$
3: $G \leftarrow$ micro-structure($P(n)$) {*Générer la micro-structure*}
4: **tant que** ($test \leq NBTESTS$) {*NBTESTS est le nombre maximal d'étapes*} **faire**
5: $f_k \leftarrow colorer(G)$
6: **si** ($k < n$) **alors**
7: **retourner** "$P(n)$ inconsistant"
8: **sinon**
9: $P'(n) \leftarrow construire\_Alldiff(P(n), f_k)$
10: $P''(n) \leftarrow GAC(P'(n))$
11: **si** ($est\_inconsistant(P''(n))$) **alors**
12: **retourner** "$P(n)$ inconsistant"
13: **sinon si** ($P''(n) \neq P'(n)$) **alors**
14: Mise_A_Jour($P(n), P''(n)$) {*simplification de $P''(n) \Rightarrow$ simplification de $P(n)$*}
15: $G \leftarrow$ micro-structure($P(n)$) {*Génération d'une nouvelle micro-structure*}
16: $test \leftarrow 0$ {*simplification de $P(n)$, faire NBTESTS nouveaux tests*}
17: **fin si**
18: **fin si**
19: $test \leftarrow test + 1$
20: **fin tant que**
21: **si** ($Old\_P(n) \neq P(n)$) **alors**
22: **retourner** "$P(n)$ simplifié"
23: **sinon**
24: **retourner** "rien"
25: **fin si**

G=$\mathcal{M}_{P(n)}$ (ligne 5). Le test d' inconsistance décrit dans [6] est effectué à la ligne 6 et 7. En effet, si une k-coloration telle que $k < n$ est trouvée, alors le CSP est inconsistant.

2. Autrement, dériver le CSP $P'(n)$=Alldiff($P(n), f_k$) à partir de $P(n)$ et f_k (ligne 9).
3. Appliquer l'algorithme d'arc consistance généralisé (GAC) sur $P'(n)$=Alldiff($P(n), f_k$) (ligne 10). On aura en retour, soit une preuve d'inconsistance du CSP $P'(n)$=Alldiff($P(n), f_k$) (ligne 11), d'où $P(n)$ est inconsistant, soit une forme simplifiée $P''(n)$ du CSP $P'(n)$, soit rien ne se passe.
4. Dans le cas d'une simplification, simplifier $P(n)$ en tenant compte de $P''(n)$ et calculer la micro-structure du CSP résultant (ligne 15), puis calculer une k-coloration pour la nouvelle micro-structure, et répéter les opérations décrites sur le CSP simplifié $P(n)$ (lignes 5-19).

Pour que l'algorithme termine, les étapes précédentes sont répétées NBTESTS fois. A la fin de l'exécution de l'algorithme, trois cas sont possibles :

– L'inconsistance de $P(n)$ est prouvée ;
– L'inconsistance n'est pas prouvée mais le CSP $P(n)$ est simplifié ;
– Ni l'inconsistance n'est montrée, ni le CSP $P(n)$ n'a subi de simplifications.

Si la méthode n'arrive pas à montrer l'inconsistance du CSP, mais retourne une forme simplifié de $P(n)$, alors on peut envisager d'utiliser un algorithme complet du type Forward Checking ou MAC par exemple, pour résoudre le CSP simplifié, ce qui pourrait être plus efficace que la résolution directe du CSP initial.

Si nous sommes dans le troisième cas (rien ne se passe), alors c'est que peut être, la coloration utilisée n'était pas pertinente. Il est donc important de trouver de bonnes heuristiques pour le choix de la k-coloration. Ce qui nous permettrait d'éviter de tomber dans ce cas défavorable.

4 Expérimentations

On se propose dans cette section d'évaluer expérimentalement les performances de notre méthode sur des instances aléatoires. Dans cette première implémentation, nous nous intéressons uniquement aux taux de détection d'instances inconsistantes par cette nouvelle méthode que nous comparerons aux taux d'autres méthodes.

4.1 Génération des problèmes

Les instances aléatoires sont générées en tenant compte des paramètres suivant :

- n le nombre de variables ;
- d la taille des domaines ;
- $density$ la densité des contraintes ;
- $tightness$ la dureté des contraintes.

Le code du générateur utilisé peut être trouvé à l'adresse (*http ://www.lirmm.fr/∼bessiere/generator.html*). Pour chaque test correspondant à n, d, $density$ et $tightness$ fixés, un échantillon de 100 problèmes est généré aléatoirement et les mesures sont prises en moyenne.

4.2 Les méthodes testées

Nous avons testé et comparé six méthodes :

1. La méthode arc consistance (AC) ;
2. La méthode chemin consistance ($Path$) ;
3. La méthode de la coloration donnée dans [6] ($Color$) ;
4. Une méthode hybride qui utilise les méthodes AC et $Color$ où AC est utilisée comme prétraitement pour $Color$ (AC_Color) ;
5. Une méthode hybride combinant les méthodes $Path$ et $Color$ où $Path$ est utilisée comme prétraitement pour $Color$ ($Path\_Color$) ;
6. La méthode Dominance-Coloration ($Dominance\_Color$).

En plus de ces méthodes, nous avons utilisé une méthode complète du type Forward Checking (FC) comme méthode référence pour évaluer et comparer les performances des différentes méthodes. La méthode FC détecte toutes les instances inconsistance, elle est utilisée pour calculer le taux de détection des instances inconsistantes des autres méthodes.

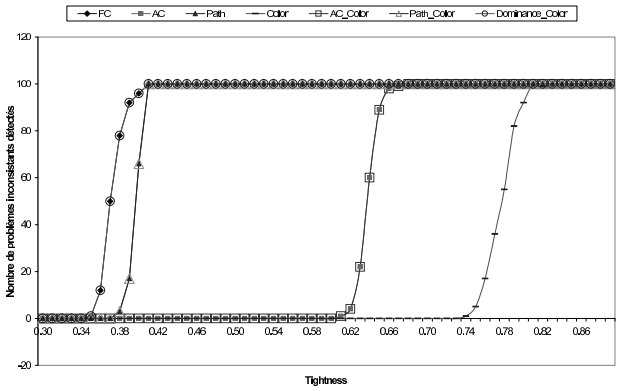

FIG. 4 – Taux d'inconsistance en fonction de $tightness$ pour ($n = 20$, $d = 10$ et $density = 0.5$)

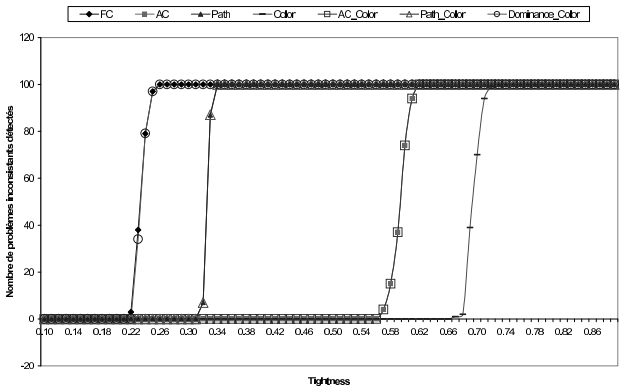

FIG. 5 – Taux d'inconsistance en fonction de $tightness$ pour $n = 20$, $d = 10$ et $density = 0.9$

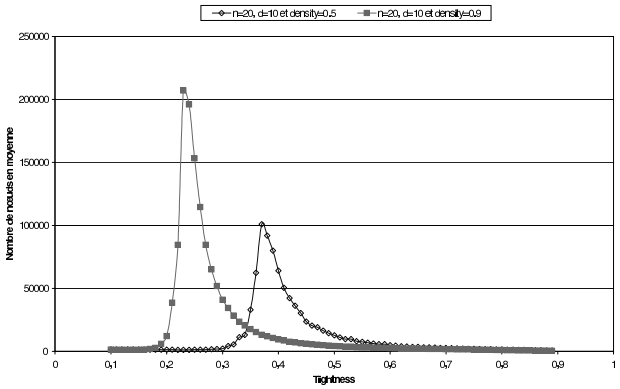

FIG. 6 – Le nombre de noeuds en moyenne en fonction de $tightness$ pour $n = 20$ et $d = 10$

4.3 Résultats

Nous avons reporté ici les courbes exprimant le taux de détection d'inconsistances en fonction de la variation de la dureté en fixant les autres paramètres n, d et $density$. La valeur limite de la constante NBTESTS de l'algorithme 1 est fixée à 5. Nous avons aussi fixé le temps d'exécution pour la résolution des 100 instances à une heure pour toutes les méthodes.

Les figures 4 et 5 montrent les résultats des toutes les méthodes sur les CSPs aléatoires où le nombre de variables est fixé à $n = 20$, la taille des domaines à $d = 10$ et où nous avons testé deux densités $density = 0.5$ et $density = 0.9$. Nous avons utilisé une version incomplète de la méthode DSATUR [4] pour le calcul d'une k-coloration de la microstructure.

Nous remarquons que la méthode $Dominance\_Color$ a le meilleur taux de détection, elle est plus efficace que toutes les autres méthodes. Sa courbe est confondue avec la courbe de la méthode complète FC pour la densité 0.5 et se détache très légèrement de celle-ci pour la densité 0.9, ce qui signifie qu'elle détecte pratiquement toutes les inconsistances.

Comme le filtrage de la chemin consistance est plus robuste que le filtrage arc consistant, les méthodes $Path$ et $Path\_Color$ ont un meilleur taux de détection que AC et AC_Color. La méthode $Color$ a le plus bas taux de détection d'inconsistances, elle a détecté uniquement les inconsistances des instances sur-contraintes. On note que les courbes des méthodes AC et AC_Color sont confondues, la même chose se passe avec les courbe des méthodes $Path$ et $Path\_Color$. Ceci explique le fait que l'opération de coloration n'améliore ni AC ni $Path$.

Nous pouvons voir que notre méthode est meilleure que les deux méthodes $Path$ et $Path\_Color$ au voisinage où les instances générées sont les plus difficiles (au voisinage du pic de difficulté). Nous pouvons voir sur les courbes de la figure 6 représentant le nombre moyen de noeuds générés en fonction de la dureté que le pic de difficultés est dans la région où la méthode $Dominance\_Coloration$ est meilleure que les deux méthodes précédentes. Le pic est situé approximativement au voisinage de $tightness = 0.37$ pour les instances ayant une densité $density = 0.5$ et au voisinage de $tightness = 0.23$ pour les instances de densité $density = 0.9$.

Notre méthode semble réussir où les autres méthodes échouent. Elle détecte la plus part des instances inconsistantes même au voisinage du pic de difficultés. Elle offre le meilleur taux de détection pour les instances difficiles. Nous espérons l'utiliser dans le futur pour détecter des cas d'inconsistance de CSP que les meilleures méthodes complètes n'arrivent pas à déterminer.

5 Conclusion et perspectives

Ce travail est une contribution à un challenge difficile qui consiste à trouver des méthodes incomplètes efficaces pour le test d'inconsistance de CSP. Nous avons étudié dans ce travail la notion de dominance et montré comment la combiner avec la coloration de graphes pour obtenir une méthode incomplète pour le test d'inconsistance. Nous avons introduit la méthode $Dominance\_Coloration$ qui inclus la méthode de coloration $Color$ donnée dans [6]. Notre méthode est basée sur la propriété de dominance entre le CSP dont on veux montrer l'inconsistance et un CSP Alldiff que nous construisons à partir du CSP original et une k-coloration de sa micro-structure. Nous avons montré que le CSP Alldiff domine le CSP original et certaines valeurs filtrées dans le CSP Alldiff par l'algorithme GAC induisent des filtrages dans le CSP original sans effort additionnel. Notre méthode peut être alors vue comme une nouvelle technique de filtrage CSP. Son efficacité vient du fait que l'algorithme GAC est polynomial dans le cas des CSPs Alldiff.

Les expérimentations faites sur des instances CSP aléatoires, montrent que notre méthode à un bon taux de détection d'inconsistances même au voisinage du pic de difficultés. Les résultats obtenus sur les problèmes testés ont montré que la nouvelle méthode est meilleure (en taux de détection) que toutes les autres méthodes : l'arc consistance (AC), la chemin consistance ($Path$), la méthode de la coloration ($Color$) et les méthodes hybrides AC_Color et $Path\_Color$.

Ce travail constitue une étude préliminaire d'une nouvelle approche pour la détection des problèmes CSPs inconsistants qui offre une réponse au challenge qui consiste à prouver l'inconsistance par exploitation de méthodes incomplètes. Cette contribution laisse envisager plusieurs autres pistes que nous sommes en train d'explorer, parmi elles :

- Nous sommes à la recherche d'heuristiques pour la coloration de la micro-structure plus efficaces pour le filtrage à base de dominance. Ceci permettrait d'améliorer sensiblement le taux de détection d'instances inconsistantes de notre méthode ;
- Comme la méthode peut induire un certain nombre de filtrage de valeurs dans le CSP initial, il serait intéressant de la combiner avec une méthode de résolution et de l'exploiter comme technique de filtrage qu'on utiliserait avant ou durant la recherche de solutions. Ceci est un point particulièrement intéressant qui est actuellement en cours d'étude.
- Finalement, il serait important de trouver d'autres CSPs pertinents que l'on pourrait dériver du CSP original $P(n)$ qui conserverait la propriété de dominance.

Références

[1] Fadi Aloul, Inês Lynce, and Steven Prestwich. Symmetry breaking in local search for unsatisfiability. In *7th International Workshop on Symmetry and Constraint Satisfaction Problems*, pages 9–13, Providence, RI, September 2007.

[2] Gilles Audemard and Laurent Simon. Gunsat : A greedy local search algorithm for unsatisfiability. In *IJCAI*, pages 2256–2261, 2007.

[3] J.N. Bès and P. Jégou. Proving graph un-colorability with a consistency check of csp. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 693–694, Hong Kong, China, novembre 2005. IEEE. POSTER.

[4] Daniel Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4) :251–256, 1979.

[5] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7) :394–397, 1962.

[6] Daya Ram Gaur, W. Ken Jackson, and William S. Havens. Detecting unsatisfiable csps by coloring the micro-structure. In *AAAI/IAAI*, pages 215–220, 1997.

[7] Eugene Goldberg. Proving unsatisfiability of cnfs locally. *J. Autom. Reasoning*, 28(5) :417–434, 2002.

[8] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search : Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[9] P. Jégou. Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In *In Proceedings AAAI'93*, 1993.

[10] Etienne De Klerk, Hans Van Maaren, and Joost P. Warners. Relaxations of the satisfiability problem using semidefinite programming. *J. Autom. Reason.*, 24(1-2) :37–65, 2000.

[11] Steve Prestwich and Inês Lynce. Local search for unsatisfiability. In *9th International Conference on Theory and Applications of Satisfiability Testing*, Lecture Notes in Computer Science, pages 283–296. Springer, August 2006.

[12] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94 : Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[13] Bart Selman, Henry A. Kautz, and David A. McAllester. Ten challenges in propositional reasoning and search. In *IJCAI*, pages 50–54, 1997.

[14] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.