Extraction de noyaux insatisfiables minimaux de réseaux de contraintes

Fred Hemery, Christophe Lecoutre, Lakhdar Sais, Frédéric Boussemart

Extraction de noyaux insatisfiables minimaux de réseaux de contraintes

Fred Hemery **Christophe Lecoutre** **Lakhdar Sais** **Frédéric Boussemart**

CRIL - CNRS FRE 2499
Université d'artois
rue de l'université, SP 16
62307 Lens cedex, France

{hemery,lecoutre,sais,boussemart}@cril.univ-artois.fr

Résumé

Nous nous intéressons au problème de l'extraction de noyaux insatisfiables minimaux (MUCs) de réseaux de contraintes. Ce problème a un intérêt pratique dans de nombreux domaines d'application tels que la configuration, la planification, le diagnostique, etc. En effet, identifier un ou plusieurs MUCs indépendants, i.e. des MUCs qui ne partagent aucune contrainte, permet d'isoler différentes sources d'inconsistance et de corriger un système incohérent. Dans cet article, nous proposons une approche originale pour extraire un MUC d'un réseau de contraintes. Cette approche comporte deux étapes. La première consiste à exécuter plusieurs fois un algorithme de recherche complète, en utilisant la pondération de contraintes, de manière à circonscrire une partie inconsistante du réseau. La seconde consiste à identifier, en utilisant un processus dichotomique, les contraintes de *transition* appartenant à un MUC. Nous montrons l'intérêt de cette approche en avançant des arguments théoriques et pratiques.

Abstract

We address the problem of extracting Minimal Unsatisfiable Cores (MUCs) from constraint networks. This computationally hard problem has a practical interest in many application domains such as configuration, planning, diagnosis, etc. Indeed, identifying one or several disjoint MUCs can help circumscribe different sources of inconsistency in order to repair a system. In this paper, we propose an original approach that involves performing successive runs of a complete backtracking search, using constraint weighting, in order to surround an inconsistent part of a network, before identifying all *transition* constraints belonging to a MUC using a dichotomic process. We show the effectiveness of this approach, both theoretically and experimentally.

1 Introduction

Un réseau de contraintes est dit insatisfiable minimal si et seulement si il est insatisfiable et si lui ôter une contrainte choisie aléatoirement le rend satisfiable. Déterminer si un ensemble de contraintes est insatisfiable minimale est reconnu DP-Complet [23]. Il peut être réduit à un problème SAT-UNSAT : étant données deux formules ϕ et ψ, est-ce que ϕ est satisfiable et ψ insatisfiable ? DP correspond au second niveau de la hiérarchie booléenne. Un problème dans cette classe peut être considéré comme la différence entre deux problèmes NP [22].

D'un point de vu pratique, quand une inconsistance est constatée dans un système, identifier les éléments qui sont en conflit peut aider à comprendre, expliquer, diagnostiquer les causes et permet de proposer un système rendu consistant. L'intérêt du sujet abordé dans ce papier peut être illustré avec le problème d'allocation de fréquences de liaisons radios (RLFAP) souvent utilisé comme banc d'essai dans la communauté CSP (Constraint Satisfaction Problem). Ce problème consiste à assigner des fréquences à un ensemble de liaisons radios définies entre des couples de transmetteurs de manière à éviter les interférences. Identifier les sous-réseaux (minimaux) inconsistants favorise la recherche de nouvelles positions pour les transmetteurs concernés par la zone.

Dans le cas des contraintes booléennes (formules CNF, c'est à dire formules booléennes sous forme normale conjonctive), trouver les formules insatisfiables minimales

est un thème de recherche très actif. Des classes de formules solubles ont été identifiées. La plupart d'entre elles sont basées sur les "deficiencies" des formules (i.e. la différence entre le nombre de clauses et le nombre de variables) [5, 8]. De plus, des avancés récentes pour le problème de satisfiabilité d'une formule booléenne (problème SAT) ont permis de nouvelles extensions des solveurs et d'aborder de manière fructueuse le problème particulièrement difficile d'extraction de noyaux insatisfiables minimaux [4, 29, 17, 21].

Dans le contexte de la satisfaction de contraintes, de nombreux travaux traitent de l'identification d'ensembles de contraintes en conflit. Ces ensembles, qui peuvent être obtenus en mémorisant des explications pendant la recherche, sont généralement utilisés pour mettre en oeuvre des techniques de retours-arrières intelligents comme *conflict-based backjumping* [25] ou *dynamic backtracking* [9, 15]. Ils peuvent être également calculés [12] de manière non-intrusive (c'est à dire, à la demande). Cependant, peu de travaux concernent de façon spécifique l'extraction de MUCs (Minimal Unsatisfiable Cores) de réseaux de contraintes. Une approche pour diagnostiquer les réseaux sur-contraints est proposée dans [1] et une méthode pour extraire tous les MUCs d'un ensemble de contraintes donné est présentée dans [10, 6]. Cette méthode effectue une parcours exhaustif d'un arbre (appelé CS), mais est limitée par l'explosion combinatoire du nombre de sous-ensembles de contraintes. Enfin, une approche "diviser pour régner" est proposée dans [13] afin d'extraire une explication (ou une relaxation) d'un problème sur-contraint sur la base de préférences donnée par l'utilisateur.

Dans ce papier, nous proposons une approche originale pour extraire un MUC d'un réseau de contraintes. Cette approche comporte deux étapes. La première étape exploite l'heuristique de choix de variable dirigée par les conflits $dom/wdeg$ [2] pour identifier (et extraire) un noyau inconsistant sur la base de plusieurs exécutions complètes d'un algorithme de recherche en profondeur d'abord avec retours-arrières. La recherche est relancée, en préservant le poids des contraintes (géré par $dom/wedg$) d'une exécution à l'autre, jusqu'à ce que la taille du noyau démontré insatisfiable ne puisse plus être réduite. A partir d'un ordre total des contraintes basé sur leur poids, et en utilisant le principe introduit dans [7, 12], la seconde étape permet d'identifier de façon itérative les contraintes d'un MUC. En comparaison avec les approches *constructive* [7] et *destructive* [1] qui ont une complexité respective dans le pire des cas en O($e.k_e$) et $\theta(e)$, l'approche *dichotomique* que nous proposons a une complexité en O($log(e).k_e$). Ici, la complexité correspond au nombre d'appels de l'algorithme de recherche complète, e représente le nombre de contraintes du réseau de contraintes et k_e le nombre de contraintes du noyau extrait. Nous comparons aussi plus finement cette complexité avec celle obtenue par U. Junker [13].

Le papier est organisé de la manière suivante. Dans un premier temps nous introduisons quelques définitions préliminaires. Puis, nous présentons les deux étapes de notre approche : extraction d'un noyau insatisfiable à l'aide de la pondération des contraintes et extraction d'un noyau minimal par identification des contraintes dite de *transition*. Ensuite une discussion sur les travaux similaires est proposée. Enfin, nous présentons quelques résultats expérimentaux avant de conclure.

2 Définitions préliminaires

Définition 1. *Un réseau de contraintes (CN) P est un couple $(\mathscr{X}, \mathscr{C})$ où :*

- *$\mathscr{X}$ est un ensemble fini de n variables tel que chaque variable $X \in \mathscr{X}$ possède un domaine dom(X) représentant l'ensemble des valeurs pouvant être affectées à X,*
- *$\mathscr{C}$ est un ensemble fini de e contraintes tel que chaque contrainte $C \in \mathscr{C}$ implique un sous-ensemble de variables de $\mathscr{X}$, noté vars(C), et à une relation associée rel(C), qui représente l'ensemble des tuples autorisés pour les variables de vars(C).*

Pour tout sous-ensemble de contraintes $S \subseteq \mathscr{C}$ de P, $P^{\uparrow S}$ représentera le réseau de contraintes obtenu à partir de P en supprimant toutes les contraintes de S et $P_{\downarrow S}$ sera équivalent à $P^{\uparrow(\mathscr{C}-S)}$. Une solution est une assignation de valeurs à l'ensemble des variables telle que toutes les contraintes soient satisfaites. Un réseau est satisfiable lorsqu'il admet au moins une solution. Le problème de satisfaction de contraintes (CSP pour Constraint Satisfaction Problem), qui consiste à déterminer si un réseau de contraintes donné est satisfiable, est NP-complet. Un réseau de contraintes est également appelé instance CSP. Dans cet article, résoudre une instance revient soit à trouver une solution, soit à démontrer qu'elle est insatisfiable.

Pour résoudre une instance, on peut utiliser un algorithme de recherche en profondeur d'abord avec retours-arrières, dans lequel à chaque étape de la recherche, une assignation de variable est effectuée suivie d'un processus de filtrage appelé propagation de contraintes. Généralement, les algorithmes de propagation de contraintes sont basés sur des propriétés du réseau de contraintes telles que la consistance d'arc et éliminent des valeurs inconsistantes (c'est à dire des valeurs qui ne peuvent apparaître dans aucune solution). L'algorithme qui maintient la consistance d'arc pendant la recherche est appelé MAC [26].

Un noyau insatisfiable est un sous-réseau insatisfiable d'un réseau de contraintes.

Définition 2. *Soient $P = (\mathscr{X}, \mathscr{C})$ et $P' = (\mathscr{X}', \mathscr{C}')$ deux CNs. P' est un noyau insatisfiable de P ssi P' est insatisfiable et $\mathscr{X}' \subseteq \mathscr{X} \wedge \mathscr{C}' \subseteq \mathscr{C}$.*

De nombreux noyaux insatisfiables peuvent exister, pour un réseau de contraintes donné. Parmi eux, ceux qui ne contiennent pas de noyau insatisfiable sont dit minimaux.

Définition 3. *Soient $P = (\mathscr{X}, \mathscr{C})$ un CN et $P' = (\mathscr{X}', \mathscr{C}')$ un noyau insatisfiable de P. P' est un noyau minimal insatisfiable (MUC) de P ssi il n'existe pas de noyau insatisfiable P'' de P' tel que $P'' \neq P'$.*

Pour montrer la minimalité d'un noyau insatisfiable, il suffit de vérifier la satisfiabilité de tous les réseaux de contraintes obtenus en éliminant une contrainte.

3 Extraction de noyaux insatisfiables

En reprenant l'idée énoncée dans [1], nous introduisons une approche qui permet de supprimer des contraintes tout en conservant l'insatisfiabilité. Ensuite nous affinons cette approche en exploitant la pondération des contraintes et la technique de redémarrage (complet) de la recherche.

3.1 Une approche basée sur une preuve

Quand l'insatisfiabilité d'une instance CSP a été prouvée par l'utilisation d'un algorithme de recherche embarquant un processus de filtrage, on peut garantir l'extraction d'un noyau insatisfiable. En effet, il suffit d'identifier toutes les contraintes qui ont été utiles à la preuve de l'insatisfiabilité, c'est à dire, toutes les contraintes qui ont été utilisées pendant la recherche pour supprimer, par propagation, au moins une valeur du domaine d'une variable. Ce principe a été mentionné dans [1] et il est comparable au concept de graphe d'implication utilisé en SAT (par exemple, voir [19, 29]).

Examinons comment fonctionne cette approche avec MAC qui maintient la consistance d'arc en exploitant, par exemple, un algorithme comme AC3 [18]. Des révisions d'arc (couple composé d'une contrainte et d'une variable) successives sont effectuées afin de supprimer des valeurs qui ne sont pas consistantes par rapport à l'état courant. La fonction au coeur du solveur est décrite par l'algorithme 1. Toutes les valeurs d'une variable donnée qui n'ont pas de support dans une contrainte donnée sont supprimées (lignes 2 à 4).

La structure de donnée, appelée $active$, qui associe un booléen à chaque contrainte, nous permet d'extraire un noyau insatisfiable. La fonction $pcore$ décrite dans l'algorithme 2 réalise cette extraction. Initialement, tous les booléens sont initialisés à la valeur $faux$ (ligne 1). Des révisions successives sont alors effectuées par le solveur MAC qui est appelé (ligne 2), et lors d'une révision effective (i.e. une révision qui permet de supprimer au moins une valeur) d'un arc (C,X), la valeur du booléen associé à la contrainte C devient égale à $vrai$ (ligne 5 de l'algorithme 1). Pour terminer, la fonction renvoie (ligne 3) le CN obtenu à partir de P en supprimant toutes les contraintes C telles que $active[C]$ est $faux$. Il est important de noter que le réseau retourné par $pcore$ est forcément insatisfiable mais pas nécessairement minimal.

Algorithme 1 revise(C : Contrainte, X : Variable) : booléen

1: $tailleDomaine \leftarrow |dom(X)|$
2: **pour** chaque $a \in dom(X)$ **faire**
3: **si** $rechercheSupport(C, X, a) = faux$ **alors**
4: supprimer a de $dom(X)$
5: $active[C] \leftarrow vrai$
6: **si** $dom(X) = \emptyset$ **alors**
7: $wght[C] \leftarrow wght[C] + 1$ // utilisé par $dom/wdeg$
8: retourner $tailleDomaine \neq |dom(X)|$

Algorithme 2 pcore(P= $(\mathscr{V}, \mathscr{C})$: CN) : CN

1: $active[C] \leftarrow faux \; \forall C \in \mathscr{C}$
2: MAC(P)
3: retourner $P^{\uparrow\{C \in \mathscr{C} | active[C] = faux\}}$

3.2 Une approche basée sur les conflits

Bien que l'approche précédente soit élégante, nous n'avons aucune idée de son efficacité pratique. En d'autres termes, nous sommes incapable de prédire la taille du noyau insatisfiable extrait par $pcore$. Il est clair que plus la taille du noyau est petite et plus l'approche est efficace. En fait, comme illustré ci-dessous, l'exploitation de l'heuristique de choix de variable $dom/wdeg$ permet de faire remonter un noyau insatisfiable en exécutant plusieurs recherches successives.

L'heuristique dom/wdeg Dans [2], un compteur, noté $wght[C]$, est associé à chaque contrainte C du problème. Ces compteurs sont utilisés pour pondérer les contraintes. Lorsqu'une contrainte est rendue insatisfiable (au cours du processus de propagation de contraintes), son poids est incrémenté de 1 (voir la ligne 7 de l'algorithme 1).

Le degré pondéré d'une variable X correspond à la somme des poids des contraintes liant X et au moins une autre variable future. L'heuristique $dom/wdeg$ [2] sélectionne la variable ayant le plus petit ratio taille du domaine sur degré pondéré. Lors de la progression de la recherche, le poids des contraintes difficiles augmente et cela aide l'heuristique à sélectionner les variables apparaissant dans la partie difficile du réseau. Cette heuristique a montré une grande efficacité [2, 16, 11].

Illustration sur un exemple L'utilisation de l'heuristique $dom/wdeg$ permet de prouver efficacement l'insatis-

fiabilité de nombreuses instances. Cependant pour obtenir un noyau insatisfiable de taille modérée, il est important d'exécuter successivement plusieurs recherches. En guise d'illustration, intéressons nous au problème qui consiste à placer des reines et des cavaliers sur un échiquier comme cela est décrit dans [2]. L'instance de ce problème pour 6 reines et 3 cavaliers se compose de 9 variables et 36 contraintes, et elle est insatisfiable. En fait, nous savons que le sous-problème qui correspond aux 3 cavaliers impliquant 3 variables et 3 contraintes est insatisfiable. Dans une première phase, résoudre cette instance avec MAC-$dom/wdeg$ (c'est à dire MAC combiné avec l'heuristique de choix de variable $dom/wdeg$) renvoie une preuve d'insatisfiabilité contenant toutes les contraintes de l'instance (c'est à dire que tous les booléens de la structure de donnée $active$ ont été positionnés à $vrai$). Cependant résoudre une nouvelle fois la même instance en utilisant les poids des contraintes obtenus après la première exécution renvois une nouvelle preuve d'insatisfiabilité contenant uniquement 9 contraintes. L'exécution suivante fournit le même résultat.

La figure 1 illustre l'évolution des preuves d'insatisfiabilité. Les réseaux de contraintes sont représentés par des graphes où chaque sommet correspond à une variable et chaque arête à une contrainte binaire. Les contraintes insatisfiables après une exécution sont représentées par des arêtes de couleur grise.

Il est possible d'améliorer l'extraction après la suppression de toutes les contraintes qui ne sont pas liées au noyau détecté, c'est à dire à la preuve d'insatisfiabilité. En effet, dans la seconde phase, nous obtenons un noyau insatisfiable qui correspond au sous problème des cavaliers.

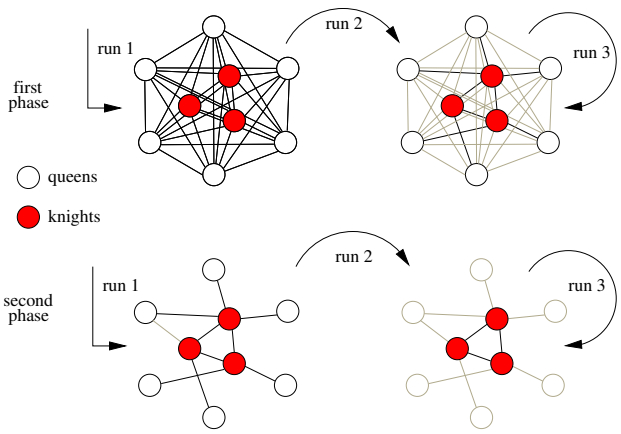

FIG. 1 – Évolution de la preuve d'insatisfiabilité

Exploitation de dom/wdeg Comme illustré précédemment, exécuter plusieurs fois le solveur MAC peut permettre de trouver un noyau insatisfiable de petite taille à la condition qu'une heuristique dirigée par les conflits comme $dom/wdeg$ soit utilisée. Cette approche est décrite par l'algorithme 3. Initialement (ligne 1), le poids de chaque contrainte vaut 1. Puis, MAC-$dom/wdeg$ est exécuté de façon itérative (ligne 6) et le nombre de contraintes trouvées dans le noyau insatisfiable est évalué à chaque exécution (ligne 7). L'itération s'arrête lorsque la taille courante du noyau insatisfiable est supérieure ou égale à la précédente. Le fait que, d'une exécution à l'autre, la valeur des compteurs $wght$ soit préservée, permet potentiellement de concentrer la recherche, vers un noyau insatisfiable de plus en plus petit. Par manque de place, l'algorithme décrit ne réalise pas l'itération de plusieurs phases comme indiqué dans l'illustration.

Algorithme 3 wcore(P= $(\mathscr{V},\mathscr{C})$: CN) : CN

1: $wght[C] \leftarrow 1\ \forall C \in \mathscr{C}$
2: $cnt_{aft} \leftarrow +\infty$
3: **répéter**
4: $\quad active[C] \leftarrow faux\ \forall C \in \mathscr{C}$
5: $\quad cnt_{bef} \leftarrow cnt_{aft}$
6: $\quad$ MAC-$dom/wdeg(P)$
7: $\quad cnt_{aft} \leftarrow \mid \{C \in \mathscr{C} | active[C]\} \mid$
8: **jusqu'à** $cnt_{aft} \geq cnt_{bef}$
9: retourner $P^{\uparrow\{C\in\mathscr{C}|active[C]=faux\}}$

4 Extraction de noyaux insatisfiables minimaux

Il est clair que les noyaux insatisfiables extraits par la fonction $wcore$ n'ont pas la garantie d'être minimaux. De façon à obtenir un noyau minimal, il est nécessaire d'identifier de manière itérative les contraintes qui le composent. Plus précisément, nous savons que, étant donnés un réseau de contraintes insatisfiable P et un ordre total sur les contraintes (pour simplifier, nous considérerons l'ordre lexicographique $C_1, C_2, \ldots, C_e$ des contraintes), il existe une contrainte C_i telle que $P_{\downarrow\{C_1,\ldots,C_{i-1}\}}$ est satisfiable et $P_{\downarrow\{C_1,\ldots,C_i\}}$ est insatisfiable[1]. Cette contrainte qui clairement appartient à un noyau minimal de P sera appelée la contrainte de *transition* de P (selon l'ordre utilisé). Notons aussi que toute contrainte C_j avec $j > i$ peut être supprimée en toute sécurité.

4.1 Identification de la contrainte de transition

Il est possible d'identifier la contrainte de *transition* d'un CN P insatisfiable en utilisant une approche constructive, une approche destructive ou une approche dichotomique. Pour la suite, on considère que MAC(P) retourne sat si P est satisfiable et $unsat$ sinon. On utilise également un paramètre k ($< |\mathscr{C}|$) qui indique le nombre de contraintes

[1] Nous supposerons qu'il n'existe pas de contrainte C dans P telle que rel(C) = $\emptyset$.

de transition identifiées précédemment (les k premières contraintes du réseau courant). Dans un premier temps, considérons que $k = 0$.

Approche constructive Le principe consiste à ajouter successivement les contraintes du réseau jusqu'à ce que le réseau courant devienne insatisfiable. Cette approche est analogue à celle introduite dans [7] et elle est décrite par l'algorithme 4.

Algorithme 4 csTransition(P= $(\mathscr{V}, \mathscr{C})$: CN, k : int) : Contrainte

1: **pour** i variant $\nearrow$ de $k+1$ à $|\mathscr{C}|$ **faire**
2: **si** MAC($P_{\downarrow\{C_1,\ldots,C_i\}}$) = $unsat$ **alors**
3: retourner C_i

Approche destructive Le principe consiste à supprimer les contraintes du réseau jusqu'à ce qu'il devienne satisfiable (notons qu'avec l'hypothèse $rel(C) \neq \emptyset\ \forall C \in \mathscr{C}$, i ne peut jamais être égal à 1). Cette approche, introduite dans [1], est décrite par l'algorithme 5.

Algorithme 5 dsTransition(P= $(\mathscr{V}, \mathscr{C})$: CN, k : int) : Contrainte

1: **pour** i variant $\searrow$ de $|\mathscr{C}|$ à $k+1$ **faire**
2: **si** MAC($P_{\downarrow\{C_1,\ldots,C_{i-1}\}}$) = sat **alors**
3: retourner C_i

Approche dichotomique Enfin, il est possible d'utiliser une recherche dichotomique pour trouver la contrainte de transition. Cette approche est décrite par l'algorithme 6. A chaque étape de la recherche, nous savons que la contrainte de transition de P appartient à $\{C_{min}, \ldots, C_{max}\}$.

Algorithme 6 dcTransition(P= $(\mathscr{V}, \mathscr{C})$: CN, k : int) : Contrainte

1: $min \leftarrow k+1$; $max \leftarrow |\mathscr{C}|$
2: **tant que** $min \neq max$ **faire**
3: $centre \leftarrow (min + max)/2$
4: **si** MAC($P_{\downarrow\{C_1,\ldots,C_{centre}\}}$) = sat **alors**
5: $min \leftarrow centre + 1$
6: **sinon**
7: $max \leftarrow centre$
8: retourner C_{min}

4.2 Extraction d'un noyau minimal

Les fonctions décrites plus haut permettent de trouver la contrainte de transition C_i d'un réseau insatisfiable P, c'est à dire un élément d'un noyau minimal de P. Pour récupérer un deuxième élément, nous appliquons le même principe sur le nouveau CN P' qui est obtenu à partir de P en supprimant toutes les contraintes C_j telles que $j > i$ (car l'insatisfiabilité est préservée) et en utilisant un nouveau ordre pour les contraintes tel que C_i est considérée comme le plus petit élément (l'ordre naturel peut être préservé simplement en renommant C_i par C_1 et C_j par C_{j+1} pour $1 \leq j < i$). Ce processus est répété jusqu'à ce que toutes les contraintes du réseau courant correspondent à des contraintes de transition successivement trouvées. Le principe de ce processus itératif a été décrit dans [7, 12, 24]. Il est représenté par l'algorithme 7 qui renvoie un MUC pour un réseau donné (les 3 approches sont obtenues en remplaçant xx par cs, ds ou dc). Notons qu'il faut déterminer si la dernière contrainte appartient au MUC (lignes 7 et 8).

Algorithme 7 xxMUC(P= $(\mathscr{V}, \mathscr{C})$: CN) : CN

1: $P' = (\mathscr{V}', \mathscr{C}') \leftarrow P$; $k \leftarrow 0$
2: **tant que** $k < |\mathscr{C}'| - 1$ **faire**
3: $C_i \leftarrow xxTransition(P', k)$
4: $k \leftarrow k + 1$
5: $P' \leftarrow P'^{\uparrow\{C_j | j>i\}}$
6: dans P', renommer C_i en C_1 et C_j en C_{j+1}, $1 \leq j < i$
7: **si** MAC($P'^{\uparrow\{C_{|\mathscr{C}'|}\}}$) = $unsat$ **alors**
8: retourner $P'^{\uparrow\{C_{|\mathscr{C}'|}\}}$
9: **sinon**
10: retourner P'

La proposition suivante suggère que, parmi les approches proposées, l'approche dichotomique est plus efficace que les deux autres.

Proposition 1. *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN insatisfiable, le nombre d'appels à MAC dans le pire des cas est :*
- *O($e.k_e$) pour $csMUC(P)$,*
- *$\theta(e)$ pour $dsMUC(P)$,*
- *O($log_2(e).k_e$) pour $dcMUC(P)$.*

Ici, $e = |\mathscr{C}|$ et k_e représente le nombre de contraintes du MUC extrait.

Preuve. Pour $csMUC$, dans le pire des cas, e appels à MAC sont requis pour trouver la première contrainte de transition, $e-1$ appels pour trouver la seconde contrainte de transition,... Comme k_e contraintes de transition doivent être trouvées, nous obtenons O($e + e - 1 + \cdots + e - (k_e - 1)$) ce qui correspond à O($e.k_e$). Pour $dsMUC$, il y a exactement un appel à MAC par contrainte. Le nombre d'appels est $\theta(e)$. Pour $dcMUC$, $log_2(e)$ appels à MAC sont requis pour trouver la première contrainte de transition, $log_2(e-1)$ appels pour trouver la seconde contrainte,... Comme k_e contraintes de transition doivent être trouvées, nous obtenons O($log_2(e).k_e$). □

En pratique, il est très probable que la fonction $dcMUC$ appelle moins souvent MAC que la fonction $csMUC$. Par

ailleurs, en étant plus précis, il apparaît que le nombre maximum d'appels à MAC par $dcMUC$ est borné plus exactement par $log_2(e).(k_e+1)$ ($+1$ car il faut prouver que les k_e contraintes de transition correspondent à un noyau minimal). Le pire des cas est donc favorable à $dcMUC$ (par rapport à $dsMUC$) lorsque $k_e < e/log(e) - 1$. Ce sera le cas lorsque la taille du noyau extrait ne sera pas élevée. Généralement, les instances insatisfiables de problèmes réels sont structurées et contiennent des noyaux minimaux de petite taille.

Une approche originale de type "diviser pour régner", appelé $QuickXplain$ a par ailleurs été proposée dans [13]. Dans le pire des cas, le nombre d'appels à MAC en utilisant cette approche est égale à $2k_e.log_2(e/k_e) + 2k_e$ [13]. Le pire des cas est favorable à $dcMUC$ (par rapport à $QuickXplain$) lorsque $2k_e/(k_e - 1).(log_2(k_e) - 1) < log_2(e)$. En prenant l'illustration donnée dans [13] avec $e = 2^{20}$ et $k_e = 2^3$, on obtient 288 appels avec $QuickXplain$ contre 180 appels avec $dcMUC$.

Finalement, pour des raisons d'efficacité, il est important d'utiliser, en pratique, une approche basée sur les conflits (fonction $wcore$) avant d'extraire un MUC (fonction $xxMUC$). Cela sera montré dans la section 6. Les méthodes que nous considérons pour la partie expérimentale sont (étant donné un réseau de contraintes P) :

- CS correspond à l'appel de csMUC(wcore(P))
- DS correspond à l'appel de dsMUC(wcore(P))
- DC correspond à l'appel de dcMUC(wcore(P))

Même si cela n'apparaît pas explicitement ci-dessus, nous considérons que toutes les contraintes sont renommées, avant l'appel de $xxMUC$, de façon à ce que l'ordre lexicographique corresponde à l'ordre décroissant du poids des contraintes. Notons aussi que nous pouvons arbitrairement borner le nombre d'appels à MAC par $wcore$ de façon à vérifier la proposition 1 pour les méthodes CS, DS et DC. En pratique, nous avons observé que le nombre d'appels à MAC par $wcore$ est toujours faible.

5 Travaux connexes

D'une part, les travaux de recherche sur l'extraction de noyaux dans les réseaux de contraintes restent limités. Bakker et al. [1] ont proposé une méthode d'extraction de MUC (dans le contexte du " Model-Based Diagnosis"). Cette méthode correspond exactement à appeler dsMUC(pcore(P)). Notre approche dcMUC(wcore(P) peut être interprétée comme un raffinement[2] de la leur par le fait d'utiliser une approche dichotomique ($dcMUC$) et une étape préliminaire basé sur les conflits ($wcore$) dont l'importance pratique sera commentée dans la section 6. D'autres travaux [12, 24] s'intéressent plus particulièrement à l'identification des ensembles de conflits Π-minimaux où Π représente un opérateur de propagation. Cependant, alors que l'extraction d'un MUC est une activité qui s'applique à l'ensemble du réseau, l'extraction d'un ensemble de conflits Π-minimal est une activité que se limite à une branche de l'arbre de recherche. En conséquence, de façon à conserver une incrémentalité dans le processus de propagation, l'algorithme proposé dans [12, 24] implique (au moins partiellement) une approche constructive. La (nouvelle) méthode QuickXplain de U. Junker [13] est basée sur une approche de type "Divide and Conquer" de même que celle proposée dans [20]. QuickXplain, qui est le composant proposant les explications dans l'outil industriel de la société *Ilog* pour la configuration basé sur les contraintes, peut être utilisé pour extraire des MUCs, mais cette méthode effectue, dans le pire des cas, un nombre d'appels à MAC supérieure à la notre dés lors que $2k_e/(k_e - 1).(log_2(k_e) - 1) < log_2(e)$. Cette condition est vérifiée pour toutes les instances traitées dans la partie expérimentale. PaLM (Propagation and Learning with Move) est un autre système de programmation par contraintes [14] qui est basé sur les explications et qui fournit à l'utilisateur des informations sur l'insatisfiabilité de l'instance. A notre connaissance, les informations données par le système ne sont pas nécessairement minimales. Dans le contexte des CSP distribués, la détection de noyaux insatisfiables est souvent utilisée pour générer des *nogoods*. Par exemple, dans l'implantation de l'algorithme MHDC [27], une approche basée sur une preuve est utilisée.

D'autre part, la recherche des noyaux insatisfiables dans le domaine de la satisfaction de formule logique propositionnelle (SAT) est plutôt active. Bruni et Sassano [4] ont proposé une recherche de noyaux adaptative pour récupérer une sous-formule insatisfiable de petite taille. La dureté des clauses est évaluée en fonction d'une analyse de l'historique de la recherche. En fixant un pourcentage de dureté des clauses, le noyau insatisfiable courant est étendu ou réduit jusqu'à qu'il devienne insatisfiable. Dans [29], l'utilisation d'une approche basée sur la preuve de résolution permet d'étendre le solveur *Zchaff* pour approximer un noyau insatisfiable (le noyau insatisfiable retourné n'est pas nécessairement minimal). Finalement, Lynce et Marques-Silva [17] ont proposé un modèle qui calcule un noyau insatisfiable *minimum*, c'est à dire un plus petit noyau insatisfiable en nombre de clauses.

Il existe de grandes similitudes entre identifier les MUCs et résoudre le problème Max-CSP qui implique de trouver une assignation complète minimisant le nombre de contraintes violées par cette assignation. En effet, la satisfaction d'une instance peut être obtenue en supprimant, soit les MUCs disjoints, soit les contraintes violées par une solution de l'instance pour le problème Max-CSP. Cependant, très souvent, on ne peut pas supprimer une partie du système modélisé et on préfère identifier l'origine de l'échec afin de réorganiser le système de façon adéquate.

[2] Il est important de noter que la pondération introduite dans [1] correspond à une préférence statique indiquée par l'utilisateur.

La relation entre ces approches duales a été étudiée dans le contexte SAT dans [3].

6 Expérimentations

De façon à montrer l'intérêt pratique de l'approche décrite dans cet article, nous avons réalisé des expérimentations avec $abscon$ sur un PC Pentium IV 2,4GHz 512Mo sous Linux. Les performances ont été mesurées en termes de nombre d'exécutions (#runs) et de temps cpu en secondes (cpu). Nous indiquons aussi le nombre de contraintes (#C) (et, parfois, de variables (#V)) des instances et des noyaux extraits.

Nos expérimentations ont été réalisées sur des instances aléatoires et des instances issues du monde réel. Les instances aléatoires insatisfiables correspondent à deux ensembles, notés ehi-85-297 et ehi-90-315, contenant chacun 100 instances. Les instances issues du monde réel correspondent aux archives RLFAP et FAPP. Les instances du problème d'allocation de fréquences de liaisons radios (RLFAP) proviennent du CELAR (Centre électronique de l'armement) alors que les instances du problème d'allocation de fréquences avec polarisation (FAPP) proviennent du challenge ROADEF'2001. Certaines de ces instances ont été utilisées comme bancs d'essai[3] lors de la première compétition de solveurs CSP [28].

Nous avons d'abord étudié l'intérêt pratique de $wcore$ par rapport à $pcore$. Le tableau 1 indique la taille des noyaux extraits par les deux méthodes pour plusieurs instances. On peut observer qu'il est vraiment intéressant d'utiliser $wcore$ car la taille des noyaux extraits est très faible et le coût additionnel pour exécuter des recherches successives n'est guère pénalisant. Ceci est illustré avec une instance reines-cavaliers, une instance aléatoire 3-SAT et une instance RLFAP. Notons que la différence entre les deux méthodes peut être négligeable pour d'autres instances (non indiquées ici).

Ensuite, nous avons comparé la méthode DC avec les méthodes CS et DS. Pour les instances aléatoires EHI (les coûts moyens sont donnés dans la première partie du tableau 2), la différence entre la méthode DS et DC n'est pas très importante. Cela peut s'expliquer par le fait que le noyau extrait renvoyé par $wcore$ est toujours petit (cela a été observé pour toutes les instances EHI mais pas sur toutes les instances issues du monde réel). Cependant, pour les instances RLFAP et FAPP (voir la seconde et la troisième partie du tableau 2), la méthode DC surpasse clairement les méthodes CS et DS. A la fois, en termes de temps cpu et de nombre d'exécutions, la méthode DC est approximativement 10 fois plus efficace que les méthodes CS et DS. En fait, nous avons obtenu ce type de résultat pour la moitié des instances RLFAP et FAPP que nous avons testées.

[3] `http://cpai.ucc.ie/05/Benchmarks.html`

Enfin, nous avons utilisé les trois méthodes afin de supprimer itérativement, jusqu'à ce que la satisfiabilité soit atteinte, les MUCs disjoints (c'est à dire les MUCS qui ne partagent aucune contrainte) pour deux instances RLFAP. Cinq minutes ont été nécessaires pour atteindre l'objectif avec la méthode DC. Les ensembles de MUCs disjoints extraits se composent approximativement de 40 et 130 contraintes respectivement pour les instances $graph05$ et $scen11$-$f10$. Cela représente environ 3.5% de l'ensemble des contraintes de chaque instance.

7 Conclusion

Dans cet article, nous avons présenté un nouvelle approche, nommée DC, qui permet d'extraire les MUCs des réseaux de contraintes. L'originalité de cette approche tient au fait qu'elle exploite l'heuristique $dom/wdeg$ lors d'exécutions successives de l'algorithme MAC pour approximer un noyau insatisfiable, et une recherche dichotomique pour identifier les contraintes de transition successives appartenant au MUC. Nous avons introduit des arguments théoriques et pratiques qui valident notre approche. En particulier, l'approche DC implique un nombre d'appels à l'algorithme MAC borné par $log(e).(k_e + 1))$ dans le pire des cas et se montre assez efficace pour les instances RLFAP et FAPP issues du monde réel. En effet, un MUC a été extrait pour la plus part de ces instances en moins d'une minute[4]. Il n'est pas sans doute pas inutile de préciser qu'un certain nombre d'instances utilisées dans cet article ne peuvent être résolues, en un temps raisonnable, lorsque l'on utilise des heuristiques de choix de variables plus classiques (dom, $dom + deg$, dom/deg, ...).

En ce qui concerne la partie concernant l'extraction d'un noyau minimal, nous pensons que notre approche $dcMUC$ est un peu plus simple à implémenter que l'approche $QuickXplain$ [13]. De plus, dans le pire des cas, lorsque la taille du noyau est petite, le nombre maximum d'appels à l'algorithme MAC est moins élevé lorsqu'on utilise $dcMUC$. Toutefois, l'une des perspectives de ce travail est d'effectuer une comparaison directe entre les 2 méthodes. Nous envisageons également d'étudier l'extraction sur la base des variables puis des contraintes.

Remerciements

Ce papier a été soutenu par le programme Cocoa de la Région Nord/Pas-de-Calais et par l'IUT de Lens.

[4] En fait, nous pensons qu'il est possible d'améliorer très sensiblement notre implantation car nous sauvons et chargeons chaque instance intermédiaire en XML.

Instance	pcore		wcore	
	cpu	#C	cpu	#C
qk-25-25-5-mul (#C = 435)	100.1	427	107.7	32
ehi-85-297-0 (#C = 4,094)	2.93	3,734	3.01	226
$graph$-14-$f28$ (#C = 4,638)	4.23	3,412	4.69	503

TAB. 1 – Coût (cpu) et taille (#C) des noyaux extraits par $pcore$ et $wcore$

Instance	Méthode	cpu	#runs	MUC	
				#V	#C
instances EHI (100 instances par séries)					
$ehi-85-297$	CS	263	1284	19	32
$\#V = 297$	DS	62	157	23	37
$\#C \approx 4100$	DC	45	153	19	32
$ehi-90-315$	CS	266	1294	19	32
$\#V = 315$	DS	64	156	23	36
$\#C \approx 4370$	DC	46	154	19	32
Instances RLFAP					
$graph$13-$w1$	CS	303	704	4	6
$\#V = 916$	DS	257	338	4	6
$\#C = 1479$	DC	39	55	4	6
$scen$01-$f9$	CS	615	1430	10	25
$\#V = 916$	DS	1,036	617	10	25
$\#C = 5548$	DC	66	118	10	25
$scen$02-$f25$	CS	145	588	10	15
$\#V = 200$	DS	130	311	12	28
$\#C = 1235$	DC	21	67	10	15
$scen$09-$w1$-$f3$	CS	468	576	6	8
$\#V = 680$	DS	533	480	6	8
$\#C = 1138$	DC	38	48	6	8
Instances FAPP					
$fapp$01-200-2	CS	1774	1723	9	9
$\#V = 200$	DS	662	396	9	9
$\#C = 1108$	DC	124	83	9	9
$fapp$03-300-5	CS	469	484	3	3
$\#V = 300$	DS	333	290	3	3
$\#C = 2326$	DC	57	37	3	3
$fapp$06-500-1	CS	394	393	3	3
$\#V = 500$	DS	306	195	3	3
$\#C = 3478$	DC	48	35	3	3
$fapp$10-900-1	CS	1206	688	4	4
$\#V = 900$	DS	743	365	4	4
$\#C = 6071$	DC	119	46	4	4

TAB. 2 – Extraction de MUCs pour les instances EHI, RLFAP et FAPP

Instance	Méthode	cpu	#runs	#MUCs
$graph$05	CS	5,175	8,825	5
$\#V = 200$	DS	1,996	2,010	5
$\#C = 1134$	DC	330	526	5
$scen$11-$f10$	CS	1,491	3,840	5
$\#V = 680$	DS	3,040	2,452	5
$\#C = 4103$	DC	263	562	5

TAB. 3 – Extractions successives de MUCs pour les instances RLFAP

Références

[1] R.R. Baker, F. Dikker, F.Tempelman, and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 276–281, 1993.

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

[3] R. Bruni. Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Applied Mathematics*, 130 :85–100, 2003.

[4] R. Bruni and A. Sassano. Detecting minimaly unsatisfiable subformulae in unsatisfiable SAT instances by means of adaptative core search. In *Proceedings of SAT'00*, 2000.

[5] H. Kleine Buning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107(1-3) :83–98, 2000.

[6] M. Garcia de la Banda, P.J. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of PPDP'03*, 2003.

[7] J.L. de Siqueira and J.F. Puget. Explanation-based generalisation of failures. In *Proceedings of ECAI'88*, pages 339–344, 1988.

[8] H. Fleischner, O. Kullmann, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoretical Computer Science*, 289 :503–516, 2002.

[9] M. Ginsberg. Dynamic backtracking. *Artificial Intelligence*, 1 :25–46, 1993.

[10] B. Han and S-J. Lee. Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. *IEEE Tansactions on Systems, Man and Cybernetics*, 29(2) :281–286, 1999.

[11] T. Hulubei and B. O'Sullivan. Search heuristics and heavy-tailed behaviour. In *Proceedings of CP'05*, pages 328–342, 2005.

[12] U. Junker. QuickXplain : conflict detection for abitrary constraint propagation algorithms. In *Proceedings of IJCAI'01 Workshop on modelling and solving problems with constraints*, pages 75–82, 2001.

[13] U. Junker. QuickXplain : preferred explanations and relaxations for over-constrained problems. In *Proceedings of AAAI'04*, pages 167–172, 2004.

[14] N. Jussien and V. Barichard. The palm system : explanation-based constraint programming. In *Proceedings of TRICS'0O workshop held with CP'00*, pages 118–133, 2000.

[15] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings of CP'00*, pages 249–261, 2000.

[16] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI'04*, pages 549–557, 2004.

[17] I. Lynce and J.P. Marques-Silva. On computing minimum unsatisfiable cores. In *Proceedings of SAT'04*, 2004.

[18] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1) :99–118, 1977.

[19] J.P. Marques-Silva and K.A. Sakallah. Conflict analysis in search algorithms for propositional satisfiability. Technical Report RT/4/96, INESC, Lisboa, Portugal, 1996.

[20] J. Mauss and M. Tatar. Computing minimal conflicts for rich constraint languages. In *Proceedings of ECAI'02*, pages 151–155, 2002.

[21] Y. Oh, M.N. Mneimneh, Z.S. Andraus, K.A. Sakallah, and I.L. Markov. AMUSE : A minimally-unsatisfiable subformula extractor. In *Proceedings of DAC'04*, pages 518–523, 2004.

[22] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28 :244–259, 1984.

[23] C.H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37 :2–13, 1988.

[24] T. Petit, C. Bessière, and J.C. Régin. A general conflict-set based framework for partial constraint satisfaction. In *Proceedings of SOFT'03 workshop held with CP'03*, 2003.

[25] P. Prosser. Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence*, 9(3) :268–299, 1993.

[26] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.

[27] M.C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, 161 :25–53, 2005.

[28] M.R.C. van Dongen, editor. *Proceedings of CPAI'05 workshop held with CP'05*, volume II, 2005.

[29] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formulas. In *Proceedings of SAT'03*, 2003.