Détection de Conflits pour la Résolution de Problèmes Sur-contraints

Thierry Petit
Ecole des Mines de Nantes,
4 rue Alfred Kastler,
44307 Nantes Cedex 3, France.
email: `thierry.petit@emn.fr`

Christian Bessière
LIRMM-CNRS
161 rue Ada
34392 Montpellier Cedex 5
email: `bessiere@lirmm.fr`

Jean-Charles Régin
ILOG,
1681 route des Dollines
06560 Valbonne, Sophia Antipolis
email: `regin@ilog.fr`

Résumé

Les problèmes sur-contraints ont été largement étudiés dans les années 90, et notamment les algorithmes de résolution de Max-CSP [Freuder et Wallace, 1992; Wallace, 1994; Verfaillie *et al.*, 1996; Affane et Bennaceur, 1998; Larrosa *et al.*, 1999]. Le principe commun à ces algorithmes est de calculer une borne inférieure du nombre de contraintes violées. Ils n'exploitent pas de mécanisme de propagation. Aussi, afin d'améliorer la résolution, des travaux récents visent à pallier ce défaut. Les algorithmes de "Soft Arc-Consistency" propagent des compteurs d'inconsistance [Larrosa, 2002; Schiex, 2000; 2002]. Une autre technique consiste à utiliser les algorithmes de filtrage des contraintes et la propagation afin d'identifier des ensembles de contraintes en conflit, appelés conflict-sets, qui sont disjoints les uns des autres [Régin *et al.*, 2001] ; le nombre de conflict-sets extraits est une borne inférieure. Dans cet article, nous plaçons cette technique dans un cadre plus général. Nous montrons qu'elle correspond à un cas polynomial d'un problème NP-Complet (HITTING SET). Nous présentons un deuxième cas polynomial où les conflict-sets peuvent avoir des contraintes en commun, et un troisième cas qui n'est pas polynomial mais tel que l'on puisse approximer le résultat polynomialement. Dans chaque cas nous fournissons l'algorithme de calcul de la borne inférieure et un schéma pour générer incrémentalement la collection de conflict-sets. Nous discutons des extensions de ce travail aux problèmatiques où l'on doit maintenir une borne inférieure calculée à partir de données non disjointes.

1 Introduction

Un réseau de contraintes $\mathcal{R}$ est un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ où $\mathcal{X}$ est un ensemble de variables et $\mathcal{D}$ est un ensemble de domaines tel qu'un domaine $D(x)$ soit associé avec chaque $x \in \mathcal{X}$. $D(x)$ définit les valeurs qui peuvent être affectées à x. $\mathcal{C}$ est un ensemble de contraintes. Chaque contrainte exprime une propriété devant être vérifiée par un sous ensemble de variables. Le problème de satisfaction de contraintes (CSP) consiste à trouver une affectation de valeurs à l'ensemble des variables du réseau (appelée instanciation des variables) telle que toutes les contraintes soient satisfaites.

Cependant, il arrive souvent qu'un CSP n'ait pas de solution. Dans ce cas on cherche un compromis, en tolérant que certaines contraintes soient violées par la solution. Le problème Max-CSP [Freuder et Wallace, 1992] consiste à trouver une instanciation minimisant le nombre de contraintes violées, appelées violations. La formulation de ce problème en logique propositionnelle est Max-SAT [Garey et Johnson, 1979]. Bien que ce paradigme soit très simple, au moins deux arguments justifient l'étude d'algorithmes de résolution de Max-CSP : 1. la fréquence de sous-problèmes correpondant à des Max-CSP dans les applications réelles. 2. la possibilité d'encapsuler ces sous-problèmes dans des cotnraintes globales dans lesquelles les algorithles de résolution sont intégrés sous la forme d'algorithmes de filtrage [Régin *et al.*, 2000; Lemaître *et al.*, 2001].

Les algorithmes de résolution de Max-CSP sont généralement formulés pour suivre un schéma de recherche de type Branch and Bound. En rálité on peut adapter ces algorithmes à n'importe quelle stratégie de recherche [Régin *et al.*, 2001], mais leur présentation est plus abordable sous cette forme.

Une exploration "en profondeur d'abord" de l'arbre de recherche est effectuée. Les nœuds internes représentent des instanciations partielles. Une feuille terminant une branche de profondeur $|\mathcal{X}|$ correspond à une instanciation complète. A chaque nœud, P (past) est l'ensemble des variables déjà instanciées, et F (future) est l'ensemble complémentaire $\mathcal{X} \setminus P$. On appelle *distance* le nombre de contraintes telles que toutes les variables impliquées appartiennent à P et qui sont violées. Le but est alors de minimiser un objectif UB, correspondant au nombre de violations de la meilleure solution trouvée jusqu'alors. Si un nœud est tel que $distance \geq UB$ alors il n'est plus utile de poursuivre la recherche au dessous de ce nœud. Aucune instanciation complète obtenue à partir de l'instanciation partielle courante ne pourra améliorer l'objectif.

Cette condition est améliorée par le calcul de borne inférieure LB (lower bound) de la distance minimale n'importe quelle feuille de profondeur $|\mathcal{X}|$ située sous le nœud courant. PAr définition, $LB \geq distance$ car $distance$ est le nombre exact de contraintes violées qui impliquent des variables instanciées. Si $LB \geq UB$ alors il n'est pas utile de pousuivre la recherche sous le nœud courant car on n'améliorera pas la meilleure solution trouvée jusqu'alors. On notera que LB peut aussi être exploitée dans les algorithmes de filtrage des contraintes [1] [Larrosa *et al.*, 1999].

Les travaux récents visent à utiliser la propagation pour amĺiorer la valeur de la borne inférieure LB. On peut citer les algorithmes de "Soft Arc-Consistency" ([Larrosa, 2002; Schiex, 2000; 2002], initialement introduits dans [Bistarelli *et al.*, 1999]), ou les algorithmes basés sur la dt́ection de conflict-sets [Régin *et al.*, 2001]. Ces méthodes sont capables de détecter des violations qui étaient ignorées par l'algorithme de référence

[1] On définit ici un algorithme de filtrage comme un algorithme qui supprime des domaines des variables futures les valeurs ne pouvant appartenir à une solution.

PFC-MRDAC [Larrosa *et al.*, 1999]. Une technique combinant la borne inférieure basée sur le calcul de conflict-sets et celle de PFC-MRDAC a également été présentée dans [Régin *et al.*, 2001].

Dans ce papier, nous proposons de nouveaux algorithmes basés qur l'extraction de conflict-sets. Tout d'abord, nous rappelons quelques définitions. Nous montrons que la technique présentée dans [Régin *et al.*, 2001] est un cas particulier (polynomial) d'un problème NP-Complet. Nous présentons alors un deuxième cas polynomial dont on peut déduire une borne inférieure du nombre de violations, calculée à partir d'un algorithme de couplage de graphes. Nous proposons un troisème algorithme pour un cas non polynomial mais tel que l'on puisse approximer le résultat. Nous montrons comment générer incrémentalement les différentes collections de conflict-sets. Enfin, nous discutons des extensions possible de ce travail à d'autres problématiques où l'on doit calculer une borne inférieure en sommant des données non disjointes, comme par exemple l'algorithme PFC-MRDAC [Larrosa *et al.*, 1999], ainsi qu'aux weighted CSPs.

2 Détection de Conflict-sets

Un conflict-set est un ensemble de contraintes amenant à une contradiction [Jussien et Lhomme, 2001]. Aucune solution satisfaisant l'ensemble des contraintes ne contient un conflict-set. Dans cet article nous considérons la classe pareticulière des conflict-sets pouvant être identifiés en propageant les contraintes. Pour plus de clarté, nous supposerons que les algorithmes de filtrage des cotnraintes maintiennent la fermeture arc-consistante. Dans les théorèmes et définitions suivantes nous considérons un réseau de contraintes $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$.

Notation 1 *Soit $\mathcal{K} \subseteq \mathcal{C}$. $\mathcal{X}[\mathcal{K}]$ le sous-ensemble de variables de $\mathcal{X}$ apparaîssant au moins dans une contrainte de $\mathcal{K}$. L'ensemblde des domaines des variables de $\mathcal{X}[\mathcal{K}]$ est noté $\mathcal{D}[\mathcal{K}]$.*

Notation 2 *$\mathcal{D}' \sqsubseteq \mathcal{D}$ signifie que $\forall x_{i_k} \in \mathcal{X}, D'(x_{i_k}) \in \mathcal{D}'$ vérifie $D'(x_{i_k}) \subseteq D(x_{i_k})$.*

Définition 1 *Soit $C \in \mathcal{C}$ défini sur les variables $var(C) = \{x_{i_1}, \ldots, x_{i_k}\}$. Un élément de $D(x_{i_1}) \times \ldots \times D(x_{i_k})$ est un tuple de $var(C)$. La valeur v pour la variable x est notée (x, v). Un tuple τ de $var(C)$ est valide si $\forall (x, v) \in \tau, v \in D(x)$. La valeur $v \in D(x)$ est consistante avec une contrainte C ssi $x \notin var(C)$ ou $\exists \tau$ valide tel que $(x, v) \in \tau$. v est arc-consistante ssi $\forall C \in \mathcal{C}$, v est consistante avec C. Un domaine $D(x)$ est arc-consistant ssi $D(x) \neq \emptyset$ et toutes les valeurs de $D(x)$ sont arc-consistantes. On dit alors que x est arc-consistante. $\mathcal{R}$ est arc-consistante ssi toutes les variables de $\mathcal{X}$ sont arc-consistantes. $\mathcal{R}$ est arc-inconsistant ssi $\forall$ $\mathcal{D}' \sqsubseteq \mathcal{D}$, $\mathcal{R}' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ n'est pas arc-consistant.*

Définition 2 *Soit $\mathcal{K} \subseteq \mathcal{C}$. $\mathcal{K}$ est un conflict-set de $\mathcal{C}$ ssi $\mathcal{R}[\mathcal{K}] = (\mathcal{X}[\mathcal{K}], \mathcal{D}[\mathcal{K}], \mathcal{K})$ n'a pas de solution.*

Nous génèrerons des conflict-sets à partir d'un ensemble de contraintes en propageant les contraintes jusqu'à vider un domaine (fail).

Définition 3 *Soit $\mathcal{K} \subseteq \mathcal{C}$. $\mathcal{K}$ est un AC-Conflict-set de $\mathcal{C}$ ssi $\mathcal{R}[\mathcal{K}] = (\mathcal{X}[\mathcal{K}]$, $\mathcal{D}[\mathcal{K}]$, $\mathcal{K})$ est arc-inconsistant.*

Nous définissons un AC-conflict-set minimal au sens de l'inclusion :

Définition 4 *Un minAC-Conflict-set $\mathcal{K}$ est un AC-Conflict-set tel que $\forall C \in \mathcal{K}$, $(\mathcal{K} \setminus \{C\})$ ne soit pas un AC-conflict-set.*

Il est possible de réduire polynomialement un AC-Conflict-set en un minAC-Conflict-set [J-L. de Siqueira N. et Puget, 1988; Junker, 2001]. Le principe est décrit en section 4.1.

3 Bornes Inférieures pour Max-CSP

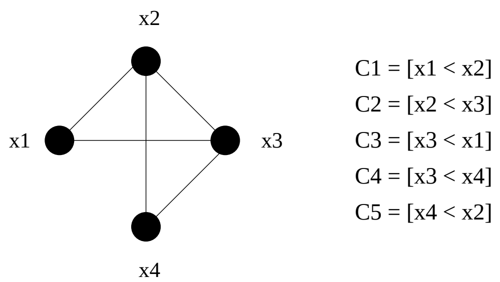
Dans cette section nous considérons uniquement des minAC-Conflict-sets.

3.1 Principe

A partir de la définition 2 il est possible de déduire que chaque minAC-conflict-set implique au moins une violation. L'idée intuitive est d'identifier plusieurs minAC-conflict-sets afin de calculer une borne inférieure de violations. Ces conflict-sets seront extraits à partir de contraintes impliquant des variables futures (pour les contraintes dont toutes les variables sont instanciées on sait si elles sont satisfaites ou non, et cette donnée est prise en compte par la *distance*).

Il n'est pas correct de considérer la cardinalité d'un ensemble de minAC-conflict-sets comme une borne inférieure à cause des contraintes communes à plusieurs conflict-sets. Considérons l'exemple suivant :

Exemple 1 *Soit $D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{0, 1, 2, 3\}$. $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$ où : $C_1 = [x_1 < x_2]$, $C_2 = [x_2 < x_3]$, $C_3 = [x_3 < x_1]$, $C_4 = [x_3 < x_4]$, $C_5 = [x_4 < x_2]$.*



Deux minAC-conflict-sets ont une contrainte en commun, $\mathcal{K}_1 = \{C_1, C_2, C_3\}$ and $\mathcal{K}_2 = \{C_2, C_4, C_5\}$, et le CSP défini par $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ admet une solution qui viole exactement une contrainte, C_2 : $\{(x_1, 1), (x_2, 2), (x_3, 0), (x_4, 1)\}$.

Dans cet exemple la condition nécessaire d'existance d'une solution qui viole uniquement une contrainte est que C_2 soit commune aux deux minAC-conflict-sets. Si une telle solution existe alors la contrainte violée est nécessairement C_2. Plus généralement, le problème de l'évaluation du nombre minimal de violations LB engendré par un ensemble de minAC-Conflict-sets est un problème NP-Complet. En effet, on recherche le nombre minimum de contraintes requises pour représenter tous les minAC-conflict-sets. Il s'agit du problème Hitting Set [Garey et Johnson, 1979] : étant donnée une collection de sous-ensembles d'un ensemble E, le but est de trouver un ensemble $E' \subseteq E$ de taille minimale tel que chaque sous-ensemble de la collection ait un représentant dans E'. Dans notre contexte, la cardinalité de E' founrit une borne inférieure.

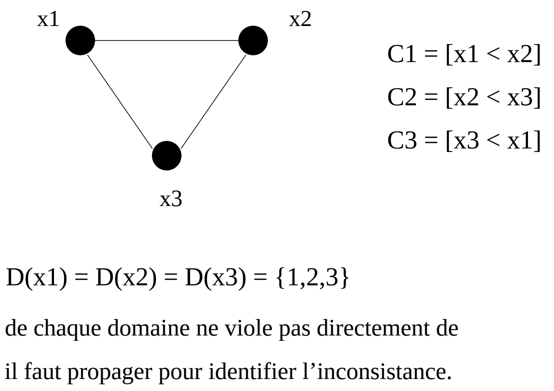
La complexité exponentielle du HITTING SET n'interdit pas d'utiliser des conflict-sets pour calculer une telle borne. En effet, il est possible de générer des collections particulières de minAC-Conflict-sets telles que le calcul de la borne soit polynomial, ou bien puisse être approximé polynomialement de façon raisonnable. Dans la suite de cette section nous présentons différentes collections et les algorithmes de calcul de borne correspondants.

3.2 Conflict-Sets Disjoints

Régin et al. [Régin *et al.*, 2000] ont proposé de garantir l'indépendance des violations comptées pour calculer la borne en considérant une collection Ψ de conflict-sets disjoints les uns des autres. Puisque les conflict-sets n'ont pas de contrainte en commun, la borne LB est égale au nombre de conflict-sets de la collection. La propriété d'avoir des conflict-sets minimaux n'est pas obligatoire mais elle permet d'obtenir une meilleure borne : plus petits sont les conflict-sets, plus grand est le nombre de contraintes pouvant être utilisées pour en calculer de nouveaux.

COMPUTELB(Ψ) {
  return $|\Psi|$;
}

Il a été montré que cette technique détecte des inconsistances ignorées par le meilleur algorithme "sans propagation", PFC-MRDAC [Larrosa *et al.*, 1999], qui fut longtemps considéré comme l'algorithme de référence. Par exemple, les cycles d'inégalités pouvant apparaître dans les problèmes d'ordonnancement (contraintes de précédence) sont identifiés comme des conflict-sets alors qu'ils étaient ignorés par PFC-MRDAC.

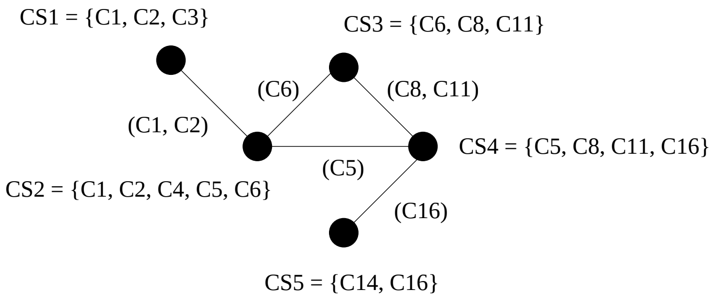


Une méthode de combinaison de cette borne avec celle de PFC-MRDAC est proposée dans [Régin *et al.*, 2001]. Elle reste valable pour les autres bornes proposés dans cet article.

3.3 Contraintes Communes à Deux Conflict-sets au Plus

Supposons que, étant donnés deux conflict-sets, aucune restriction ne soit faite sur le nombre de contraintes qu'ils partagent, mais que l'on impose qu'aucune de ces contrainte ne puisse appartenir à un troisième conflict-set. Dans ce cas on peut représenter la collection par un graphe tel que :

- Les sommets soient les conflict-sets conflict-sets,
- Chaque arête représente une contrainte ou un ensemble de contraintes partagées par deux conflict-sets (les deux conflcit-sets étant donc les deux sommets de l'arête).



La borne inférieure peut alors être calculée par un algorithme polynomial. En effet, on cherche un ensemble minimal d'arêtes nécessaires à couvir tous les sommets du graphe. Il s'agit du problème Edge Cover. Il est mentionné dans [Lawler, 1976; Garey et Johnson, 1979] que l'on peut résoudre ce problème par des algorithmes de couplage. Nous allons détailler cette affirmation. On appelle $d(x)$ le nombre d'arêtes inciedentes à un sommet x dans un graphe G (le degré de x).

Définition 5 *Soit un graphe $G = (\mathcal{X}, E)$. Un couplage est un sous-ensemble $E' \subseteq E$ tel qu'aucun couple d'arêtes de E' n'ai de sommet commun. $|E'|$ est la cardinalité du couplage. Un couplage de cardinalité maximale, généralement notée μ, est appelé un couplage maximal.*

Théorème 1 : Norman and Rabin, 1959 [Norman et Rabin, 1959]
Soit un graphe $G = (\mathcal{X}, E)$. Soit μ la cardinalité d'un couplage maximal de G. $|\mathcal{X}| - \mu$ est la cardinalité d'un Edge Cover *minimal de G.*

Nous proposons une preuve qui donne l'idée intuitive du résultat.

Proof : soit E' l'ensemble d'arêtes d'un Edge Cover minimal, de cardinalité $|E'|$. Soit $(u, v) \in E'$. Considérons $G' = (X, E')$. Soit $d(u) = 1$, soit $d(v) = 1$ dans G'. Si ce n'était pas le cas E' ne serait pas minimal, puisque en supprimant (u, v) on obtiendrait une couverture des sommets par les arêtes. Si on supprime depuis chaque sommet de degré > 1 toutes les arêtes incidentes sauf une alors on obtient par construction un couplage M de cardianlité m. Chaque arête de ce couplage couvre deux sommets donc le couplage couvre $2 * m$ sommets. Par définition, pour couvrir n'importe quel sommet hors du couplage une arête est nécessaire ; soit pour les couvrir tous $|X| - 2 * m$ arêtes. Donc $|E'| = m + |X| - 2 * m = |X| - m$. Par définition d'un couplage maximal μ' de G', on déduit $m \leq \mu'$, et à fortiori $m \leq \mu$. En conséquence, $|E'| \geq |X| - \mu$ (1). En outre, soit M un couplage maximal de cardinalité μ. $2 * \mu$ sommets sont couverts par M et

$|X| - 2 * \mu$ restent. Si on sélectionne une arête incidente à chaque sommet n'appartenant pas au couplage, le deuxième sommet de chaque arête appartient au couplage. Si ce n'était pas le cas celui-ci ne serait pas maximal puisqu'on pourait y ajouter des arêtes. En ajoutant ces arêtes à celle du couplage on obtient donc un ensemble d'arêtes formant un EDGE COVER. Sa cardinalité est $\mu + |X| - 2 * \mu = |X| - \mu$. D'où $|E'| \leq |X| - \mu$ (2). d'après (1) et (2) on sait que $|X| - \mu$ est exactement la cardinalité d'un EDGE COVER minimal.

L'algorithme suivant est issu de ce théorème. $\Psi = \{\mathcal{K}_1, \ldots, \mathcal{K}_{|\Psi|}\}$ est un ensemble de conflict-sets et E l'ensemble d'arêtes, chacune représentant une intersection non vide entre deux conflict-sets. Si le graphe a des sommets isolés on ajoute leur nombre à la borne (conflict-sets disjoints).

```
computeLB(G = (Ψ, E)) {
    LB ← 0
    for each x ∈ Ψ s.t. d(x) = 0 do
        LB ← LB + 1 ;
        Ψ ← Ψ \ {x} ;
    μ ← cardinality of a maximum matching in G ;
    LB ← LB + |Ψ| − μ ;
    return LB ;
}
```

N'importe quel nombre de contraintes peut être partagé par deux conflict-sets (en violer une est suffisant pour couvrir les deux conflcit-sets). Cette propriété renforce l'intérêt d'un tel calcul. La complexité pour trouver la cadinalité d'un couplage maximal de $G = (\Psi, E)$ est $O(|E| * \sqrt{|\Psi|})$ [Ahuja *et al.*, 1993]. Lorsque les contraintes sont communes à plus de deux conflict-sets le problème n'est plus polynomial.

3.4 Intersections Unaires entre Paires de Conflict-Sets

Le HITTING SET reste un problème NP-Complet même lorsque les sous-ensembles considérés sont de taille au plus 2 [Garey et Johnson, 1979]. On en déduit que, plus généeralement, même lorsque l'intersection entre deux conflict-sets est de au plus une contrainte, calculer une borne inférieure demeure un problème exponentiel. On propose donc une approximation, consistant à sous-estimer le nombre de contraintes requises pour prendre en compte tous les conflict-sets.

Soit $G = ((\mathcal{C}, \Psi), E)$ le graphe biparti[2] tel que une arête soit définie entre une contrainte C et un conflict-set $\mathcal{K} \in \Psi$ ssi $C \in \mathcal{K}$. On ordonne les contraintes par degré décroissant dans G. Considérons le sous-ensemble $\mathcal{C}'$ des premières contraintes selon l'ordre telles que :

– La somme des degrés des contraintes de $\mathcal{C}'$ est plus grande que $|\Psi|$
– $\forall$ $\mathcal{C}$" $\subset \mathcal{C}'$ la somme des degrés est strictement inférieure à $|\Psi|$.

Dans un tel graphe le nombre minimal possible de contraintes requises pour représenter tous les conflict-sets ne peut pas être inférieur à $|\mathcal{C}'|$. C'est pourquoi la cardinalité de $\mathcal{C}'$ est une borne inférieure. Cette technique peut être appliquée dans le cas général mais

[2]Un graphe $G = ((X, X'), E)$ est biparti ssi $\forall e = (u, v) \in E$, soit $u \in X$ et $v \in X'$ soit $u \in X'$ et $v \in X$.

elle fournit une borne précise si (presque) toutes les intersections sont vides ou unaires.

computeLB($G = ((\mathcal{C}, \Psi), E)$){
    $LB \leftarrow 0$;
    $sumDeg \leftarrow 0$;
    $L \leftarrow$ list of constraints $\in \mathcal{C}$ ordered by decreasing degree in G ;
    while $sumDeg < |\Psi|$ **do**
        pick and remove the first constraint C from L ;
        $sumDeg \leftarrow sumDeg+$ degree of C ;
        $LB \leftarrow LB + 1$;
    return LB ;
}

Etant donné Ψ, la complexité de l'algorithme est $O(|\mathcal{C}|)$. Cette approximation peut être reliée aux travaux Beldiceanu [Beldiceanu, 2001] (minimum constraint family).

4 Génération des Collections

Dans cette section nous présntons les techniques de génération de l'ensemble Ψ de min-AC-Conflict-sets utiles au calcul de la borne LB, en fonction de la propriété que l'on veut maintenir : conflict-sets disjoints, contraintes partagés au plus par deux conflict-sets, intersections unaires. Nous proposons un schéma incrémental basé sur les concepts présentés dans [Régin *et al.*, 2002] pour des conflict-sets disjoints. Deux ensembles sont maintenus :

- $\mathcal{A} \subseteq \mathcal{C}$ l'ensemble des contraintes disponibles pour générer de nouveaux conflict-sets
- Ψ la collection courante de conflict-sets.

Initialement, quand la recherche débute, $\Psi = \emptyset$ et $\mathcal{A} = \mathcal{C}$. La première fonction utile à tous les algorithmes que nous présentons est celle qui permet de maintenir des conflict-sets minimaux au sens de l'inclusion.

4.1 Minimisation d'AC-Confllict-sets

Notation 3 *Etant donnés un ensemble de contraintes $\mathcal{A}$ et un ordre total σ sur ces contraintes, on note $\mathcal{A}_\sigma$ l'ensemble contenant toutes les contraintes de $\mathcal{A}$ ordonnées par σ.*

La fonction computeAC-Conflict-Set($\mathcal{A}_\sigma$) retourne l'ensemble $\mathcal{K}$ des k premières contraintes de $\mathcal{A}_\sigma$ telles que le réseau $\mathcal{R}[\mathcal{K}_\sigma]$ soit arc-inconsistant et que le réseau que l'on obtient en considérant $\mathcal{K}_\sigma$ sauf sa dernière contrainte C_{last} soit arc-consistant (cette fonction sera décrite dans la section 4.2.2 car elle est dépendante de la collection). $\mathcal{K}$ forme un conflict-set mais il n'est pas nécessairement minimal. On sait seulement que C_{last} appartient au conflict-set minimal que l'on extraiera au final de $\mathcal{K}$.

Aussi, le schéma suivant est répété successivement : on considère un nouvel ordre σ' sur le constraintes, déduit du précédent en plaçant la dernière contrainte du conflict-set en première position. $\mathcal{K}$ est affecté avec le résultat de computeAC-Conflict-Set($\mathcal{K}_{\sigma'}$). Le nouveau conflict-set peut contenir moins de contraintes que le précédent si un domaine a été vidé en propageant un plus petit nombre de contraintes, à cause du changment de l'ordre où on les propage.

Le schéma s'arrête quand la dernière contrainte dans le conflict-set est à nouveau C_{last}. Un tour complet a été réalisé et par construction le dernier AC-Conflict-set généré ne peut pas être réduit davantage. C'est un minAC-Conflict-set.

4.2 Génération Incrémentale

A chaque nœud, on supprime tout d'abord des contraintes de Ψ pour les ajouter dans $\mathcal{A}$. Cette étape est nécessaire si on souhaite maintenir des conflict-sets minimaux. Le principe est expliqué ci-dessous.

4.2.1 4.2.1 Suppression des Contraintes de la Collection Ψ

Quand un minAC-Conflict-set $\mathcal{K}$ est généré et ajouté à Ψ, il est tel que $\forall C \in \mathcal{K}$, $(\mathcal{K} - \{C\})$ n'est pas un AC-conflict-set. Cependant, lorsqu'on poursuit la recherche, les domaines des variables peuvent être réduits. Les AC-Conflict-sets restent des AC-conflict-sets mais ils ne sont pas nécessairement minimaux. A partir de ces deux propriétés il est possible de mettre à jour l'ensemble Ψ au lieu de le recalculer systématiquement. Dans cette section nous nous intéressons aux contraintes qui peuvent être supprimées de Ψ afin d'être replaçées dans $\mathcal{A}$. La section suivante présentera les techniques de génération de nouveaux conflict-sets à partir de l'ensemble $\mathcal{A}$ qui a été mis à jour. On considère qu'il est nécessaire de toujours maintenir des minAC-Conflict-sets.

A. Schéma Général Nous décrivons tout d'abord un algorithme général commun à toutes les collections. Les parties spécifiques seront décrites par la suite pour chaque collection. Ainsi, l'algorithme appelle la procédure UPDATECTDATA, utile à la mise à jour des données nécessaires au maintien des propriétés requise par chaque collection (voir les paragraphes B., C., D.). L'algorithme appelle aussi la fonction COMPUTEMINAC-CONFLICT-SET définie en section 4.1.

UPDATEDATA(Ψ, $\mathcal{A}$) {
    for each $\mathcal{K} \in \Psi$ **do**
        $\mathcal{K}' \leftarrow$ COMPUTEMINAC-CONFLICT-SET($\mathcal{K}$) ;
        $\Psi \leftarrow (\Psi \setminus \{\mathcal{K}\}) \cup \{\mathcal{K}'\}$;
        for each $C \in \mathcal{K} \setminus \mathcal{K}'$ such that $C \notin \mathcal{A}$ **do**
            $\mathcal{A} \leftarrow \mathcal{A} \cup \{C\}$;
            UPDATECTDATA($\mathcal{K}$, $\mathcal{K}'$) ;
    return (Ψ, $\mathcal{A}$) ;
}

La complexité de cet algorithme est $|\mathcal{C}|^2$ fois le coût de propagation d'une contrainte (dans le pire des cas, lorsque la taille de $\mathcal{A}$ est négligeable par rapport à celle de Ψ). En effet, comme nous allons le voir, le coût de la procédure UPDATECTDATA n'influe pas sur la complexité globale. On notera que dans le cas de conflict-sets disjoints il n'est pas absolument nécessaire de les maintenir minimaux au sens de l'inclusion. On peut ainsi remplacer l'appel à la fonction COMPUTEMINAC-CONFLICT-SET($\mathcal{K}$) par un appel à COMPUTEAC-CONFLICT-SET($\mathcal{K}$, σ) avec un ordre σ arbitraire. A présent, nous définissons la procédure UPDATECTDATA en fonction de chaque collection.

B. Conflict-sets Disjoints Dans ce cas il n'est pas nécessaire de stocker quoi que ce soit concernant les contraintes.

UPDATECTDATA($\mathcal{K}$, $\mathcal{K}'$) { }

C. Contraintes Partagées au plus une fois Dans ce cas on maintient pour chaque $\mathcal{C} \in \mathcal{A}$ le nombre de conflict-sets où elle apparaît. Pour plus de clarté on considère une structure de données globale telle que pour chaque contrainte $C \in \mathcal{C}$ la valeur $\#\mathbf{CS}(C)$ soit le nombre de conflict-sets $\mathcal{K} \in \Psi$ tels que $C \in \mathcal{K}$.

UPDATECTDATA($\mathcal{K}$, $\mathcal{K}'$) {
    for each $C \in \mathcal{K} \setminus \mathcal{K}'$ **do**
        $\#\mathbf{CS}(C) \leftarrow \#\mathbf{CS}(C) - 1$;
}

D. Intersections unaires entre paires de Conflict-sets Dans ce cas on maintient le graphe biparti $G = ((\mathcal{C}, \Psi), E)$ tel qu'une arête soit définie entre une contrainte C et un conflict-set $\mathcal{K} \in \Psi$ ssi $C \in \mathcal{K}$. En supprimant les arêtes on maintient la propriété d'avoir des intersections au plus unaires. Pour des raisons de clarté on considère ce graphe comme une structure globale.

UPDATECTDATA($\mathcal{K}$, $\mathcal{K}'$) {
    remove vertex $\mathcal{K}$ from G and all its incident edges ;
    add a new vertex $\mathcal{K}'$ to G ;
    for each $C \in \mathcal{K}'$ **do**
        add $(C, \mathcal{K}')$ in E ;
}

4.2.2 4.2.2 Génération de Conflict-sets à partir de l'ensemble $\mathcal{A}$

Dans cette section nous dérivons comment augmenter la taille de la collection Ψ à partir de l'ensemble de contraintes $\mathcal{A}$.

A. Schéma Général L'algorithme utilise une routine qui teste si un ensemble de contraintes $\mathcal{K}$ est un AC-Conflict-set (par arc-consistance). Cette routine est appelée ISAC-CONFLICT-SET. Chaque conflict-set détecté dans $\mathcal{A}$ est ajouté à Ψ, et $\mathcal{A}$ et eventuellement les structures globales sont mises à jour. La fonction d'extraction d'un AC-Conflict-set en respectant la propriété requise par la collection est appelée COMPUTEAC-CONFLICT-SET. La fonction de mise à jour de $\mathcal{A}$ et des données globales est appelée UPDATEAFTERGENERATE. Elle admet $\mathcal{A}$ et le conflict-set extrait $\mathcal{K}$ comme argument. Ces deux fonctions sont spécifiques aux collections et elles seront décrites plus loin. Auparavant, nous décrivons la partie commune, l'algorithme COMPUTECOLLECTION de génération d'une collection de min-AC-Conflict-sets. L'ordre σ sur les contraintes est arbitraire, on le conserve ici afin d'avoir une unique définition de COMPUTEAC-CONFLICT-SET dans l'article (voir section 4.1).

```
computeCollection(𝒜_σ, Ψ) {
    𝒦 ← computeAC-Conflict-Set(𝒜_σ) ;
    while 𝒦 ≠ ∅ do
        𝒦 ← computeMinAC-Conflict-Set(𝒦) ;
        Ψ ← Ψ ∪ {𝒦} ;
        𝒜_σ ← updateAfterGenerate(𝒜_σ, 𝒦) ;
        𝒦 ← computeAC-Conflict-Set(𝒜_σ) ;
    return (𝒜_σ, Ψ) ;
}
```

B. Conflict-sets Disjoints Les contraintes de $\mathcal{A}_\sigma$ sont successivement propagées jusqu'à ce qu'un domaine soit vidé.

```
computeAC-Conflict-Set(𝒜_σ) {
    𝒦 ← {first constraint C in 𝒜_σ}
    𝒜_σ ← 𝒜_σ \ {C}
    while ¬ isAC-Conflict-Set(𝒦) ∧ 𝒜 ≠ ∅ do
        𝒦 ← 𝒦 ∪ {first constraint C in 𝒜_σ}
        𝒜_σ ← 𝒜_σ \ {C}
    if ¬ isAC-Conflict-Set(𝒦) then return ∅ ;
    return 𝒦 ;
}
```

Toutes les contraintes du minAC-Conflict-set $\mathcal{K}$ sont supprimées de $\mathcal{A}$.

```
updateAfterGenerate(𝒜_σ, 𝒦) {
    𝒜_σ ← 𝒜_σ \ 𝒦 ;
    return 𝒜_σ ;
}
```

La complexité de l'algorithme COMPUTECOLLECTION est $|\mathcal{A}_\sigma|^2$ fois le coût de propagation d'une contrainte.

C. Contraintes Partagées au plus une fois La fonction COMPUTEAC-CONFLICT-SET($\mathcal{A}$, σ) est identique à la précédente. La condition de suppresssion de C de $\mathcal{A}$ dépend de la valeur **#CS**(C), qui donne le nombre de conflict-sets $\mathcal{K} \in \Psi$ tels que $C \in \mathcal{K}$. **#CS**(C) (voir section 4.2.1).

```
updateAfterGenerate(𝒜_σ, 𝒦) {
    for each C ∈ 𝒦 do
        #CS(C) ← #CS(C) + 1 ;
        if #CS(C) = 2 then 𝒜_σ ← 𝒜_σ \ {C} ;
    return 𝒜_σ ;
}
```

L'ordre de complexité est identique au cas disjoint.

D. Intersections unaires entre paires de Conflict-sets On utilise le graphe biparti $G = ((\mathcal{C}, \Psi), E)$ tel qu'une arête soit définie entre une contrainte C et un conflict-set $\mathcal{K}$ $\in \Psi$ ssi $C \in \mathcal{K}$ (structure globale).

computeAC-Conflict-Set($\mathcal{A}_\sigma$) {
    $\mathcal{K} \leftarrow \emptyset$;
    $C \leftarrow$ first constraint C in $\mathcal{A}_\sigma$
    $\mathcal{A}_\sigma \leftarrow \mathcal{A}_\sigma \setminus \{C\}$
    add $\leftarrow$ **true** ;
    while $\neg$ isAC-Conflict-Set($\mathcal{K}$) $\wedge$ $\mathcal{A}_\sigma \neq \emptyset$ **do**
        for each $C' \in \mathcal{K}$ s.t. $C' \neq C$ **do**
            if $\exists\, \mathcal{K}' \in \Psi : C, C' \in \mathcal{K}'$ **then**
                add $\leftarrow$ **false** ;
        if add **then** $\mathcal{K} \leftarrow \mathcal{K} \cup \{C\}$;
        $C \leftarrow$ {first constraint C in $\mathcal{A}_\sigma$}
        $\mathcal{A}_\sigma \leftarrow \mathcal{A}_\sigma \setminus \{C\}$
        add $\leftarrow$ **true** ;
    if $\neg$ isAC-Conflict-Set($\mathcal{K}$) **then return** $\emptyset$;
    return $\mathcal{K}$;
}

La fonction updateAfterGenerate met à jour G en fonction de chaque nouveau conflict-set. Initialement, G contient toutes les contraintes comme des sommets isolés.

updateAfterGenerate($\mathcal{A}_\sigma$, $\mathcal{K}$) {
    add a new vertex $\mathcal{K}$ to G ;
    for each $C \in \mathcal{K}$**do**
        add $(C, \mathcal{K})$ in E ;
    return $\mathcal{A}_\sigma$;
}

La complexité de l'algorithme computeCollection est $|\mathcal{A}|^2$ le coût de propagation d'une contrainte.

5 Perspectives

5.1 Un Schéma Général

Les techniques présentées dans cet article pour traiter des conflict-sets peuvent être généralisées à d'autres algorithmes où une borne inférieure est calculée en effectuant une somme de données non indépendantes. Par exemple l'algorithme de résolution de Max-CSP PFC-MRDAC [Larrosa *et al.*, 1999] maintient d'une borne inférieure basée sur l'évaluation des violations directes de contraintes par des valeurs. Une valeur viole directement une contrainte ssi elle n'est pas consistante avec cette contrainte. Pour chaque variable future (i.e., non instanciée), le principe est de compter pour chaque valeur de son domaine le nombre de violations directes dans un sous ensemble de contraintes de $\mathcal{C}$. Les contraintes de ce sous-ensemble seront exclusivement prises en compte pour cette variable et ignorées pour toutes les autres. On prend alors le plus petit nombre de violations trouvé dans le domaine. On somme ces minima sur l'ensemble des variables, ce qui fournit une borne inférieure. La partition des contraintes (sous-ensembles disjoints de $\mathcal{C}$) garantit qu'aucune violation ne soit comptée plus d'une fois. Les principes de cet article pouraient être adaptés afin de traiter des sous-ensembles partageant des contraintes.

5.2 Weighted CSP

Le problème Weighted CSP [Freuder et Wallace, 1992] consiste à associer à chaque contrainte un poids, et à chercher une solution minimisant la somme des poids des contraintes violées. Le but est de distinguer différents degrés d'importance des contraintes et de favoriser la satisfaction des plus importantes. Afin d'adapter la technique basée sur des conflict-sets disjoints les uns des autres, il suffit de compter le poids de la contrainte de poids minimal dans chaque conflict-set[3]. Quand les contraintes peuvent être partagées, les algorithmes peuvent également être adaptés. Considérons deux minAC-Conflict-sets partageant au moins une contrainte. Afin d'effectuer la somme des poids, il n'est pas juste de considérer uniquement les contraintes communes à ces deux conflict-sets. En effet, il peut exister deux contraintes qui n'appartiennent pas à l'intersection dont la somme des poids est inférieure au plus petit poids d'une contrainte appartenant à l'intersection. Ainsi la contribution de ces deux conflict-sets sera la valeur minimale obtenue en comparant le poids minimal d'une contrainte partagée avec la somme des deux poids minimaux de contraintes dans chaque conflict-sets. Comme nous connaissons uniquement la cardinalité d'un Edge Cover, on cosidèrera la valeur minimale par rapport à toutes les intersections. On peut ainsi obtenir une borne inférieure des violations.

6 Conclusion

Nous avons proposé un schéma général de maintien incrémental d'une borne inférieure du nombre de violations pour le problème Max-CSP, basé sur la génération de conflict-sets. Ces techniques comportent l'avantage d'exploiter des mécanismes de propagation, et étend les algorithmes proposés dans [Régin *et al.*, 2001]. Nous discutons des extensions de ce travail à des algorithmes où l'on calcule une borne en effectuant une somme de données non indépendantes les unes des autres. Nous montrons l'extension aux Weighted CSP.

Références

[Affane et Bennaceur, 1998] M. S. Affane et M. Bennaceur. A weighted arc consistency technique for Max-CSP. *Proceedings ECAI*, pages 209–213, 1998.

[Ahuja *et al.*, 1993] R.K. Ahuja, T.L. Magnanti, et J.B. Orlin. Networks flows : theory, algorithms, and applications. *Prentice Hall, Inc.*, 1993.

[3] Un Max-CSP peut être codé comme un weighted CSP tel que toutes les contraintes aient un poids de 1.

[Beldiceanu, 2001] N. Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. *Proceedings CP*, pages 377–391, 2001.

[Bistarelli *et al.*, 1999] S. Bistarelli, P. Codognet, Y. Georget, et F. Rossi. Labeling and partial local consistency for soft constraint programming. *Second International Workshop on Practical Aspects of Declarative Languages, PADL'OO*, pages 230–248, 1999.

[Freuder et Wallace, 1992] E.C. Freuder et R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70, 1992.

[Garey et Johnson, 1979] M. R. Garey et D. S. Johnson. Computers and intractability : A guide to the theory of NP-completeness. *W.H. Freeman and Company*, ISBN 0-7167-1045-5, 1979.

[J-L. de Siqueira N. et Puget, 1988] J-L. de Siqueira N. et J-F. Puget. Explanation-based generalization of failures. *Proceedings ECAI*, pages 339–344, 1988.

[Junker, 2001] Ulrich Junker. QuickXplain : Conflict detection for arbitrary constraint propagation algorithms. *IJCAI'01 workshop on modelling and solving problems with constraints*, pages 75–82, 2001.

[Jussien et Lhomme, 2001] Narendra Jussien et Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, pages 21–45, 2001.

[Larrosa *et al.*, 1999] J. Larrosa, P. Meseguer, et T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107 :149–163, 1999.

[Larrosa, 2002] Javier Larrosa. Node and arc consistency in weighted CSP. *Proceedings AAAI*, pages 48–53, 2002.

[Lawler, 1976] E. Lawler. Combinatorial optimization : Networks and matroids. *Holt, Rinehart and Winston*, 1976.

[Lemaître *et al.*, 2001] M. Lemaître, G. Verfaillie, E. Bourreau, et F. Laburthe. Integrating algorithms for Weighted CSP in a constraint programming framework. *CP'2001 workshop on soft constraints, SOFT'O1, Paphos, Cyprus*, 2001.

[Norman et Rabin, 1959] R. Z. Norman et M. O. Rabin. An algorithm for minimum cover of a graph. *American Math. Soc.*, 10, 1959.

[Régin *et al.*, 2000] J-C. Régin, T. Petit, C. Bessière, et J-F. Puget. An original constraint based approach for solving over constrained prolems. *Proceedings CP*, pages 543–548, 2000.

[Régin *et al.*, 2001] J-C. Régin, T. Petit, C. Bessière, et J-F. Puget. New lower bounds of constraint violations for over constrained prolems. *Proceedings CP*, pages 332–345, 2001.

[Régin *et al.*, 2002] J-C. Régin, J-F. Puget, et T. Petit. Representation of soft constraints by hard constraints. *Proceedings JFPLC'02*, 2002.

[Schiex, 2000] T. Schiex. Arc consistency for soft constraints. *Proceedings CP*, pages 411–424, 2000.

[Schiex, 2002] T. Schiex. Une comparaison des cohérences d'arc dans les Max-CSP. *Proceedings JNPC*, pages 209–223, 2002.

[Verfaillie *et al.*, 1996] G. Verfaillie, M. Lemaître, et T. Schiex. Russian doll search for solving constraint optimisation problems. *Proceedings AAAI*, pages 181–187, 1996.

[Wallace, 1994] R.J. Wallace. Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. *Proceedings ECAI*, pages 69–77, 1994.