Algorithmes de Filtrage Spécifiques pour les Problèmes Sur-Contraints

Thierry Petit
ILOG-LIRMM,
1681 route des Dolines
06560 Valbonne, France.
tpetit@ilog.fr
tpetit@lirmm.fr

Jean-Charles Régin
ILOG,
1681 route des Dolines
06560 Valbonne, France.
regin@ilog.fr

Christian Bessière
LIRMM-CNRS,
161 rue Ada
34392 Montpellier Cedex 5, France.
bessiere@lirmm.fr

Résumé

Ces dernières années, de nombreux algorithmes de filtrage basés sur la sémantique des contraintes ont été proposés. Leur intérêt s'est avéré crucial pour la résolution des problèmes de satisfaction de contraintes (CSP). Dans ce papier, nous étendons ce concept aux problèmes sur-contraints. Nous proposons de prendre en compte la sémantique d'une contrainte afin de réduire plus efficacement les domaines des variables qui sont impliquées, même dans le cas où la violation de cette contrainte est autorisée. Nous présentons un paradigme dans lequel la sémantique des contraintes est préservée, et tel qu'aucune restriction ne soit faite sur l'utilisation d'algorithmes de filtrage spécifiques à certaines d'entre elles. Nous étudions la contrainte Alldiff, pour laquelle nous proposons deux algorithmes de filtrage exploitant des algorithmes de flot.

1 Introduction

Un problème est sur-contraint quand toutes les contraintes ne peuvent pas être simultanément satisfaites. Dans cette situation, le but consiste à trouver un compromis. On tolère que certaines contraintes soient violées dans les solutions, à condition que ces solutions conservent un intérêt pratique. Il est nécessaire de prendre en compte des règles et des critères d'acceptabilité définis par l'utilisateur. Notamment, un coût est associé à chaque contrainte afin de quantifier sa violation [BMR+99]. Les coûts peuvent alors être bornés. Par exemple, le coût associé à la violation d'une contrainte temporelle imposant qu'une personne termine sa journée de travail avant une certaine heure est proportionnel au dépassement horaire qu'elle effectue. Et ce temps supplémentaire ne doit pas être excessif.

En outre, un critère d'otpimisation, ou objectif, est généralement défini. Il est relatif à l'ensemble des coûts. Par exemple, le but peut être de minimiser la somme totale des coûts. Dans certaines applications il est nécessaire d'exprimer des règles plus complexes, qui impliquent plusieurs coûts indépendamment de l'objectif. De telles règles peuvent être ajoutées au problème sous la forme de méta-contraintes [PRB00]. Dans ce papier, nous nous intéressons à la résolution de ces problèmes.

Les algorithmes existants [FW92, VLS96, LMS99, Sch00, RPBP00] sont génériques : la sémantique des contraintes n'est pas prise en compte. Or, il est généralement nécessaire d'utiliser des algorithmes de filtrage spécifiques à certaines contraintes pour résoudre des problèmes réels. C'est la riason pour laquelle de nombreux algorithmes ont été proposés pour les contraintes globales [Rég94, BC94, Rég96]. En ce qui concerne les problèmes sur-contraints, ces algorithmes ne peuvent être utilisés que pour des contraintes dures, devant impérativement être satisfaites. En effet, un tel algorithme supprime des valeurs qui ne sont pas consistantes avec la contrainte. La condition de suppression est donc directement issue de l'obligation de satisfaire la contrainte. Elle n'est plus applicable quand on tolère sa violation.

En revanche, les domaines peuvent être réduits en fonction de l'objectif et des coûts associés aux violations. L'idée majeure de ce papier est de réaliser un tel filtrage de façon spécifique, c'est à dire de tirer profit de la sémantique de la contrainte et de la sémantique de son coût afin d'obtenir un algorithme de filtrage qui soit plus efficace. La condition de suppression sera liée à la nécessité d'avoir un coût acceptable, au lieu d'être liée à la satisfaction. Par exemple, soit la contrainte $x \leq y$, telle que $D(x) = [0, 3]$ et $D(y) = [0, 1]$. Supposons que, lorsque elle est violée, le coût soit défini comme étant la différence $x - y$. Si le coût doit être inférieur ou égal à 1, alors la valeur 3 peut être supprimée de $D(x)$. On note que le domaine des valeurs autorisées d'un coût peut se réduire en cours de recherche, par propagation de l'objectif et des coûts des autres contraintes.

Les paradigmes existants pour modéliser des problèmes sur-contraints [BMR$^+$99] imposent l'usage de contraintes définies en extension : chaque contrainte C et transformée en une contrainte définie par l'ensemble des tuples possibles (i.e., combinaisons de valeurs entre les variables impliquées dans C), et une valuation est associée à chaque tuple. Ainsi, les sémantiques de C et de son coût sont perdues. Pour cette raison, ces paradigmes ne sont pas les mieux adaptés à l'utilisation d'algorithmes de filtrage spécifiques[1]

Nous présentons un modèle tel que les contraintes puissent être exprimées dans leur forme naturelle, et leur sémantique préservée. En gros, le principe comsiste à définir un problème sur-contraint comme un problème d'optimisation classique. Toutes les fonctionnalités d'un solver de contraintes peuvent alors être utilisées, et, en particulier, les algorithmes de filtrages spécifiques à certaines contraintes.

Nous discutons des différentes quantifications possibles de la violation d'une contrainte. Chaque définition peut correspondre à un algorithme de filtrage différent.

Nous étudions la contrainte Alldiff [Rég94]: nous présentons deux algorithmes, relatifs

1. De plus, certaines contraintes ne peuvent pas être représentées en extension car la taille de l'ensemble des tuples est trop importante. Par exemple, une contrainte posée sur dix variables telles que la taille des domaines soit 100 contient 100^{10} tuples.

aux deux définitions possibles de son coût, basés sur la théorie des flots.

2 Préliminaires

2.1 CSP

Un *réseau de contraintes* $\mathcal{N}$ est defini par un ensemble de n *variables* $X = \{x_1, \ldots, x_n\}$, un ensemble de *domaines* $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ où $D(x_i)$ est l'ensemble fini des *valeurs* possibles pour la variable x_i, et un ensemble $\mathcal{C}$ de *contraintes*. Une *contrainte* C sur l'ensemble ordonné des variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ est un sous-ensemble $T(C)$ du produit Cartésien $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ qui spécifie les combinaisons de valeurs *autorisées* pour les variables $x_{i_1}, \ldots, x_{i_r}$. $D(X(C)) = \cup_{x \in X(C)} D(x)$. Un élément de $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ est appellé *tuple* sur $X(C)$. $|X(C)|$ est l'*arité* de C. C est *binaire* ssi $|X(C)| = 2$. La valeur a pour la variable x est notée (x, a). Un tuple τ sur $X(C)$ est *valide* si $\forall (x, a) \in \tau, a \in D(x)$. C est *consistante* ssi il existe un tuple τ de $T(C)$ qui est valide. Une valeur $a \in D(x)$ est *consistante avec* C ssi $x \notin X(C)$ ou il existe un tuple valide τ de $T(C)$ dans lequel a est la valeur affectée à x. Etant donné $Y \subseteq X$, une *instanciation* I de Y est une affectation de valeurs aux variables Y telle que $\forall x \in Y$, la valeur affectée à x appartienne à $D(x)$. Etant donné $Y \subseteq X$ et $C \in \mathcal{C}$ tel que $X(C) \subseteq Y$, une instanciation I de Y *satisfie* une contrainte C ssi la projection de I sur $X(C)$ appartient à $T(C)$. Le *Problème de Satisfaction de Contraintes* (CSP) consiste à trouver une instanciation I de X telle que $\forall C \in \mathcal{C}$, I satisfie C.

2.2 Problèmes Sur-Contraints

Il arrive souvent qu'un CSP n'ait pas de solution. Dans cette situation, on dit que le problème est *sur-contraint*. $\mathcal{C}_h \subseteq \mathcal{C}_s$ est l'ensemble des contraintes *dures*, qui doivent nécessairement être satisfaites. $\mathcal{C}_s = \mathcal{C} \setminus \mathcal{C}_h$ est l'ensemble des contraintes *molles*. Soit I une instanciation de X. Si I est une solution d'un problème sur-contraint alors $\forall C \in \mathcal{C}_h$, I satisfie C.

2.3 AllDiff

La contrainte *AllDiff* est la contrainte C imposant que les variables impliquées $X(C) = \{x_{i_1}, \ldots, x_{i_k}\}$ soient instanciées avec des valeurs toutes différentes les unes des autres : elle est définie par l'ensemble des tuples $tuples(C) = \{D(x_{i_1}), \ldots, D(x_{i_k})\} \setminus \{(d_1, \ldots, d_k) \in D(x_{i_1}) \times \ldots \times D(x_{i_k})$ tel que $\exists u, v$: $d_u = d_v\}$.

2.4 Definitions : Graphes

Un *graphe* $G = (X, E)$ est consitiué d'un ensemble de sommets X et d'un ensemble d'arrêtes E, tel que chaque arrête soit une paire de sommets distincts. Un *graphe orienté* $G = (X, U)$ est consitiué d'un ensemble de sommets X et d'un ensemble d'arcs U, tel que chaque arc (u, v) soit une paire ordonnée de sommets distincts. $\Gamma^-(v)$ est l'ensemble d'arrêtes entrant dans v dans le graphe G. $\Gamma^+(v)$ est l'ensemble d'arrêtes sortant de v dans G. $\Gamma(v) = \Gamma^- \cup \Gamma^+$. $G' = (X, E')$ est un *sous-graphe* de G ssi $E' \subseteq E$.

2.4.1 Couplages

Etant donné un graphe $G = (X, E)$, un *couplage* $M \subseteq E$ est un sous-ensemble d'arrêtes tel qu'aucune paire d'arrêtes n'ait un sommet en commun. La *cardinalité* d'un couplage est le nombre d'arrêtes qu'il contient. Un couplage de cardinalité maximale $\mu(G)$ est appellé un *couplage maximal.* Etant donné un couplage M, chaque arrête de E n'appartenant pas à M est *libre.* Chaque sommet v de X qui n'est pas l'extrémité d'une arrête de M est *libre.* Pour plus de lisibilité, si v n'est pas libre alors on note : $v \in M$.

2.4.2 Flots

La théorie des flots a été introduite par Ford and Flukerson [FF62]. Soit $G = (X, U)$ un graphe orienté, tel que à chaque arc (u, v) soit associé deux entiers positifs $lb(u, v)$ et $ub(u, v)$. $ub(u, v)$ est la *borne supérieure de capacité* de (u, v) et $lb(u, v)$ est la *borne inférieure de capacité* de (u, v). Un *flot* dans G est une fonction f satisfiant les deux conditions suivantes :
1. Pour tout arc (u, v), $f(u, v)$ représente la quantité de flot qui traverse l'arc (u, v), c'est à dire, de u vers v.
2. Une *loi de conservation* est vérifiée à chaque sommet : $\forall v \in X(G) : \sum_{u \in \Gamma^-(v)} f(u, v) = \sum_{w \in \Gamma^+(v)} f(v, w)$.
Le *problème* du flot compatible est le problème d'existence d'un flot dans G qui satisfie la *contrainte de cpacité*, c'est à dire : $\forall (u, v) \in U : lb(u, v) \leq f(u, v) \leq ub(u, v)$.

3 Propagation des coûts

3.1 Exemple Préliminaire

Soit C la contrainte $x \leq y$. Afin de quantifier sa violation, un coût, noté *cost*, est associé à C. La sémantique est la suivante : si C est satisfaite alors $cost = 0$; si C est violée alors $cost > 0$ et sa valeur est proportionnelle à l'écart entre x et y, i.e., $cost = x - y$.

Supposons que $D(x) = [90000, 100000]$ et $D(y) = [0, 200000]$, et que la contrainte suivante soit posée sur le coût : $cost \leq 5$. On déduit que $x - y \leq 5$, et, par propagation, $D(y) = [89995, 200000]$.

Cette déduction est effectuée directement en propageant les bornes des variables x, y et *cost*. Ce type de propagation peut être réalisé car il s'agit de contraintes d'inégalité. Plus généralement, cet exemple met en évidence l'intérêt d'intégrer les contraintes sur le coût dans le problème comme n'importe quelles autres contraintes, et dans leur définition la plus naturelle. Ceci implique que les coûts des violations soient exprimés par des variables.

La plupart des paradigmes existants [BMR+99] sont basés sur le principe suivant : chaque contrainte est définie en extension par l'ensemble de ses tuples (valides ou non), et une valuation (i.e., une valeur de coût) est associée à chaque tuple. L'objectif global prend en compte l'ensemble des coûts. La sémantique des contraintes est perdue, et les coûts ne sont pas exprimés par des variables intégrées au problème. Etant donnée une variable x impliquée dans une contrainte C, tester si $a \in D(x)$ peut être supprimée nécessite d'étudier la valuation de chaque tuple de C tel que a soit la valeur de x. Par exemple, réaliser le filtrage de $D(x)$ dans l'exemple est coûteux : au moins $|D(x)| * 89995 = 899950000$ tests sont nécessaires.

3.2 Nouveau Modèle

Le paradigme présenté dans cette section est adapté à l'utilisation d'elgorithmes de filtrage spécifiques aux contraintes. Pour des raisons de clarté, nous considérons que les valeurs de coût sont des entiers positifs. La valeur 0 exprime que C est satisfaite, et les valeurs strictement positives sont proportionnelles au degré d'importance de la violation. Cette hypothèse ne constitue pas une forte restriction, mais implique que les coûts appartiennent à un ensemble totalement ordonné. Si nécessaire, ce modèle peut être adapté à d'autres types de coûts. Nous proposons de résoudre un problème d'optimisation issu du problème initial. On considère le même ensemble de contraintes dures $\mathcal{C}_h$. En revanche, un ensemble de contraintes disjonctives remplace $\mathcal{C}_s$. Cet ensemble est noté $\mathcal{C}_{disj}$, et une bijection est définie entre $\mathcal{C}_s$ et $\mathcal{C}_{disj}$. Chaque disjonction implique une nouvelle variable $cost \in X_{cost}$, utilisée pour exprimer le coût de la violation de $C \in \mathcal{C}_s$. Ainsi, une bijection est également définie entre $\mathcal{C}_s$ et X_{cost}. Etant donnée $C \in \mathcal{C}_s$, la disjonction est la suivante :

$$[C \wedge [cost = 0]] \vee [\bar{C} \wedge [cost > 0]]$$

$\bar{C}$ est la contrainte incluant la variable de $cost$, qui exprime la sémantique du coût. Elle peut être définie en extension ou en intension, et un algorithme de filtrage spécifique peut lui être associé. Dans l'exemple préliminaire, les contraintes C et $\bar{C}$ sont respectivement $x \leq y$ et $cost = x - y$:

$$[[x \leq y] \wedge [cost = 0]] \vee [[cost = x - y] \wedge [cost > 0]]$$

Le nouveau problème n'est plus sur-contraint : il s'agit d'un problème d'optimisation classique. Il consiste à satisfaire les contraintes $\mathcal{C}_h \cup \mathcal{C}_{disj}$, et à optimiser un critère défini sur les variables X_{cost}. Les contraintes posées sur les coûts peuvent être directement propagées. Un point intéressant est que ce modèle peut être utilisé pour modéliser des problèmes sur-contraints dans un solver de contraintes classique [RPBP00]. En outre, des contraintes supplémentaires peuvent être posées sur les variables de coût, afin de sélectionner des solutions qui soient acceptables pour l'utilisateur [PRB00].

3.3 Consistance et Filtrage des Domaines

Dans cette section nous décrivons plus formellement les motivations de ce travail. Supposons que l'objectif soit de minimiser la somme (modélisée par la variable objectif obj) de m coûts $\{cost_1, ..., cost_m\}$, correspondant aux contraintes $\mathcal{C}_s = \{C_1, ..., C_m\}$.

Soit $C_i \in \mathcal{C}_s$, et soit $cost_i$ la variable de coût qui lui est associée. Nous nous intéressons au calcul d'une borne inférieure lb_i de la valeur minimale de $cost_i$ qui soit consistante avec $\bar{C}_i$. Cette borne peut être utilisée pour tester la consistance de $\bar{C}_i$. Si $lb_i > max(D(cost_i))$, ou si $lb_i + \sum_{k \in \{1,...,n\}} min(D(cost_k)) - min(D(cost_i)) > max(obj)$, alors un échec est détecté.

En outre, le même type de borne inférieure peut être employée pour filtrer les domaines des variables impliquées dans C_i. Soit x une variable de C_i, et $a \in D(x)$. Soit $lb_{i(x,a)}$ une borne de la plus petite valeur de $cost_i$ consistante avec $\bar{C}_i \wedge (x = a)$. Si $lb_{i(x,a)} > max(D(cost_i))$, ou si $lb_{i(x,a)} + \sum_{k \in \{1,...,n\}} min(D(cost_k)) - min(D(cost_i)) > max(obj)$, alors a peut être supprimée de $D(x)$.

On peut raisonner de façon similaire avec des objectifs différents d'une minimisation de la somme des coûts. En outre, pour certains problèmes, les coûts doivent être normalisés afin de les rendre comparables. Ils peuvent également être pondérés afin de favoriser la satisfaction de certaines contraintes plutôt que d'autres.

3.4 Définitions du Coût

Dans le cas le plus simple, le domaine de la variable *cost* est réduit à deux valeurs, qui expriment si la contrainte est satisfaite ou bien violée. Cependant, en général, ce n'est pas suffisant : la violation doit être quantifiée. Soit la contrainte $C : x \leq y$. Si $x > y$ alors la contrainte est violée, et le degré de cette violation est égal à le différence entre x et y. Le domaine de *cost* contient donc initialement l'ensemble des valeurs autorisées pour cette différence. En outre, il arrive que plusieurs définitions du coût soient envisageables. Elles dépendent de la nature du problème traité.

3.4.1 L'exemple de la contrainte Alldiff

Soit C une contrainte AllDiff définie sur l'ensemble $X(C) = \{x_1, x_2, x_3, x_4\}$, tel que $\forall i \in [1,4], D(x_i) = \{a, b, c, d\}$. Si nous ignorons les cas symétriques, en considérant qu'aucune valeur (resp. aucune variable) n'a plus d'importance qu'une autre, on peut énumérer les instanciations possibles : (1) : a, b, c, d, (2) : a, a, c, d, (3) : a, a, c, c, (4) : a, a, a, d, (5) : a, a, a, a.

Intuitivement, il est évident que la violation (5) est plus importante que la violation (2). Ce fait doit être exprimé le mieux possible par le coût associé à C.

3.4.2 Définitions

Nous proposons deux définitions génériques du coût associé à la violation d'une contrainte.

Définition 1 *Soit C une contrainte. Le coût de sa violation peut être défini comme étant égal aux nombre de minimal de variables qui devraient changer de valeur pour que C soit satisfaite.*

L'avantage de cette définition est qu'elle peut être appliquée à n'importe qu'elle contrainte (non binaire). Son défaut est que des affectations très différentes peuvent avoir le même coût. Dans l'exemple du AllDiff ci-dessus, les affectations (3) et (4) ont le même coût. Pour une contrainte Alldiff définie sur plus de quatre variables, de nombreuses instanciations différentes correspondent à la même valeur de coût.

Aussi, nous proposons une autre définition, que l'on peut appliquer seulement à des contraintes pouvant être représentées par un graphe primal [Dec92]:

Définition 2 *Le* **graphe primal** $Primal(C) = (X(C), E_p)$ *d'une contrainte C est le graphe tel que chaque arrête représente une contrainte binaire et tel que l'ensemble des solutions du CSP défini par $\mathcal{N} = (X(C), D(X(C)), E_p)$ soit l'ensemble des tuples autorisés de C.*

Pour une contrainte AllDiff C, $Primal(C)$ est un graphe complet où chaque arrête représente une inégalité binaire.

Définition 3 *Soit C une contrainte représentable par un graphe primal, $Primal(C)$. Le coût de sa violation peut être défini comme étant égal au nombre de violations de contraintes dans le CSP issu de $Primal(C)$.*
L'avantage de cette définition est qu'une valeur de coût différente est associée à chaque affectation possible de valeurs aux variables. Notamment, dans l'exemple, les coûts des affectations (3) et (4) sont différents. Un défaut est que certaines contraintes ne sont pas représentables par un graphe primal (par exemple la contrainte booléenne $C : p = q \vee r$).

4 Algorithmes pour la contrainte AllDiff

Soit $C \in \mathcal{C}_s$ une contrainte AllDiff et *cost* sa variable de coût. Nous utilisons par la suite les deux graphes bipartis suivants [2]:
Définition 4 : Graphe de Valeurs *Soit C une contraint AllDiff.*
$VG(C) = (X(C), D(X(C)), E)$ est le graphe tel que $(x, a) \in E$ ssi $a \in D(x)$.
On peut noter qu'une instanciation de $X(C)$ correspond à un graphe tel que pour chaque variable $x \in X(C)$ il existe exactement une arrête sortant de x. Si ce graphe correspond à un couplage, alors C is satisfaite.
Définition 5 : Graphe de Valeurs avec x=a *Soit C une contrainte Alldiff. Soit $x \in X(C)$ et $a \in D(x)$. $VG_{(x,a)}(C) = (X(C), D(X(C)), F \cup \{(x, a)\})$ est le graphe tel que $(y, b) \in F$ ssi $y \neq x$ et $b \in D(y)$.*

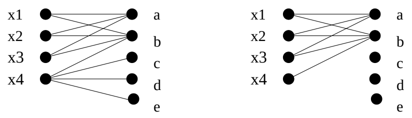

Figure 1: $D(x_1) = \{a, b\}, D(x_2) = \{a, b\}, D(x_3) = \{a, b\}, D(x_4) = \{b, c, d, e\}$.
le graphe de gauche est $VG(C)$, le graphe de droite est $VG_{(x4,b)}(C)$.

4.1 Première Définition du coût

$\bar{C}$ est une contrainte d'arité $|X(C)| + 1$, telle que la variable supplémentaire soit *cost*. D'après la définition 1, *cost* doit être égale au nombre de variables devant changer de valeur afin de satisfaire C.

4.1.1 Condition Nécessaire de Consistance de $\bar{C}$

Propriété 1 *Soit C une contrainte Alldiff et $cost \in X_{cost}$ sa variable de coût. Soit $\mu(VG(C))$ la cardinalité d'un couplage maximal de $VG(C)$. $lb = |X(C)| - \mu(VG(C))$ est une borne inférieure de cost.*

Preuve: L'existence d'une instanciation de $X(C)$ telle que C puisse être satisfaite en changeant la valeur de $N < lb$ variables implique l'existence d'un couplage de taille $X(C) - N$, i.e., strictement supérieur à $\mu(VG(C))$ (contradiction).
Théorème 1 *Si $lb > max(D(cost))$ alors $\bar{C}$ n'est pas consistante.*
Etant donné un graphe biparti, la cardinalité maximale d'un couplage peut être calculée polynomialement [AMO93], en $O(\sqrt{|X(C)|} * m)$, où $m = \sum |D(x)|, x \in X(C)$.

2. Un graphe biparti est un graphe tel que l'ensemble des sommets soit partitionné en deux sous-ensembles disjoints, et tel qu'il n'existe aucune arrête entre deux sommets d'un même sous-ensemble.

4.1.2 Algorithme de Filtrage

Le même principe peut être appliqué pour réduire les domaines des variables $X(C)$. Soit $x \in X(C)$ et $a \in D(x)$. On déduit[3], de la propriété 1, en considérant $VG_{(x,a)}(C)$:

Théorème 2 *Soit $x \in X(C)$ et $a \in D(x)$. Soit $\mu(VG_{(x,a)}(C))$ la cardinalité d'un couplage maximal de $VG_{(x,a)}(C)$. Soit $lb_{(x,a)} = |X(C)| - \mu(VG_{(x,a)}(C))$. Si $lb_{(x,a)} > max(D(cost))$ alors a peut être supprimée de $D(x)$.*

4.2 Deuxième Définition du coût

$\bar{C}$ est une contrainte d'arité $|X(C)|+1$, telle que la variable ajoutée soit *cost*. D'apreѐs la définition 3, *cost* doit être égale au nombre d'inégalités binaires violées dans le CSP défini par la représentation primale de C.

4.2.1 Condition Nécessaire de Consistance de $\bar{C}$

Notation 1 *Soit C une contrainte Alldiff, et I une instanciation de $X(C)$. $XP(C,I)$ est la partition de $X(C)$ telle que deux variables appartiennent au même ensemble ssi elles sont instanciées avec la même valeur.*

On peut visualiser $XP(C,I)$ en associant une couleur distincte à chaque valeur de $D(X(C))$. les sommets du graphe primal $Primal(C)$ peuvent être colorés en fonction de I. *cost* est la somme du nombre d'arrêtes des cliques de chaque couleur.

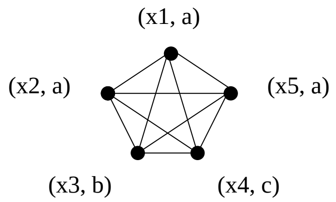

Figure 2: $XP(C,I) = \{\{x_1, x_2, x_5\}, \{x_3\}, \{x_4\}\}$. $cost = (3*2)/2 + 0 + 0 = 3$.

On peut noter que, d'apreѐs la partition $XP(C,I)$, on connait directement la cardinalité d'un couplage maximal de $VG(C)$: elle est égale au nombre d'ensembles de $XP(C,I)$.

Supposons à présent que toutes les variables ne soient pas instanciées. Soit $\mu(VG(C))$ la cardinalité d'un couplage maximal de $VG(C)$: si $\mu(VG(C)) = |X(C)| - 1$, alors on sait que l'on aura au moins un ensemble de taille 2 dans n'importe quelle partition correspondnant à une instanciation complète I de $X(C)$. On en déduit $cost \geq 1$; si $\mu(VG(C)) = |X(C)| - 2$, alors on aura deux possibilités : soit deux ensembles de taille 2 dans I (coût = 2), soit un ensemble de taille 3 (coût = 3). Ainsi, on sait que $cost \geq 2$. Plus généralement, si $\mu(VG(C)) = |X(C)| - k, k \geq 1$, alors k est une borne inférieure du coût, correspondant à une instanciation I telle qu'il n'y ait que des ensembles de taille ≤ 2 dans $XP(C,I)$. Cependant, nous n'avons aucune garantie concernant l'existence d'une telle partition I. Au contraire, si on est capable de détecter que la taille minimale possible pour le plus grand ensemble d'une partition est strictement suprérieure à 2, alors on peut améliorer la borne.

3. Un algorithme efficace (en terme de complexité), basé sur ce théorème, peut être déduit de la propriété suivante [Rég95]: si $lb < max(D(cost))$ alors toutes les valeurs des domaines des variables de $X(C)$ sont consistantes avec $\bar{C}$.

Considérons l'exemple d'une contrainte AllDiff C definie sur $X(C) = \{x_1, ..., x_8\}$. A partir des cardinalités possibles de couplages maximaux de $VG(C)$, de nombreuses instanciations I de $X(C)$ peuvent être obtenues, correspondant à différentes partitions $XP(C,I)$. Tous les cas non symétriques sont présentés dans le tableau ci-après, avec les coûts correspondants. "p of q" signifie p ensembles de q variables dans la partition $XP(C,I)$.

Taille d'un couplage maximal	Taille maximale d'un ensemble dans $XP(C,I)$	Ensembles possibles	Coût
1	8	1 of 8	28
2	4	2 of 4	12
	5	1 of 5 + 1 of 3	13
	6	1 of 6 + 1 of 2	16
	7	1 of 7 + 1 of 1	21
3	3	2 of 3 + 1 of 2	7
	4	1 of 4 + 2 of 2	8
	4	1 of 4 + 1 of 3 + 1 of 1	9
	5	1 of 5 + 1 of 2 + 1 of 1	11
	6	1 of 6 + 2 of 1	15
4	2	4 of 2	4
	3	1 of 3 + 2 of 2 + 1 of 1	5
	3	2 of 3 + 2 of 1	6
	4	1 of 4 + 1 of 2 + 2 of 1	7
5	2	3 of 2 + 2 of 1	3
	3	1 of 3 + 1 of 2 + 3 of 1	4
	4	1 of 4 + 4 of 1	6
6	2	2 of 2 + 4 of 1	2
	3	1 of 3 + 5 of 1	3
7	2	1 of 2 + 6 of 1	1
8	1	8 of 1	0

Cette exemple met en évidence le fait que différentes valeurs de coût puissent exister pour une taille de couplage maximal donnée, dépendant de la taille maximale d'un ensemble dans $XP(C,I)$. Nous proposons un algorithme pour calculer une borne inférieure de *cost*, basée sur le schéma suivant :

Etape 1: Calculer la cardinalité maximale $\mu(VG(C))$ d'un couplage de VG.

Etape 2: Déterminer la valeur minimale d'un ensemble de $XP(C,I)$ parmi toutes les instanciations possibles I pouvant être obtenues. Cette valeur sera calculée en fonction de $\mu(VG(C))$, et appellée $minMaxSetSize(VG, \mu(VG))$.

Nous pouvons alors déduire la valeur minimale de *cost* correspondant à $\mu(VG)$ et $minMaxSetSize(VG, \mu(VG))$. Intuitivement, il s'agit d'une borne inférieure car :

– il ne peut pas exister un couplage de cardinalité $s < \mu(VG)$ tel que $minMaxSetSize(VG, s) < minMaxSetSize(VG, \mu(VG))$.

– si $minMaxSetSize(VG, s) = minMaxSetSize(VG, \mu(VG))$ alors la plus petite valeur de *cost* pour un couplage de taille $\mu(VG)$ est strict. inférieure à celle d'un couplage de taille s.

Dans l'exemple, si $\mu(VG(C)) = 5$ alors la borne inférieure de *cost* varie de 3 à 6, selon la valeur de $minMaxSetSize(VG(C), \mu(VG(C)))$. Nous pouvons à présent définir plus formellement les concepts présentés ci-dessus, et décrire les algorithmes :

Propriété 2 *Soit M un couplage de $VG(C)$, $x \in X(C)$ une variable libre, et $a \in D(x)$. Si M est un couplage de cardinalité maximale, par définition a est l'extréminté d'une arrête de M.*

Preuve : Soit a n'est pas une valeur libre : dans ce cas $(x, a) \in M$. Soit a est une valeur libre, alors $M \cup (x, a)$ est un couplage de taille supérieure à la taille de M, et par conséquent M n'est pas maximal (contradiction).

D'après cette Propriété, nous déduisons que si $\mu(VG)$ est la cardinalité d'un couplage maximal, alors $|X(C)| - \mu(VG)|$ est une borne inférieure correcte de *cost*.

Cependant, comme nous l'avons précédamment souligné, la qualité de cette borne semble insuffisante. Nous proposons donc une méthode pour l'améliorer.

Définition 6 : Matching-Instantiation
Soit M un couplage. Une "matching-instantiation" M, notée $MI(M)$, est une instanciation de $Y \subseteq X(C)$ variables telle que :
- $(x,a) \in M \Rightarrow (x,a) \in MI(M)$;
- $(x \notin M \wedge (x,a) \in MI(M)) \Rightarrow a \in M$;

$MI(M)$ est une **matching-instantiation complète** *ssi $Y = X(C)$.*

Notation 2 *$setM(s)$ est l'ensemble des couplages de VG de taille s:*
- *$setMI(s)$ est l'ensemble des matching-instantiations complètes de $X(C)$ qui peuvent être obtenues à partir des couplages $setM(s)$.*
- *$minMaxSetSize(VG,s)$ est la plus petite valeur possible de la taille maximale d'une ensemble de la partition $XP(C,I) \in setMI(s)$.*

L'algorithme que nous proposons pour calculer $minMaxSetSize(VG,s)$ est basé sur le calcul de flots compatibles sur le graphe suivant :

Définition 7 : Value Assignment Graph
Soit C une contrainte Alldiff et $VG(C)$ son graphe de valeurs. Le "Value Assignment graph" de C est le graphe orienté $VAG(C) = (X(C) \cup D(X(C)) \cup \{s,t\}, E')$ tel que:
- *s est un sommet tel que $\forall x \in X(C)$, $(s,x) \in E'$.*
- *t est un sommet tel que $\forall v \in D(X(C))$, $(v,t) \in E'$.*
- *$(t,s) \in E'$.*
- *$\forall (x,v) \in E$, $(x,v) \in E'$.*

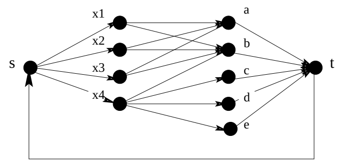

Figure 3: *Exemple de "Value Assignment Graph".*
$D(x_1) = \{a,b\}, D(x_2) = \{a,b\}, D(x_3) = \{ab\}, D(x_4) = \{b,c,d,e\}$.

Les bornes inférieures et supérieures de capacité dans E' sont :
- *$\forall (s,x)$, $x \in X(C)$, $lb(s,x) = ub(s,x) = 1$.*
- *$\forall (x,v)$, $x \in X(C)$, $v \in D(X(C))$, $lb(x,v) = 0$ et $ub(x,v) = 1$.*
- *$\forall (v,t)$, $v \in D(X(C))$, $lb(v,t) = 0$ et $ub(v,t) = maxSetSize$ où $maxSetSize$ est un entier tel que $2 \leq maxSetSize \leq |X(C)|$. On notera $(VAG(C), maxSetSize)$ pour spécifier la valeur de cet entier.*
- *$lb(t,s) = ub(t,s) = |X(C)|$.*

Le lien entre $VAG(C)$ et une instanciation est un flot satisfaisant la condition suivante : si une valeur v est affectée à une variable x alors $f(x,v) = 1$, sinon $f(x,v) = 0$.

Supposons que la cardinalité d'un couplage maximal soit strictement inférieure à $|X(C)|$.

L'idée majeure de l'algorithme est de tester un premier problème de flot compatible sur $VAG(C)$, tel que $maxSetSize$ soit égal à 2. Si il n'y en a pas, un nouveau problème de flot compatible est testé avec $maxSetSize$ ǵal à 3, et ainsi de suite jusqu'à ce que l'on trouve une valeur de $maxSetSize$ telle qu'il existe un flot compatible.

La valeur trouvée est $minMaxSetSize(VG(C), \mu(VG(C)))$, où $\mu(VG(C))$ est la cardinalité d'un couplage maximal dans $VG(C)$:

Propriété 3 *Soit s la taille d'un couplage maximal de $VG(C)$. Les couplages de taille $s-1$ satisfient $minMaxSetSize(VG(C), s-1) \geq minMaxSetSize(VG(C), s)$.*

Preuve: Soit M' un couplage de cardinalité $s-1$ pouvant être étendu à une matching-instantiation complète $MI(M')$ telle que la taille maximale d'un ensemble de la partition soit $minMaxSetSize(VG(C), s-1)$. Si un couplage de taille s existe, alors il existe au moins une valeur v libre dans M' (et d'après la définition 6, $v \notin MI(M')$) qui puisse être affectée à une variable appartenant à un ensemble non vide noté $Y \subseteq X(C)$. Alors : **(1)** Soit $\exists y \in Y$ partageant une valeur avec au moins une variable de $MI(M')$. Dans ce cas un couplage de cardinalité s peut être obtenu en affectant la valeur libre à y. Il satisfie $minMaxSetSize(VG(C), s-1) \geq minMaxSetSize(VG(C), s)$ (car un ensemble a été divisé). **(2)** Soit toutes les variables Y sont instanciées avec des valeurs non partagées dans $MI(M')$. Dans ce cas, on peut transformer $MI(M')$ en une instanciation I'' telle que la valeur libre v soit affectée à une variable $y \in Y$. I'' contient une nouvelle valeur libre, celle qui était précédamment affectée à y. I'' correspond à un couplage $s-1$, tel que la taille maximale d'un ensemble de la partition soit $minMaxSetSize(VG(C), s-1)$. On peut repéter le raisonnement ci dessus avec I'', et ainsi de suite, en sélectionnant à chaque étape une valeur v différente. L'existence d'un couplage de taille s garantit qu'au moins une fois la condition (**1**) sera vérifiée.

Propriété 4 *Soit $\mu(VG(C))$ la taille d'un couplage maximal de $VG(C)$. Les couplages de taille $s < \mu(VG(C))$ vérifient :*
$minMaxSetSize(VG(C), s) \geq minMaxSetSize(VG(C), \mu(VG(C)))$.

preuve : d'après la Propriété 3.

Etant donné $minMaxSetSize(VG(C), s)$, la fonction COMPUTEMINCOST retourne la valeur de *cost* correspondante. Elle peut être appellée avant la recherche afin de calculer tous les coûts possibles pour un nombre de variables donné $|X(C)|$.

Nous proposons ci-après une implémentation de cette fonction; la fonction EXISTSSIZE teste l'existence d'une taille d'ensemble pour une cardinalité de couplage et un nombre de variables donnés :

ISTOOLOW($setSize$, $\#vars$, s) : **1: retourner** $setSize * s < \#vars$;

ISTOOHIGH($setSize$, $\#vars$, s) : **1: retourner** $\#vars - setSize < s - 1$;

EXISTSSIZE($setSize$, $\#vars$, s)
1: retourner $\neg$ ISTOOLOW($setSize$, $\#vars$, s) $\wedge$ $\neg$ ISTOOHIGH($setSize$, $\#vars$, s);

$\textsc{computeMinCost}(setSize,\ X(C),\ s)$

1: si $\neg$ $\textsc{existsSize}$($setSize$,$X(C)$, s)) **alors retourner -1;**
2: si $setSize \neq 1$ **alors** $cost \leftarrow setSize * (setSize - 1)/2$;
3: $\#vars \leftarrow |X(C)| - setSize$;
4: $\#takenVals \leftarrow 1$;
5: tant que $\#vars > 0$ **faire**
  tant que $\neg$ $\textsc{isTooLow}$ ($setSize - 1$, $\#vars$, $s - \#takenVals$) **faire**
    $setSize - -$;
  si $setSize \neq 1$ **alors**
   $cost \leftarrow cost + setSize * (setSize - 1)/2$;
  $\#takenVals + +$;
  $\#vars \leftarrow \#vars - setSize$;
6: retourner $cost$;

Notation 3 *Soit s la taille d'un couplage de $VG(C)$:*
$cost(minMaxSetSize(VG(C), s), s)$ est la valeur retorunée par
$\textsc{computeMinCost}$*($minMaxSetSize(VG(C), s)$,$VG(C)$,s).*
Nous prouvons que si $\mu(VG(C))$ est la taille maximale d'un couplage de $VG(C)$, alors $cost(minMaxSetSize(VG(C), \mu(VG(C))), \mu(VG(C)))$ est une borne inférieure de $cost$:

Propriété 5 *Soit $\mu(VG(C))$ la cardninalité d'un couplage maximal de $VG(C)$. Soit $maxSetSize$ une taille maximale possible d'un ensemble (i.e., la fonction* $\textsc{existsSize}$ *retourne vrai)*
telle que $maxSetSize > minMaxSetSize(VG(C), \mu(VG(C)))$. On a :
$cost(maxSetSize, \mu(VG(C))) > cost(minMaxSetSize(VG(C), \mu(VG(C))), \mu(VG(C)))$.
Preuve : A partir de la partition correspondant à $maxSetSize$ on peut générer une partition correspondant à $minMaxSetSize(VG(C), \mu(VG(C)))$, en otant des variables des ensembles de taille strictement suprérieure à $minMaxSetSize(VG(C), \mu(VG(C)))$ et en les ajoutant à des ensembles de taille strictement inférieure à $minMaxSetSize(VG(C), \mu(VG(C)))$. Chaque transformation remplace une variable partageant une valeur avec au moins $minMaxSetSize(VG(C), \mu(VG(C)))$ par une variable partageant une valeur avec moins de $minMaxSetSize(VG(C), \mu(VG(C)))$ variables, entraînant une diminution du coût.

Propriété 6 *Soit $\mu(VG(C))$ la cardinalité d'un couplage maximal $VG(C)$.*
Les couplages de taille $s < \mu(VG(C))$ satisfient $cost(minMaxSetSize(VG(C), s), s)$
$\geq cost(minMaxSetSize(VG(C), \mu(VG(C))), \mu(VG(C)))$.
Preuve : D'après la propriété 4, $minMaxSetSize(VG(C), s) \geq minMaxSetSize(VG(C), \mu(VG(C)))$ (1). Soit $c = cost(minMaxSetSize(VG(C), s))$ et $MI \in setMI(s)$ de coût c. En divisant les ensembles dans MI jusqu'à ce que l'on ait $\mu(VG(C))$ valeurs utilisées, on obtient une instanciation issue d'un couplage de taille $\mu(VG(C))$ et telle que le coût soit inférieur à c. (2) D'après (1), (2) et la Propriété 5, la propriété est vérifée.

D'après les Propriétés 4 et 6 :
Corollaire 1 *Soit $\mu(VG(C))$ la taille d'un couplage maximale de $VG(C)$;*
$cost(minMaxSetSize(VG(C), \mu(VG(C))), \mu(VG(C)))$ est une borne inférieure de cost.
La borne inférieure est égale au plus petit coût possible correspondant à $minMaxSetSize(VG(C), \mu(VG(C)))$. Elle n'est pas optimale, car pour une cardinalité

donnée $\mu(VG(C))$ et une valeur $minMaxSetSize(VG(C), \mu(VG(C)))$, il peut exister plusieurs coûts possibles.

Par exemple, si $|X(C)| = 8$, $\mu(VG(C)) = 4$ et $minMaxSetSize(VG(C), \mu(VG(C))) = 3$, alors la valeur de *cost* peut être 5 ou 6.

Algorithme : ISCONSISTENT($\bar{C}$)

1: $\mu(VG(C)) \leftarrow$ **cardinalité d'un couplage maximal de** $VG(C)$**;**
2: $sizeSet \leftarrow 2$**;**
3: tant que $sizeSet \leq |X(C)|$ **faire**
  si il existe un flot compatible sur $(VAG(C), sizeSet)$ **alors**
    retourner COMPUTEMINCOST($sizeSet$, $X(C)$, $\mu(VG(C))$)$\leq max(D(cost))$**;**
  sinon $sizeSet++$**;**

La complexité de cet algorithme est $O(|X(C)| * \sqrt{|X(C)|} * m)$, où $m = \sum |D(x)|, x \in X(C)$.

4.2.2 Algorithme de Filtrage

Soit $x \in X(C)$ et $a \in D(x)$. Soit $(VAG_{(x,a)}(C) = (X(C) \cup D(X(C)) \cup \{s,t\}, E'')$ le sous-graphe de $VAG(C) = (X(C) \cup D(X(C)) \cup \{s,t\}, E')$ tel que $E'' = E' \setminus \Gamma_+(x) \cup \{(x,a)\}$:

Algorithme : FILTER($\bar{C}$)

1: pour tout $x \in X(C)$
  pour tout $a \in D(x)$
    $\mu(VG_{(x,a)}(C)) \leftarrow$ **cardinalité d'un couplage maximal de** $VG_{(x,a)}(C)$**;** $sizeSet \leftarrow 2$**;**
    tant que $sizeSet \leq |X(C)|$ **faire**
      si il existe un flot compatible sur $(VAG_{(x,a)}(C), sizeSet)$ **alors**
        si COMPUTEMINCOST($sizeSet$, $X(C)$, $\mu(VG_{(x,a)}(C))$) $> max(D(cost))$ **alors**
          $D(x) \leftarrow D(x) \setminus \{a\}$**;**
        $sizeSet = |X(C)| + 1$**;**
      sinon $sizeSet++$**;**

La complexité de cet algorithme est $O(|X(C)|^2 * \sqrt{|X(C)|} * m * d)$, où $m = \sum |D(x)|, x \in X(C)$ et $d = max(|D(x)|), x \in X(C)$.

5 Perspectives

Une des perspectives majeures est de fournir des algorithmes de filtrage pour d'autres contraintes globales. Certains d'entre eux devraient être similaires à ceux présentés dans ce papier. C'est le cas pour la contrainte de cardinalité globale [Rég96], lorsqu'on quantifie sa violation selon la définition 1. En effet, un algorithme consiste à calculer un flot maximal sur un "Value Assignment Graph" particulier. Dans ce graphe, les bornes supérieures des capacités des arcs sortant de sommets valeurs (arcs (v,t)) doivent être égales aux bornes supérieures des domaines des variables de cardinalité associées à chacune des valeurs. Les autres capacités sont indentiques à celles du graphe décrit dans la section 4.2.

En outre, un des thèmes futurs est l'expérimentation de ces algorithmes et leur combinaison avec les méthodes de résolution génériques existantes.

6 Conclusion

Ce papier introduit un paradigme pour la résolution des problèmes sur-contraints tel que des algorithmes de filtrage spécifiques aux contraintes puissent être utilisés. Deux définitions du coût que l'on peut associer à une contrainte afin de quantifier sa violation sont proposées. Une étude de la contrainte AllDiff est réalisée.

7 Remerciements

Ce travail a été partiellement financé par la Commission Européenne (projet ECSPLAIN (IST-1999-11969)). Nous remercions Olivier Lhomme, Jean-François Puget et les relecteurs des JNPC'2001 pour leurs commentaires et leurs remarques.

Références

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. Networks flows: theory, algorithms, and applications. *Prentice Hall, Inc.*, 1993.

[BC94] N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[BMR+99] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4:199–240, 1999.

[Dec92] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

[FF62] L. Ford and D. Flukerson. Flows in networks. *Princeton University Press*, 1962.

[FW92] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[LMS99] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for MAX-CSP. *Artificial Intelligence*, 107:149–163, 1999.

[PRB00] T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proceedings IEEE-ICTAI*, pages 358–365, 2000.

[Rég94] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proceedings AAAI*, pages 362–367, 1994.

[Rég95] J-C. Régin. Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique. *Ph.D. Dissertation, Université Montpellier II*, 1995.

[Rég96] J-C. Régin. Generalized arc consistency for global cardinality constraint. *Proceedings AAAI*, pages 209–215, 1996.

[RPBP00] J-C. Régin, T. Petit, C. Bessière, and J-F. Puget. An original constraint based approach for solving over constrained prolems. *Proceedings CP*, pages 543–548, 2000.

[Sch00] T. Schiex. Arc consistency for soft constraints. *Proceedings CP*, pages 411–424, 2000.

[VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimisation problems. *Proceedings AAAI*, pages 181–187, 1996.