Amélioration de l'expressivité des contraintes de table

Jean-Charles Régin
Université de Nice-Sophia Antipolis, I3S CNRS
2000, route des Lucioles - Les Algorithmes - BP 121
06903 Sophia Antipolis Cedex - France
jcregin@gmail.com

La complexité des algorithmes de cohérence d'arc pour les contraintes exprimées en extension dépend principalement du nombre de tuples impliqués. Aussi, plusieurs travaux ont été menés à bien afin de diminuer ce nombre. Notamment, Katsirelos et Walsh ont proposé de compresser les tuples. Ils ont intégré leur méthode dans GAC-Schema et montré comment on peut représenter un ensemble de F tuples interdits par un ensemble de ndF tuples compressés. Enfin, ils ont laissé ouvert la question de l'intégration des tuples compressés avec les meilleures versions de GAC-Schema comme celle de Lhomme et Régin. Dans ce papier, nous introduisons une généralisation des tuples compressés : les séquences de tuples. Nous montrons comment les meilleurs versions de GAC-Schema peuvent être aisément adaptées pour utiliser ces nouveaux tuples et comment F tuples interdits peuvent être facilement représentés par $F + 1$ tuples compressés, ce qui élimine tout besoin d'un algorithme particulier dans le cas de tuples interdits. Nous mentionnons aussi l'intérêt de notre approche au niveau de l'expressivité des contraintes de table.

1 Introduction

Les contraintes de table (Table constraints) font parties des contraintes les plus utilisées en PPC. Elles peuvent être définies à partir de la liste des combinaisons de valeurs autorisées (tuples autorisés), par la liste des tuples interdits ou par une fonction Booléenne équivalente à la satisfaction de la contrainte.

Un algorithme générique permettant de calculer la fermeture par arc consistance de ces contraintes a été développé. Il s'agit de GAC-Schema [1]. Il est basé sur la traversée de la liste des tuples autorisés jusqu'à en trouver un valide, c'est-à-dire tel que chaque valeur du tuple appartienne au domaine courant de sa variable. Sa complexité depend donc du nombre de tuples autorisés. Aussi, certains travaux récents ont été menés à bien afin de restructurer l'ensemble des tuples pour en diminuer le nombre [4, 5, 6, 2]. En particulier, Katsirelos et Walsh [5] ont proposé de compresser les tuples en utilisant des tuples "Glocal Cut Seed" (tuples GCS [3]). Un tel ensemble de tuples contient tous les tuples du produit Cartésien de sous-ensembles des domaines des variables. Par exemple, le tuple compressé $(\{1, 2\}, \{2\}, \{1, 2\})$ est équivalent à l'ensemble des tuples $\{(1, 2, 1), (1, 2, 2), (2, 2, 1), (2, 2, 2)\}$. En compressant la liste des tuples autorisés ils ont montré que la fermeture par arc consistance pouvait être accélérée. En outre, ils ont expliqué comment leur algorithme pouvait être intégré dans GAC-Schema.

Les travaux de Katsirelos et Walsh sont principalement focalisé sur l'algorithme calculant cette compression. Ils ne se sont pas réellement intéresser à l'expressivité de ces tuples compressés. Il sont seulement considérés ce point afin de représenter les tuples interdits à l'aide de tuples autorisés. Malheureusement, la méthode qu'ils proposent peut requérir nd fois plus de tuples compressés qu'il n'y a de tuples interdits.

Dans ce papier, nous proposons de montrer comment une simple généralisation des tuples GCS, que nous appellerons séquence de tuples, est beaucoup plus intéressante pour représenter des groupes de tuples. Une séquence de tuples est constituée d'un tuple GCS associé à un tuple minimum et un tuple maximum (pris par rapport à un ordre). L'avantage de ces ensembles de tuples est qu'il permettent de représenter facilement les tuples interdits car on peut grâce

à eux représenter facilement le complémentaire d'un ensemble.

Par exemple, considérons une contrainte de table définie sur des variables ayant pour domaines l'ensemble $\{a, b, c, d\}$ et dont la liste des tuples interdits est (a, b, c, d), (b, c, d, a) et (d, d, a, a). Pour simplifier, les tuples sont ordonnés de façon lexicographique. Nous pouvons définir une séquence de tuples qui représente les tuples du tuple (a, a, a, a) au tuple (a, b, c, c) qui est le tuple qui précède immédiatement le tuple (a, b, c, d). Cette séquence de tuple est simplement représentée par le tuple minimum $min = (a, a, a, a)$, le tuple maximum $max = (a, b, c, c)$ et le tuple GCS $(\{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\})$. Nous écrirons cette séquence : $((a, a, a, a), (a, b, c, c), (\{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\}))$. Bien sûr, nous pourrions aussi ne pas introduire d'autres valeurs que a pour la première variable, mais cela n'est même pas nécessaire. Ensuite, nous pouvons répéter cette méthode et définir la séquence de tuples $((a, b, d, a), (b, c, c, d), (\{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\}, \{a, b, c, d\}))$ représentant les tuples de (a, b, d, a) à (b, c, c, d) et la séquence de tuples qui représente les tuples de (b, c, d, b) à (d, c, d, d) et enfin une séquence de tuples correspondant aux tuples de (d, d, a, b) à (d, d, d, d). Ainsi, nous pouvons représenter F tuples interdits avec seulement $F + 1$ séquences de tuples. C'est une amélioration de la méthode de Katsirelos et Walsh par un facteur nd.

Nous allons aussi montrer que les séquences de tuples peuvent être utiles pour améliorer l'expressivité des contraintes de tables, parce qu'elle permettent de mélanger des tuples interdits et des tuples autorisés.

`Organisation de l'article.` Tout d'abord, nous rappellerons certaines définitions. Puis, nous introduirons la notion de cluster de tuples et nous définirons les séquences de tuples. Ensuite, nous expliquerons comment représenter un ensemble de tuples interdits par des séquences de tuples autorisés et nous montrons comment les séquences de tuples peuvent améliorer l'expressivité des contraintes de table. Avant de conclure, nous détaillerons l'intégration de séquences de tuples dans GAC-Schema.

2 Préliminaires

Réseau de contraintes. Un *réseau de contraintes fini* $\mathcal{N} = (X, \mathcal{D}, \mathcal{C})$ est défini 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* entre variables. La valeur a d'une variable x est souvent notée (x, a).

Contrainte. Une contrainte C portant sur l'ensemble ordonné de variable $X(C) = (x_{i_1}, \ldots, x_{i_r})$ est un sous-ensemble $T(C)$ du produit Cartésien $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ qui spécifie les combinaisons *autorisées* de valeurs pour les variables $x_{i_1}, \ldots, x_{i_r}$. Un élément de $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ est appelé un *tuple* sur $X(C)$ et $\tau[x]$ est la valeur de τ affectée à x. Les tuples sur $X(C)$ non autorisés par C sont appelés des tuples *interdits* de C. $|X(C)|$ est *l'arité* de C. Sans perte de généralité nous pouvons considérer que les tuples autorisés sont ordonnés par la relation $\prec$. Nous noterons d la taille du plus grand domaine initial et n l'arité.

Arc consistance. Soit C une contrainte. Un tuple τ sur $X(C)$ est *valide* si et seulement si $\forall x \in X(C), \tau[x] \in D(x)$; et τ est un *support* pour (x, a) ssi $\tau[x] = a$ et $\tau \in T(C)$. Une valeur $a \in D(x)$ est *consistante avec* C ssi $x \notin X(C)$ ou s'il existe un support valide pour (x, a). C est *arc consistant* ssi $\forall x \in X(C), D(x) \neq \emptyset$ et $\forall a \in D(x)$, a est consistant avec C.

Ordre Lexicographique. Un ordre total $<_d$ peut être défini sur $D(x), \forall x \in X$, sans perte de généralité. Deux tuples τ et τ' sur $X(C)$ peuvent être ordonnés par l'ordre lexicographique naturel $\prec_{lo}$ dans lequel $\tau \prec_{lo} \tau'$ ssi $\exists k$ tel que $\forall j \in [1..k-1], \tau[x_{i_j}] = \tau'[x_{i_j}]$ et $\tau[x_{i_k}] <_d \tau'[x_{i_k}]$.

3 Cluster de tuples

Un cluster de tuples est un concept général qui encapsule le concept de groupe de tuples à condition que l'on puisse les utiliser efficacement avec GAC-Schema.

Definition 1 *Un* **cluster de tuples** *est une structure de données correspondant à un ensemble de tuples associé à une opération efficace retournant un tuple valide de cet ensemble s'il existe et nil sinon.*

Les tuples "Global Cut Seed" (tuples GCS) proposés par Focacci et Milano [3] sont un exemple de cluster de tuples. Un tuple GCS est équivalent à l'ensemble des tuples obtenus en calculant le produit Cartésien de sous-ensembles des domaines des variables. Par exemple, le tuple compressé $(\{a, b\}, \{b, c\}, \{a, d\})$ est équivalent à l'ensemble de tuples $\{(a, b, a), (a, b, d), (a, c, a), (a, c, d), (b, b, a), (b, b, d), (b, c, a), (b, c, d)\}$.

Il est facile de vérifier si un tuple GCS contient un tuple valide ou pas : s'il n'y a pas d'intersection entre le domaine du tuple GCS et le domaine courant correspondant, alors au moins un tuple valide est contenu dans le tuple GCS. Il est aussi facile de vérifier si un tuple GCS contient un support pour la valeur (x, a) : on fixe le domaine de x à $\{a\}$ et on calcule les intersections entre les domaines du tuple GCS et ceux des va-

riables. S'il n'y a pas d'intersection vide alors le tuple GCS contient un tuple valide impliquant (x, a), c'est-à-dire un support pour (x, a). Cette opération peut être effectuée en $O(nd)$ dans le pire des cas, car tester si l'intersection de deux domaines n'est pas vide peut être fait en $O(d)$, pour des domaines de taille d. Nous verrons plus tard comment améliorer ce point.

Dans cet article, nous proposons une généralisation de tuples GCS : les séquences de tuples. Dans un tuple GCS il n'y a pas d'ordre et nous avons quelques problèmes pour exprimer certains sous-ensembles de tuples obtenus par les produits Cartésiens des domaines. Les séquences de tuples permettent de remédier à cet inconvénient en introduisant des bornes à cette énumération. Une séquence de tuples est un tuple GCS associé à un tuple minimum et un tuple maximum respectant l'ordre lexicographique. Autrement dit, il contient tous les tuples d'un tuple GCS qui sont plus grands ou égaux que le tuple minimum et plus petits ou égaux que le tuple maximum. Formellement, on a :

Definition 2 *Soit C une contrainte de table définie sur $X(C) = \{x_1, ..., x_n\}$.*
Une **séquence de tuples** *s définie par le triplet $(\tau^s_{min}, \tau^s_{max}, (D^s(x_1), D^s(x_2), ..., D^s(x_n)))$ est l'ensemble des tuples t tels que $t \in D^s(x_1) \times \cdots \times D^s(x_n)$ et $\tau^s_{min} \preceq_{lo} t \preceq_{lo} \tau^s_{max}$*

L'avantage principal des séquences de tuples est leur capacité à représenter de façon positive un ensemble de tuples interdits. Même s'il est possible de compresser les domaines, par exemple un seul joker ('*') peut représenter toutes les valeurs du domaine initial, nous devons considérer que la consommation mémoire d'une séquence de tuples est égale au nombre de valeurs qu'elle contient. Le minimum est n et le maximum est en $O(nd)$. Dans ce cas, il est important de rappeler qu'un nombre exponentiel de tuples peuvent être représenté par une seule séquence de tuples.

4 Tuples interdits

Nous allons montrer comment un ensemble de tuples interdits peut-être représenté par un ensemble de séquences de tuples autorisés. L'idée est de représenter cet ensemble par son complémentaire. Puis, nous verrons comment cet ensemble complémentaire peut être représenté par des séquences de tuples autorisés.

Nous utiliserons les notations suivantes :

- la valeur minimale de $D(x)$ est notée $\underline{D}(x)$, et sa valeur maximale est notée $\overline{D}(x)$.
- $\underline{\tau}(C)$ est le nombre minimum de combinaisons de valeurs sur $X(C)$: $(\underline{D}(x_1), ..., \underline{D}(x_n))$
- $\overline{\tau}(C)$ est le nombre maximal de combinaisons de valeurs sur $X(C)$: $(\overline{D}(x_1), ..., \overline{D}(x_n))$
- *n-comb*(t, C) est la combinaison de valeurs suivant immédiatement t pour $X(C)$ par rapport à l'ordre lexicographique. Elle est égale à *nil* si t est la dernière combinaison.
- *p-comb*(t, C) est la combinaison de valeurs précédant immédiatement t pour $X(C)$ par rapport à l'ordre lexicographique. Elle est égale à *nil* si t est la première combinaison.

Considérons une contraint de table C définie sur $X(C) = \{x_1, ..., x_n\}$ dont le domaine de x_i est $D(x_i)$, et $F = \{f_1, ...f_q\}$ un ensemble de tuples interdits ordonnés lexicographiquement. Soit$P$ le produit Cartésien des domaines des variables de $X(C)$, c'est-à-dire $D(x_1) \times \cdots \times D(x_n)$. Alors, l'ensemble des tuples autorisés est égal à $A = P - F$.

Soit $\mathcal{D}$ le tuple GCS constitué de tous les domaines des variables de $X(C)$. L'ensemble A peut être représenté par au plus $|F|+1$ séquences de tuples autorisés, comme suit :

- la première séquence de tuples est définie par ($\underline{\tau}(C)$,*p-comb*(f_1, C),$\mathcal{D}$).
- pour $i = 2, ..., n$ la séquence de tuples est définie par (*n-comb*(f_{i-1}, C),*p-comb*(f_i, C),$\mathcal{D}$)
- la dernière séquence de tuples est (*n-comb*$(f_n$,$\overline{\tau}(C)$,$\mathcal{D}$)

Bien entendu, la partie GCS de ces séquences de tuples peut être raffinée. On remarquera que certains tuples peuvent aussi être vides. C'est le cas lorsque le tuple minimum ou maximum d'une séquence est *nil* ou lorsque le tuple maximum est plus petit que le tuple minimum. Il n'est pas nécessaire de représenter de telles séquences.

Clairement, chaque séquence de tuples peut être représentée en $O(nd)$. Aussi, la consommation globale en mémoire est en $O(nd|F|)$. C'est supérieur à la représentation de $|F|$ tuples interdits qui est en $O(n|F|)$ (car un tuple contient n valeurs), mais c'est moins que la méthode de Katsirelos et Walsh qui requiert $O(nd|F|)$ tuples GCS et qui a donc une complexité mémoire en $O(n^2d^2|F|)$. En conclusion, nous gagnons un facteur nd par rapport à une représentation à l'aide tuples GCS et nous perdons un facteur d par rapport à la représentation explicite des tuples interdits.

Un autre avantage de notre approche par rapport à celle de Katsirelos and Walsh est sa grande simplicité.

5 Expressivité

L'idée principale des séquences de tuples est d'introduire une certaine structure dans la définition de

contraintes de table. Les contraintes de table sont fréquemment utilisées parce qu'elle peuvent exprimer n'importe quel type de contraintes, par exemple des règles entre des variables. En outre, la fermeture par arc consistance peut être calculée, même si cela peut s'avérer coûteux. En conséquence, grâce aux contraintes de table, nous pouvons exprimer des contraintes complexes et bénéficier d'algorithmes de filtrage associés puissants. L'inconvénient majeur est que le nombre de tuples définissant certaines contraintes de table peut-être tellement grand que l'on ne peut pas définir la contrainte. Le but des clusters de tuples est justement de contourner cet inconvénient en réduisant le nombre de tuples nécessaire pour exprimer certaines contraintes en extension. Les clusters de tuples permettent en plus de pouvoir facilement exprimer certaines contraintes. Nous proposons de détailler un peu ce point.

Dans la pratique, les contraintes de table sont bien souvent définies de plusieurs façons [1] :

- les tuples autorisés proviennent d'une base de données.
- toutes les solutions d'un sous-problèmes sont calculées et elles forment l'ensemble des tuples autorisées.
- l'utilisateur veut exprimer directement certaines compatibilités et incompatibilités entre des combinaisons de valeurs. Dans ce cas, il est utile de pouvoir exprimer que certaines combinaisons de valeurs sont autorisées alors que d'autres sont interdites. Par exemple, (x_1, a) ne peut être combiné qu'avec (x_2, b).

Dans le premier cas, les méthodes de compression sont certainement les meilleures méthodes pour représenter ces tuples. On peut aussi les utiliser pour le second point, mais il peut être plus intéressant de calculer l'ensemble des solutions de façon à pouvoir compresser cet ensemble. En d'autres termes, la méthode d'énumération devrait essayer de compresser la tuples pendant le calcul de toutes les solutions, par exemple sous la forme de séquences de tuples.

Nous proposons détudier plus précisément le dernier cas.

Comme nous l'avons vu, les tuples GCS peuvent être utilisés pour représenter certaines combinaisons de valeurs. Cependant, les séquences de tuples ont une expressivité plus forte car elles permettent de représenter efficacement des contraintes définies à la fois par des tuples autorisés et par des tuples interdits comme nous allons le montrer.

Considérons un ensemble A de séquences de tuples autorisés et un ensemble F de tuples interdits. Tout d'abord, on ordonne lexicographiquement l'ensemble des tuples interdits. Ensuite nous sélectionnons chaque séquence de tuples $s \in A$ et nous recherchons les tuples interdits qui sont contenus dans s. S'il y a k tuples interdits qui sont inclus dans s alors nous pouvons décomposer la séquence de tuples s en $k+1$ séquences de tuples avec une méthode similaire à celle que nous avons présentée dans la section précédante. Globalement, nous obtiendrons donc $|A| \times |F|$ séquences de tuples. Cette valeur est certainement un majorant du nombre de séquences réellement obtenues. Par exemple, si les séquences de tuples sont disjointes, ce qui est souvent le cas, alors nous aurons besoin de seulement $O(|A| + |F|)$ séquences, puisqu'un tuple interdit ne peut être contenu que dans une seule séquence de tuples autorisés.

Enfin, nous pourrions aussi considérer des séquences de tuples interdits, mais nous n'avons pas de méthode générale pour les combiner avec des séquences de tuples autorisés. Nous pouvons énumérer les tuples interdits et appliquer l'algorithme précédant et essayer de recombiner des séquences de tuples mais nous n'avons pas de garantie quant au nombre de tuples que nous obtiendrons à la fin. Ce problème mérite plus d'attention dans le futur.

6 Intégration dans GAC-Schema

Dans cette section, nous montrons comment les séquences de tuples peuvent être intégrées dans GAC-Schema. Tout d'abord nous montrons qu'une séquence de tuples est un cluster de tuples.

6.1 Tuple valide minimum

La recherche pour un tuple valide minimum [2] d'une séquence de tuples peut être effectuée en appliquant l'algorithme suivant :

1. On commence avec le tuple minimum de s (i.e. τ^s_{min}). Si τ^s_{min} est valide alors on le renvoie et on arrête l'algorithme. Sinon, on recherche le premier indice i tel que la valeur de τ^s_{min} impliquant x_i n'est plus dans le domaine de $D(x_i)$. C'est-à-dire que nous avons $\forall j = 1..i-1\ \tau^s_{min}[x_j] \in D(x_j)$ and $\tau^s_{min}[x_i] \notin D(x_i)$.
2. On recherche de i à 1, le premier indice j tel que $D^s(x_j)$ contienne une valeur valide plus grande que $\tau^s_{min}[x_j]$. Si un tel indice j n'existe pas alors on renvoie *nil* et l'algorithme s'arrête.
3. On crée un nouveau tuple t comme suit : pour $k = 1$ à $j-1$ on définit $t[x_k] = \tau^s_{min}[x_k]$; $t[x_j]$ contient

1. Il est aussi possible de définir une contrainte de table à partir d'un prédicat, mais cette méthode reste peu utilisée en pratique. Aussi, nous ne la considérerons pas.

2. le tuple valide minimum dans s est le tuple $t \in s$ tel que t est valide et il n'existe pas de tuple valide $t' \in s$ avec $t' \prec_{lo} t$.

la première valeur de $D^s(x_j)$ qui est valide et plus grande que $\tau^s_{min}[x_j]$ et pour $k = j+1$ à n on définit $t[x_k]$ comme la valeur minimum valide de $D^s(x_k)$. Si t est plus petit ou égal à τ^s_{max} alors l'algorithme renvoie t sinon il renvoie nil.

Par exemple, supposons que nous ayons $D(x_1) = \{a, b, c\}$,$D(x_2) = \{a\}$,$D(x_3) = \{a, b\}$ et $D(x_4) = \{a, b, c\}$ et $s = (t^s_{min} = (a, b, c, c), t^s_{max} = (c, b, b, b), (\{a, b, c\}, \{a, b, c\}, \{a, b, c\}, \{a, b, c\}))$. Alors, l'indice calculé par l'étape 1 est $i = 3$. D'après l'étape 2 nous calculons $j = 1$ et après l'étape 3 nous obtiendrons (b, a, a, a) qui est le premier tuple valide de s.

Clairement, cet algorithme est en $O(nd)$ parce que chaque étape est en $O(nd)$. Cela montre qu'une séquence de tuples est un cluster de tuples.

Il est aisé de rechercher le tuple valide minimum impliquant une valeur donnée (x, a). On a juste besoin de considérer que $D(x) = \{a\}$ dans l'algorithme. Aussi, nous pouvons facilement rechercher un nouveau support pour une valeur.

il est aussi intéressant de remarquer que cet algorithme peut être utilisé de façon incrémentale. Dans ce cas, nous pouvons prouver que la complexité en temps peut être amortie. Cet algorithme peut aussi être facilement modifiée pour recherche le tuple valide minimum plus grand qu'un autre tuple donné $\sigma \succeq \tau^s_{min}$: on a juste besoin de remplacer τ^s_{min} par σ dans l'algorithme. Donc, si pour une valeur (x, a) on doit effectuer plusieurs recherches, alors chaque recherche peut commencer là ou la précédante s'était arrêtée. De plus, il est clair que cela ne peut pas coûter plus que de traverser indépendamment toutes les tuples impliquant (x, a) et contenus dans la séquence. Cela signifie qu'utiliser des séquences de tuples ne peut pas coûter plus cher que de considérer indépendamment les tuples de ces séquences.

Nous pouvons maintenant détailler l'intégration dans GAC-Schema. Nous ne considérerons que la version de GAC-Schema qui traite les tuples autorisés.

6.2 Séquences de tuples au lieu de tuples

Calculer la fermeture par consistance d'arc signifie maintenir un support pour chaque valeur de chaque variable. Quand une valeur n'a plus de support alors elle peut être supprimée puisqu'elle n'est plus consistante avec la contrainte.

Il existe deux étapes dans les algorithmes généraux calculant la fermeture par consistance d'arc : une qui détermine les valeurs pour lesquelles un support doit être recherché ; et une autre qui recherche un nouveau support pour une valeur. La première étape est habituellement réalisée en associant à chaque valeur la liste de tuples contenant cette valeur et supportant une autre valeur. Ainsi, quand une valeur (x, a) est supprimée on connait les valeurs qui ont perdues leur support (parce que leur tuple support n'est plus valide) en traversant cette liste de tuples supportés. La seconde étape est la clé des algorithmes et les développements récents sont focalisés sur la rechercher d'un nouveau support pour une valeur donnée.

Détaillons ces deux étapes. Considérons qu'une valeur (x, a) a été supprimée de $D(x)$: pour toutes les valeurs (y, b) qui étaient supportées par un tuple contenant (x, a) un autre support doit être trouvé puisque le support courant n'est plus valide. Ces valeurs sont impliquées dans un tuple contenu dans une liste notée $S_C(x, a)$. GAC-Schema énumère tous les tuples dans les listes S_C et considère les valeurs valides supportées par ces tuples. Ensuite l'algorithme cherche un nouveau support pour elles.

La recherche d'un support valide est la tâche principale et la plus difficile de GAC-Schema. Par souci de clarté, nous ne parlerons pas dans cet article de l'aspect "multidirectionnel" de GAC-Schema et nous supposerons que la recherche d'un nouveau support est principalement basée sur l'étude des tuples autorisés de la contrainte. GAC-Schema considère successivement les tuples autorisés contenant la valeur pour laquelle on recherche un support jusqu'à en trouver un valide. Cela peut être fait efficacement en reliant entre eux les tuples autorisés contenant la même valeur. C'est-à-dire, que pour chaque valeur (y, b) nous définissons la liste des tuples autorisés l'impliquant. Cela coûte seulement un pointeur (vers le tuple suivant) pour chaque valeur de chaque tuple. C'est équivalent au coût mémoire pour représenter tous les tuples.

Il n'y a presque pas de changement si nous considérons des clusters de tuples au lieu de tuples : nous avons juste besoin de relier les deux notions. Quand nous considérons un tuple, nous avons aussi besoin de connaitre le cluster duquel il provient. L'ensemble de tuples devient un ensemble de clusters de tuples. Chaque valeur dans le cluster de tuples est reliée au prochain cluster de tuples qui la contient. Cela n'augmente pas la consommation mémoire puisqu'on ajoute seulement un pointeur pour chaque valeur de chaque cluster. Ainsi, lorsque l'on recherche un support pour (y, b), au lieu de traverser les tuples contenant (y, b), on traverse les clusters impliquant (y, b) en suivant les pointeurs associé à (y, b), jusqu'à ce qu'un tuple valide t appartenant au cluster courant soit trouvé. Un tel tuple valide est un support pour (y, b). Comme plusieurs tuples contenant (y, b) peuvent être valides dans le cluster courant on calcule le minimum en utilisant l'algorithme donné en section 6.1. Cette information est transmise à GAC-Schema ainsi que le cluster de tuples k d'où provient t. Autrement dit, une paire (t, k)

est renvoyée. Ensuite, GAC-Schema place t dans les listes S_C et utilise t et k pour calculer de nouveau supports.

Une paire (t, k) contient deux informations importantes qui vont s'avérer utiles pour contrôler la complexité en temps : k est le premier cluster de tuples contenant un tuple valide impliquant (y, b) et t est le tuple valide minimum contenant (y, b) dans k. Aussi, lorsque t ne sera plus valide on pourra continuer la recherche d'un support à partir de l'endroit où on s'était arrêté. On évite ainsi de répéter des calculs et on va pouvoir amortir la complexité en temps. De plus, on peut aller de cluster de tuples en clusters de tuples comme on le fait de tuples en tuples dans GAC-Schema. En conséquence, considérer des clusters de tuples au lieu de tuples peut économiser de la mémoire et n'augmente pas la complexité en temps dans le pire des cas. En pratique, cela améliore souvent l'algorithme.

Traverser la liste des tuples autorisés ou la liste des clusters de tuples autorisés contenant une valeur (y, b) lors de la recherche d'un support pour (y, b) est une méthode simple mais qui présente un inconvénient majeur : les domaines courants des variables ne sont pas considérés afin d'améliorer cette recherche. Les domaines sont utilisés uniquement pour tester la validité des tuples. En 2005, Lhomme et Régin [7] ont considérablement amélioré GAC-Schema en utilisant cette information sur les domaines pendant la recherche d'un nouveau support.

6.3 Intégration dans l'algorithme deLhomme et Régin

Dans GAC-Schema les tuples sont liés entre eux : chaque valeur est associée à un pointeur vers le prochain tuple la contenant. Lhomme et Régin ont montré que l'algorithme peut être accéléré si pour chaque tuple t et pour chaque valeur (x, a) on connait le premier tuple suivant t qui contient (x, a). Il est important de remarquer que nous avons besoin de cette information pour toutes les valeurs et pas seulement pour celles appartenant à t. Grâce à cette information, on va éviter de considérer de nombreux tuples de l'ensemble des tuples parce qu'on sait qu'ils ne peuvent pas être valides. En effet, le support pour une valeur (y, b) doit contenir (y, b) mais il doit aussi être valide, c'est-à-dire contenir des valeur valides pour les autres variables que y. La validité des valeurs est donc utilisée pendant la recherche de supports grâce à ce chainage.

Avec des tuples, il y a deux façons d'implémenter ce chainage :

- pour chaque tuple, on associe à chaque valeur de chaque variable, un pointeur qui représente le lien vers le prochain tuple contenant la valeur considérée. Cette méthode est particulièrement efficace en pratique. Cependant, elle multiplie la consommation mémoire par un facteur d ce qui est important !
- une structure de données complexe est utilisée : les "hologram tuples". Cette structure de données sacrifie une partie de la complexité en temps afin de réduire la complexité en espace. La consommation mémoire n'est pas augmentée mais on a besoin de $O(d)$ étapes pour accéder au pointeur suivant de n'importe quelle valeur.

Nous proposons d'appliquer les mêmes idées pour les clusters de tuples. Si on considère des clusters de tuples au lieu de tuples nous ne changerons pas l'algorithme. On associe à chaque valeur de chaque cluster de tuples un pointeur vers le prochain cluster de tuples contenant la valeur. Notons que si le cluster de tuple implique toutes les valeurs, nous pouvons atteindre le suivant de chaque valeur ! Si ce n'est pas le cas, alors nous pouvons ajouter cette information manquante, soit explicitement, ce qui augmente donc la consommation mémoire, soit en utilisant la structure de données des "hologram tuples" dans laquelle les tuples sont remplacés par des clusters. On remarquera que l'augmentation mémoire dans le premier cas est moins problématique avec les clusters de tuples qu'avec les tuples, parce qu'on dispose déjà de certaines informations dans les clusters puisqu'ils peuvent impliqués plus que n valeurs. Aussi, nous pouvons considérer qu'il n'y a pas de changement particulier si on utilise des clusters de tuples au lieu de tuples simples.

7 Discussion

Les séquences de tuples peuvent être généralisées en considérant pour chaque séquence de tuples un ordre spécifique pour les variables et un ordre spécifique pour les domaines de chaque variable. Cela ne changerait pas l'algorithme. Nous n'avons pas imposé de tels ordres parce que cela aurait complexifier les définitions et parce que nous n'avons pas trouvé d'exemple réel justifiant une telle modification.

Les techniques de compression, comme celle proposée par Katsirelos et Walsh pourrait aussi être utilisée dans notre cas, parce que les séquences de tuples intègre des tuples GCS. L'inconvénient de cette méthode est qu'elle peut être couteuse en temps (voir [5]). Néanmoins, il pourrait être intéressant d'examiner de plus près la possibilité de compresser des tuples dans des séquences de tuples au lieu de tuples GCS.

Nous avons aussi mentionné deux autres points qui méritent plus d'attention. Tout d'abord, quand la contrainte est définie à partir de l'ensemble des solutions d'un problème (habituellement définie par un

sous-ensemble de contraintes du problème général), on devrait essayer de compresser cet ensemble de solutions pendant qu'on les calcule. De cette façon, on pourrait améliorer le temps pour calculer toutes les solutions et obtenir un ensemble plus pertinent de tuples pour GAC-Schema. Ensuite, le problème de la combinaison d'ensembles de tuples interdits avec des ensembles de tuples autorisés devrait être considérée. Nous avons proposé une première méthode, mais elle est inefficace pour représenter certaines contraintes.

8 Conclusion

Dans cet article, nous avons introduit un nouveau type de cluster de tuples : les séquences de tuples. Ils généralisent les tuples GCS utilisés dans les contraintes de table par Katsirelos et Walsh. La représentation de tuples interdits par des séquences de tuples autorisés est simple et demande moins de mémoire que leur représentation par des tuples GCS. Il n'y a donc actuellement presque plus de raison pour utiliser une version dédiée de GAC-Schema pour les tuples interdits. Nous avons aussi montré que les séquences de tuples peuvent être utilisées pour mélanger des tuples autorisés et des tuples interdits de façon simple. Enfin, nous avons expliqué comment les clusters de tuples peuvent être facilement intégrés dans les meilleures implémentation de GAC-Schema comme celle de Lhomme et Régin.

Références

[1] C. Bessière and J-C. Régin. Arc consistency for general constraint networks : preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, Nagoya, 1997.

[2] K. Cheng and R. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15, 2010.

[3] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. CP'01*, pages 77–92, Paphos, Cyprus, 2001.

[4] I. Gent, C. Jefferson, I. Miguel, and P. Nightingale. Data structures for generalised arc consistency for extensional constraints. In *Proc. AAAI'07*, pages 191–197, Vancouver, Canada, 2007.

[5] G. Katsirelos and T. Walsh. A compression algorithm for large arity extensional constraints. In *Proc. CP'07*, pages 379–393, Providence, USA, 2007.

[6] C. Lecoutre. Optimization of simple tabular reduction for table constraints. In *Proc. CP'08*, pages 128–143, Sydney, Australia, 2008.

[7] O. Lhomme and J-C. Régin. A fast arc consistency algorithm for n-ary constraints. In *Proc. AAAI'05*, pages 405–410, Pittsburgh, USA, 2005.