La Contrainte Smart Table

Jean-Baptiste Mairy[1] **Yves Deville**[1] **Christophe Lecoutre**[2]

[1] ICTEAM, Université catholique de Louvain, Belgique
[2] CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
{jean-baptiste.mairy,yves.deville}@uclouvain.be lecoutre@cril.fr

Résumé

Cet article est un résumé en français de [5]. Les contraintes table sont très utiles pour la modélisation de problèmes combinatoires complexes en Programmation par Contraintes (PPC). Elles sont un moyen universel de représentation, mais malheureusement, la taille des tables peut grandir de manière exponentielle. Dans [5], nous proposons la possibilité d'introduire des contraintes arithmétiques simples au niveau des entrées des tables. Les tuples classiques de valeurs sont, dès lors, remplacés par des smart tuples. Il est possible de voir les contraintes smart table comme des combinaisons logiques de contraintes arithmétiques simples. Cette nouvelle forme de tuples permet un encodage compact de nombreuses contraintes, incluant une douzaine de contraintes globales connues. Dans [5], nous montrons également comment, avec la seule exigence d'acyclicité des smart tuples, un algorithme de filtrage de cohérence d'arc (généralisée) peut être conçu. Les expériences réalisées montrent que la contrainte table smart est un outil générique prometteur pour la PPC.

Les contraintes table donnent explicitement la liste des tuples acceptés (ou interdits). Cette liste est appelée la table de la contrainte. Ces contraintes peuvent théoriquement encoder n'importe quelle contrainte (d'une certaine manière, on peut les qualifier de contraintes universelles). Malheureusement, la taille des tables peut augmenter exponentiellement avec l'arité des contraintes, ce qui représente un problème de taille. Parmi les solutions proposées à ce problème, deux d'entre elles correspondent à une modification de la définition des tuples. Il s'agit des tuples compressés [3, 6, 8] et des supports courts appliqués aux contraintes table [2]. Les tuples compressés permettent de remplacer certaines valeurs par des ensembles ; un tuple compressé représente alors le produit cartésien des différents ensembles impliqués. Les supports courts appliqués aux contraintes table permettent de ne spécifier aucune valeur pour certaines variables au niveau d'un tuple ; une variable non spécifiée pouvant prendre n'importe quelle valeur de son domaine. Dans [5], nous proposons de généraliser ces deux approches en introduisant, au niveau des entrées de tables, des contraintes arithmétiques simples. Ces tuples sont appelés *smart tuples* et les contraintes table sur des smart tuples, des contraintes *smart table*. Par exemple, l'ensemble des tuples $\{(1,2,1), (1,3,1), (2,2,2), (2,3,2), (3,2,3), (3,3,3)\}$ pour les variables $\{x_1, x_2, x_3\}$ dont les domaines sont $\{1,2,3\}$ peut être représenté par le smart tuple suivant :

x_1	x_2	x_3
$= x_3$	≥ 2	$*$

ou, sous une forme équivalente par $(x_1 = x_3, x_2 \geq 2)$. Une étoile dans la forme tabulée signifie que la variable peut prendre n'importe quelle valeur de son domaine, si elle n'est pas contrainte ailleurs (ce qui n'est pas le cas dans l'exemple).

Il est à noter que les smart tuples peuvent être vus comme une disjonction de conjonctions de contraintes arithmétiques simples. Les combinaisons logiques de contraintes ont été largement étudiées dans la littérature. L'originalité de l'approche présentée dans ce papier est que la forme des smart tuples (disjonctions de conjonctions, conjonctions formant des réseaux acycliques) permet de définir un algorithme de filtrage efficace pour obtenir la cohérence d'arc généralisée (GAC). Les contraintes smart table peuvent être perçues comme un sous ensemble de l'algèbre logique définie dans [1]. Les règles de filtrage pour contraintes smart table sont d'ailleurs directement dérivées de celles présentées dans [1]. La restriction aux disjonctions de conjonctions est motivée par un désir de garder la sémantique des contraintes smart table proche de celle des contraintes table classiques. La restriction aux conjonctions acycliques est une nécessité pour pouvoir utiliser les règles de filtrage de [1] et garantir l'obtention de GAC de manière efficace.

1 Définition de la contrainte smart table

Une contrainte smart table est définie par un ensemble de smart tuples. Cet ensemble de smart tuples est appelé la smart table et représenté par $table(sc)$ pour une contrainte smart table sc. Un smart tuple σ est lui même un ensemble de contraintes de tuple. Une contrainte tuple peut prendre quatre formes différentes :

1. <var> <op> a
2. <var> $\in S$ ou <var> $\notin S$
3. <var> <op> <var>
4. <var> <op> <var> + b

où <var> est une variable de la portée de la contrainte smart table, a et b, des constantes, S, un ensemble de constantes et <op> un opérateur choisi dans $\{<, \leq, =, \neq, \geq, >\}$. La sémantique d'une contrainte smart table est simple et naturelle : un tuple classique τ est accepté par une contrainte smart table sc ssi il existe $\sigma \in table(sc)$ tel que τ satisfait σ. Une variable de la contrainte qui n'est pas contrainte par une contrainte de tuple peut prendre n'importe quelle valeur de son domaine dans le smart tuple.

Les contraintes smart table peuvent être utilisées pour modéliser efficacement de nombreuses contraintes, y compris certaines contraintes globales bien connues. Nous donnons ci-dessous un exemple de contrainte smart table encodant la contrainte Element. Les contraintes de tuple sont écrites directement dans la table de la contrainte dans la colonne correspondant à la première variable de la contrainte de tuple. Par exemple, la contrainte de tuple $x_1 \leq x_3 + 2$ sera représentée par $\leq x_3 + 2$ dans la colonne de x_1.

Element$(I, [x_1, x_2, \ldots, x_m], R)$ *:* $\bar{\boldsymbol{x}}[I] = R$

I	x_1	$\ldots$	x_m	R
$= 1$	$*$	$\ldots$	$*$	$= x_1$
$= 2$	$*$	$\ldots$	$*$	$= x_2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$= m$	$*$	$\ldots$	$*$	$= x_m$

2 Filtrage de la contrainte smart table

Cette section concerne le filtrage GAC de la contrainte smart table. La propriété GAC nécessite que chaque littéral (couple variable-valeur) ait un support sur chaque contrainte. Un support pour un littéral est un tuple valide et accepté par la contrainte prouvant que le littéral peut satisfaire la contrainte avec les domaines courants. L'identification de l'ensemble des supports d'une contrainte permet donc en général d'obtenir GAC pour cette contrainte. Pour une contrainte smart table sc, chaque smart tuple σ correspond à un CSP P_σ. Les variables de P_σ sont les variables de la portée de sc et les contraintes sont les contraintes de tuples de σ. Nous imposons que les graphes de contraintes liés aux CSPs de chaque smart tuple soient acycliques. Les tuples classiques supportés par un smart tuple σ sont ceux dans $sol(P_\sigma)$. L'ensemble des supports pour une contrainte smart table est donc $\bigcup_{\sigma \in table(sc)} sols(P_\sigma)$. Calculer l'ensemble des solutions pour chaque CSP P_σ peut sembler coûteux. Heureusement, la forme acyclique des CSPs de smart tuples permet d'obtenir efficacement l'ensemble des littéraux apparaissant dans $sols(P_\sigma)$. En effet, nous avons la propriété suivante :

Propriété 1 *Si σ est un tuple smart d'une contrainte smart table et que P'_σ est la fermeture GAC de P_σ, alors P'_σ est globalement consistant.*

Cette propriété nous permet d'obtenir l'ensemble des supports en calculant la fermeture GAC des smart tuples.

L'algorithme de filtrage pour les contraintes smart table, que nous appelons smartSTR, est basé sur STR et STR2 [7, 4]. Le principe de fonctionnement de STR / STR2 est de parcourir les tuples de la table de la contrainte. La validité de chaque tuple est testée. Si le tuple est valide, les supports sont collectés. Sinon, le tuple est retiré de la table. Pour SmartSTR, un smart tuple σ est valide si P_σ est satisfaisable et les supports collectés sont ceux de $sols(P_\sigma)$. L'astuce de la fermeture GAC de P_σ est utilisée dans les deux cas.

Références

[1] Fahiem Bacchus and Toby Walsh. Propagating logical combinations of constraints. In *IJCAI*, pages 35–40, 2005.

[2] Christopher Jefferson and Peter Nightingale. Extending simple tabular reduction with short supports. In *Proceedings of IJCAI'13*, pages 573–579, 2013.

[3] George Katsirelos and Toby Walsh. A compression algorithm for large arity extensional constraints. In *Proceedings of CP'07*, pages 379–393, 2007.

[4] Christophe Lecoutre. STR2 : optimized simple tabular reduction for table constraints. *Constraints*, 16(4) :341–371, 2011.

[5] Jean-Baptiste Mairy, Yves Deville, and Christophe Lecoutre. The smart table constraint. In *Integration of AI and OR Techniques in Constraint Programming*. Springer International Publishing, 2015.

[6] Jean-Charles Régin. Improving the expressiveness of table constraints. In *Proceedings of CP'11 Workshop on Constraint Modelling and Reformulation*, 2011.

[7] Julian R Ullmann. Partition search for non-binary constraint satisfaction. *Information Sciences*, 177(18) :3639–3678, 2007.

[8] Wei Xia and Roland H. C. Yap. Optimizing STR algorithms with tuple compression. In *Proceedings of CP'13*, pages 724–732, 2013.