Extension de Compact-Table aux Tables Simplement Intelligentes

Hélène Verhaeghe[1] *__Christophe Lecoutre__[2] **Yves Deville**[1] **Pierre Schaus**[1]

[1]UCLouvain, ICTEAM, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgique
[2]CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
[1]{*prenom.nom*}@*uclouvain.be* [2]*lecoutre@cril.fr*

Résumé

Ces dernières années, plusieurs algorithmes pour la contrainte table ont été proposés pour assurer la propriété de cohérence d'arc généralisée (GAC). Compact-Table (CT) [1] est un algorithme récent de l'état-de-l'art, que nous avons étendu dans notre article [2], intitulé "Extending Compact-Table to Basic Smart Tables" et publié à CP-17, aux tables simplement intelligentes (contenant des éléments tels que $= v$, $*$, $\neq v$, $\geq v$, $> v$, $< v$,$\leq v$, $\in S$ et $\notin S$, où v est une valeur et S un ensemble de valeurs). Nos expérimentations montrent une amélioration du temps de résolution lorsqu'il y a compression.

1 Introduction

La contrainte table, ou contrainte en extension, exprime explicitement pour les variables impliquées, soit les combinaisons de valeurs acceptées (*supports*), soit les combinaisons de valeurs rejetées (*conflits*). En théorie, toute contrainte peut se reformuler sous forme de contrainte table, c'est l'une des raisons pour lesquelles ce type de contrainte est très important en programmation par contraintes.

Beaucoup d'efforts ont été consentis au développement d'algorithmes de filtrage pour les contraintes tables, le plus efficace démontré étant l'algorithme Compact-Table [1], proposé en 2016. L'un des inconvénients avec les tables est que leur taille, et donc l'utilisation mémoire requise, peut potentiellement augmenter de manière exponentielle par rapport à l'arité. Pour pallier ce problème, plusieurs méthodes de compressions ont été proposées facilitant le processus de filtrage : les *arbre préfixe* (*tries*), les Diagrammes de Décision Multi-valeurs (*MDDs*) et les Automates Finis Déterministes (*DFAs*).

*Papier doctorant : Hélène Verhaeghe[1] est auteur principal.

D'autre approches, basées sur le concept de produit cartésien, ont également été envisagées : tuples compressés (*compressed tuples*), supports courts (*short support*), tables découpées (*sliced tables*), tables intelligentes (*smart tables*),...

Dans cet article, nous étendons l'algorithme Compact-Table, initialement introduit pour les tables positives (chaque tuple est une solution possible) sans aucune compression [1], afin de pouvoir gérer les tables simplement intelligentes. Ces tables sont des tables positives pouvant contenir dans une même table :

- l'assignation à une valeur unique $= v$ (initialement accepté par Compact-Table)
- la valeur universelle $*$ (déjà supporté dans une première version étendue CT* [3])
- la simple restriction $\neq v$
- les restrictions de bornes $\geq v$, $> v$, $< v$ et $\leq v$
- les restrictions d'ensembles $\in S$ et $\notin S$

2 Modification introduites

2.1 Gérer les $\neq v$

En partant de CT*, il est trivial d'ajouter la possibilité d'intégrer les $\neq v$. Lorsqu'il reste deux valeurs ou plus dans le domaine, il est trivial de voir qu'au moins une valeur rend le tuple valide (du point de vue de la variable considérée). Dans ce cas, le tuple doit donc rester valide comme dans le cas d'une $*$. Dans le cas limite où une seule valeur est présente dans le domaine, le bitset `support` est celui utilisé. En remplissant les `supports`* avec 0 pour le bit correspondant (comme pour les $*$) et les `supports` avec 1 pour tous les supports associés à une valeur autre que v, la propagation correcte est assurée.

2.2 Gérer les $\geq v$, $> v$, $< v$ et $\leq v$

Une variable x respecte la contrainte $\geq v$ (ou $> v$ qui est trivialement équivalent à $\geq v+1$ dans le cas de variables entières) si sa valeur maximale est plus grande ou égale à v (ou strictement plus grande que v). Pour aider à la vérification, nous avons donc ajouté de nouveaux bitsets, appelés `supportMax`, associés à chacune des valeurs du domaine. Le remplissage de ces bitsets se fait en supposant que la valeur associée est la valeur maximale du domaine : pour toutes les valeurs respectant $\geq v$ (ou $> v$), le bit correspondant est mis à 1, 0 dans l'autre cas. Les bits associées à un élément autre que $\geq v$ et $> v$ sont tous mis à 1. Lorsqu'une mise à jour de type *classique* est lancée et que le maximum à changé, une intersection entre `currTable` et le bitset correspondant au nouveau maximum est effectué, retirant ainsi les tuples contenant un $\geq v$ ou un $> v$ qui ne sont plus valide.

Gérer les $\leq v$ et $< v$ a été fait suivant le même principe avec introduction de bitsets appelés `supportMin`.

2.3 Gérer les $\in S$ et $\notin S$

Dû au nombre exponentiel d'ensembles différents pouvant se trouver dans une même table, la seule solution viable est l'application systématique d'une mise à jour de type *reset* pour les variables auxquelles une restriction d'ensemble est appliquée.

3 Algorithme de compression

Le problème de la compression optimale étant NP-complet, nous avons créé un algorithme de type glouton récursif introduisant des $\leq v$ et $\geq v$ (avec post-traitement pour ajouter des $\neq v$ et $*$). A chaque étape, l'algorithme prend le sous-ensemble des tuples contenant N éléments de compression et essaye d'en extraire un maximum de tuples avec $N+1$ éléments de compression. Nous avons pu observer la compression des différentes tables présentes dans nos différents bancs d'essai Fig.1 (ratio de compression en fonction du nombre de tuple initial) et en observer la diversité. Cela varie entre aucune compression (bancs d'essais Kakuro ou Nonogram) à une compression importante (PigeonsPlus).

4 Résultats

Les résultats obtenus (Fig.2) sur des instances variées ayant plus de 5% de compression nous montre que l'utilisation de CT^{bs}est compétitif. Lorsque la compression est moindre, il y peut y avoir un léger ralentissement dû aux opérations supplémentaires.

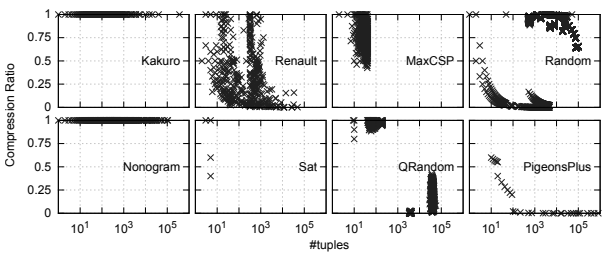

FIGURE 1 – Distribution du ratio de compression sur 8 séries d'instances

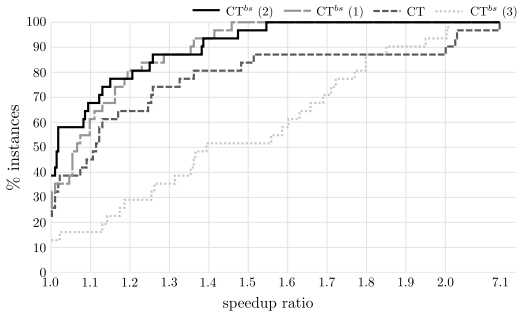

FIGURE 2 – Profile de performance comparant CT^{bs}et CT

5 Conclusion

Nous avons proposé une extension de Compact-Table (CT^{bs}) permettant de gérer les tables simplement intelligentes de manière efficace lorsqu'il y a une compression significative (à partir de 5% de tuples en moins). Nous avons également proposé un algorithme glouton pour la compression de table en table simplement intelligentes. Pour plus de détails concernant l'implémentation et les résultats, n'hésitez pas à lire l'article originel.

Références

[1] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-table : efficiently filtering table constraints with reversible sparse bit-sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[2] Hélène Verhaeghe, Christophe Lecoutre, Yves Deville, and Pierre Schaus. Extending compact-table to basic smart tables. In *Proceedings of CP'17*, pages 297–307, 2017.

[3] Hélène Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Extending compact-table to negative and short tables. In *Proceedings of AAAI'17*, 2017.