Relations entre MDDs et Tuples et Modifications dynamique de contraintes MDDs

Guillaume Perez **Jean-Charles Régin**

Université Nice-Sophia Antipolis, CNRS, I3S UMR 7271, 06900 Sophia Antipolis, France

guillaume.perez06@gmail.com jcregin@gmail.com

Résumé

Nous étudions les relations entre Multi-Valued Decision Diagrams (MDD) et les tuples (i.e. les éléments du produit cartésien du domaine des variables). D'abord nous améliorons les méthodes existantes pour transformer un ensemble de tuples, des Global Cut Seeds et des séquences de tuples en MDDs. Puis nous présentons plusieurs algorithmes sur place pour ajouter et supprimer des tuples d'un MDD. Ensuite nous considérons une contrainte MDD qui est modifiée durant la recherche en supprimant plusieurs tuples. Nous donnons un algorithme qui adapte MDD-4R à ces modifications dynamiques et persistantes. Plusieurs expérimentations montrent que les contraintes MDD sont compétitives avec les contraintes de table.

Abstract

We study the relations between Multi-valued Decision Diagrams (MDD) and tuples (i.e. elements of the Cartesian Product of variables). First, we improve the existing methods for transforming a set of tuples, Global Cut Seeds, sequences of tuples into MDDs. Then, we present some in-place algorithms for adding and deleting tuples from an MDD. Next, we consider an MDD constraint which is modified during the search by deleting some tuples. We give an algorithm which adapts MDD-4R to these dynamic and persistent modifications. Some experiments show that MDD constraints are competitive with Table constraints.

1 Introduction

Les contraintes de table sont fondamentales et implémentées dans tous les solveurs CP. Elles sont explicitement définies par un ensemble d'éléments du produit cartésien du domaine des variables, aussi appelés tuples, qui sont valides.

Dans cet article, nous détaillons uniquement la phase de suppression d'un ensemble de tuples d'un MDD, pour le reste de ces méthodes, le lecteur est invité à lire [16].

Cheng et Yap ont proposé de compresser l'ensemble des tuples d'une contrainte en utilisant un Multi-valued Decision Diagram (MDD) et ont défini `mddc` un des premiers algorithme établissant la cohérence d'arc pour ceux-ci [7, 6]. Récemment, nous avons présenté MDD-4R un nouvel algorithme qui améliore `mddc` [14]. MDD-4R procède comme GAC-4R et, contrairement à `mddc`, maintient le MDD durant la recherche d'une solution. Nous avons aussi introduit un algorithme efficace pour réduire un MDD et plusieurs algorithmes pour combiner les MDDs [15]. Grâce à ces nouveaux algorithmes, plusieurs expérimentations basées sur des problèmes réels montrent qu'une approche basée sur les MDD devient compétitive avec les méthodes ad-hoc comme l'algorithme de filtrage associé avec les contraintes regular ou knapsack. De ce fait, le remplacement des contraintes de table par un MDD semble être possible dans le futur. Dans ce papier, nous espérons faire un pas de plus dans cette direction.

Les contraintes de table sont très utiles pour modéliser et résoudre beaucoup de problèmes réels. Elles peuvent être spécifiées soit directement par l'utilisateur, soit en synthétisant d'autres contraintes ou sous-problèmes [13, 12]. Elles ont été modifiées pour travailler avec des tuples ou des séquences de tuples [8, 10, 17]. De plus, leur expressivité est forte.

Si nous voulons être compétitif avec les contraintes de table, nous avons besoin de représenter efficacement différents type d'ensembles de tuples compressés. Pour cela nous montrons comment les GCSs (Global Cut Seeds) et les séquences de tuples peuvent être représentés par un MDD. Notamment, nous allons montrer qu'un MDD peut être construit directement depuis un ensemble trié de tuples sans utiliser de structure intermédiaire coûteuse en espace

comme celle proposée par Cheng and Yap.

Ensuite, nous considérons l'ajout et la suppression d'un tuple d'un MDD. Nous verrons que cette opération peut être efficacement effectuée en utilisant une méthode qui consiste à isoler le chemin du MDD correspondant au tuple dans le cas de la suppression et a celui du préfixe commun dans le cas de l'ajout. Ces opérations rendent plus facile l'ajout/suppression d'un ensemble de tuples, qui sont les modifications que nous pouvons apporter à un MDD. D'un coté, cela renforce l'expressivité des MDDs. D'un autre coté, cela ouvre aussi la porte à des algorithmes dynamiques pour maintenir la cohérence d'arc des contraintes MDD. Aussi nous proposons un algorithme de cohérence d'arc pour ces contraintes quand plusieurs tuples sont définitivement supprimés durant la recherche. Cet algorithme est basé sur les opérations effectuées au préalable et doit être implémenté efficacement car la suppression d'un tuple peut créer de nouveaux noeuds dans le MDD et cela cause plusieurs problème avec la restaurations après le backtrack. En d'autres termes, une suppression persistante est une modification monotone par rapport à la consistance de la contrainte, mais la maintenance du MDD n'est pas monotone.

Être capable de maintenir la cohérence d'arc d'un MDD auquel plusieurs tuples sont supprimés définitivement possède deux gros avantages. D'abord, cela est utile pour travailler avec des problèmes comme l'enregistrement des nogoods. Actuellement, des algorithmes ad-hoc ou des tables dynamiques sont utilisés. Donc cela renforce notre idée de voir les contraintes MDD comme un possible remplacement des contraintes de table. Ensuite, les contraintes MDD sont maintenant compétitives avec les algorithmes ad-hoc pour la contrainte regular. Donc avoir un algorithme dynamique pour eux nous donne immédiatement un algorithme dynamique pour les contraintes regular.

Dans cet article, nous rappelons quelques définitions, nous détaillons uniquement la suppression sur place de tuples dans un MDD. Nous validons notre approche à l'aide d'une étude expérimentale et pour finir concluons. Les opérations d'ajout de tuple sur place sont détaillé dans le rapport [16].

2 Définitions

Un multi-valued decision diagram (MDD) est une méthode pour représenter une fonction discrète. C'est une extension multi-valeur des BDDs [5].

Un MDD, comme utilisé en CP, [1, 11, 12, 3, 9], est un Directed Acyclic Graph (DAG) avec une racine, utilisé pour représenter des fonctions $f : \{0...d-1\}^r \rightarrow \{true, false\}$, basées sur un entier donné d (Voir Figure .). Pour r variables données, la représentation par un DAG est faite pour contenir r niveaux, tel que chaque variable soit représentée par un niveau spécifique du graphe. Chaque

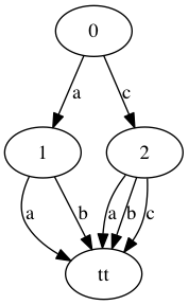

FIGURE 1 – Un MDD representant l'ensemble de tuples {{a,a},{a,b},{c,a},{c,b},{c,c}}

noeud d'un niveau spécifique possède au plus d arcs sortants vers les noeuds du niveau suivant du graphe. Chaque arc possède une étiquette correspondant à sa valeur. Le niveau final est représenté par le noeud terminal true (le noeud terminal false est souvent omis). Il y a une équivalence entre $f(v_1, ..., v_r) = true$ et l'existence d'un chemin du noeud racine au noeud terminal true et dont les arcs sont étiquetés par $v_1, ..., v_r$. Les noeuds sans aucun arc sortant ou arc entrant sont supprimés.

Dans une contrainte MDD, le MDD modélise l'ensemble des tuples qui satisfont la contrainte, tel que chaque chemin du noeud racine au noeud terminal true correspond à un tuple autorisé. Chaque variable du MDD correspond à une variable de la contrainte. Un arc associé à une variable du MDD correspond à une valeur de la variable correspondante de la contrainte.

Par commodité, nous allons noter d le nombre maximum de valeur dans le domaine d'une variable et un arc sortant de x vers y étiqueté par v sera noté (x, v, y). Nous noterons aussi par $\omega^+(n)$ l'ensemble des arcs sortants de n.

Un exemple de MDD est donné en Figure 1. Ce MDD représente les tuples {a,a}, {a,b}, {c,a}, {c,b} et {c,c}. Pour chaque tuple, il y a un chemin du noeud racine (0) vers le noeud terminal (tt) dont les arcs sont étiquetés par les valeurs du tuple.

La réduction d'un MDD est une des opérations les plus importante. Elle consiste à fusionner les noeuds équivalents, i.e. les noeuds possédant le même ensemble ω^+ en ne considérant que les deux dernières composantes (valeur et extrémité terminale). Généralement, un algorithme de réduction fusionne les noeuds jusqu'à ce qu'il n'y ait plus de noeuds équivalents. Le plus souvent, seuls les MDD réduits sont considérés car ils sont plus petits. Notons que l'opération de réduction ne peut pas augmenter le nombre d'arc.

3 Suppression de tuples d'un MDD

Plusieurs travaux ont été effectués pour appliquer des opérations sur les BDDs. Par exemple, Bryant a définit des algorithmes pour appliquer différents opérateurs [5, 4]. Ce-

pendant, les algorithmes décrits ne sont pas sur place (i.e. il y a la creation d'un BDD résultat) et ce n'est pas facile de généraliser ces algorithmes écrit pour des BDDs aux MDDs. Cela est en partie due aux règles booléennes qui ne sont plus vraies quand nous avons d valeurs dans le domaine et parce-que la complexité de certains d'entre eux est multiplié par $O(d)$ quand nous travaillons avec d valeurs. Plusieurs algorithmes ont été proposés pour appliquer des opérateurs aux MDDs [2, 15]. Cependant, ils ne sont pas sur place.

Dans cette section nous définissons un algorithme sur place pour la suppression de tuples d'un MDD. Ces algorithmes sont nécessaires pour être capable de définir des algorithmes dynamiques pour la contrainte MDD compétitifs avec ceux des contraintes de table.

Nous donnons d'abord un algorithme pour la suppression d'un seul tuple d'un MDD. Puis nous le généralisons pour un ensemble de tuples.

3.1 Suppression d'un tuple d'un MDD

La suppression d'un tuple d'un MDD est faite par une opération appelée "path isolation". L'idée de cette opération est de construire un chemin spécifique dont les arcs sont étiquetés par les valeurs du tuple qui doit être supprimé. De plus les arcs équivalents aux arcs du chemin isolé sont supprimés du MDD.

Cela se déroule en quatre étapes :

1. L'isolation du premier niveau
2. L'isolation de tous les niveaux intermédiaires (ni le premier, ni le dernier)
3. L'isolation du dernier niveau
4. L'appel d'un algorithme de réduction incrémentale sur le MDD

Détaillons ces étapes. Soit τ le tuple que nous voulons supprimer. soit $\tau[i]$ la valeur pour la variable $x[i]$.

Étape 1. D'abord nous identifions $a_1 = (r, n_1, \tau[1])$ l'arc du premier niveau étiqueté par $\tau[1]$ la première valeur du tuple. Nous créons le noeud ne_1, l'arc $(r, ne_1, \tau[1])$ et nous supprimons l'arc a_1. Nous affectons $mddNode$ à n_1 et $isolatedNode$ à ne_1.

Étape 2. Pour chaque niveau i de 2 à $r-1$ nous répétons l'opération suivante. Nous identifions $a_i = (mddNode, n_{i+1}, \tau[i])$ l'arc sortant partant de $mddNode$ étiqueté par $\tau[i]$. Nous créons le noeud ne_{i+1} et l'arc $(isolatedNode, ne_{i+1}, \tau[i])$. Pour chaque arc $(mddNode, y, w)$ tel que $w \neq \tau[i]$ nous créons l'arc $(isolatedNode, y, w)$. Nous affectons $mddNode$ à n_{i+1} et $isolatedNode$ à ne_{i+1}.

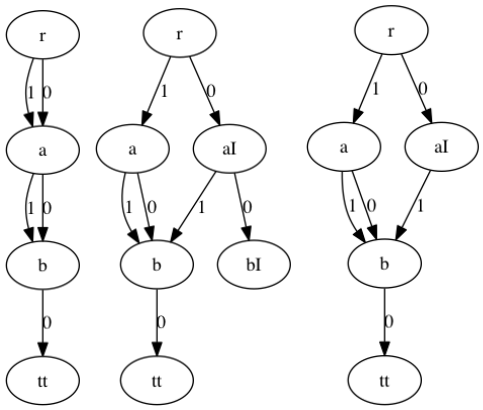

FIGURE 2 – le MDD de gauche est le MDD initial auquel nous retirons le tuple {0,0,0}. Le MDD du milieu MDD est le résultat de l'isolation du chemin correspondant au tuple. les noeuds aI et bI sont crées depuis les noeuds a et b dans le but de créer le chemin isolé. Le MDD de droite est le résultat de cette opération.

Étape 3. Pour chaque arc $(mddNode, tt, w)$ tel que $w \neq \tau[i]$ nous créons l'arc $(isolatedNode, tt, w)$.

Étape 4. Nous appliquons un algorithme de réduction en ne considérant que le chemin et les voisins du chemin.

Si à un moment nous ne pouvons plus identifier un arc, cela signifie que τ n'appartient pas au MDD. Figure 2 montre l'application de cet algorithme.

la complexité de cette suppression est bornée par $O(rd)$ car pour chaque noeud isolé, nous avons besoin de recréer ses arcs. cependant, en pratique c'est souvent proche de $O(r)$. Donc nous pouvons simplement gérer la suppression d'un ensemble de tuples en répétant cet algorithme. Nous proposons d'améliorer cette méthode.

3.1.1 Suppression d'un ensemble de tuples.

Nous donnons un algorithme sur place pour supprimer un ensemble de tuples d'un MDD. Dans ce cas, nous transformons l'ensemble de tuples en MDD puis nous soustrayons ce nouveau MDD du MDD initial. Cet algorithme généralise le précédant et suit les étapes suivantes. Il isole les noeuds ayants un chemin commun au deux MDD, ensuite il supprime les arcs communs des noeuds isolés au dernier niveau du second. L'algorithme 1 est une possible implémentation.

La Figure 3 montre la soustraction de l'ensemble de tuples { {1,0,1} ,{1,1,1} ,{1,2,1} ,{1,3,1} } au MDD représentant tout les tuples possibles pour les valeurs {0,1,2,3}. L'ensemble est isolé du MDD. Ensuite, la suppression de

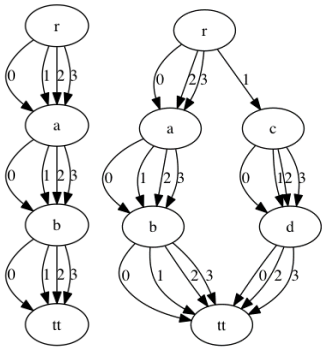

FIGURE 3 – Le MDD de gauche représente le MDD autorisant tous les tuples pour le domaine {0,1,2,3}. Le MDD de droite représente la suppression de l'ensemble de tuples { {1,0,1} ,{1,1,1} ,{1,2,1} ,{1,3,1} } au MDD de gauche.

l'arc étiqueté 1 du noeud d correspond à la suppression seule des tuples contenus dans le l'ensemble.

Il est difficile de borner la complexité de la suppression de T tuples, car le MDD créé peut compresser l'information.

3.2 Réduction incrémentale

Une réduction est requise après la suppression ou l'ajout de tuples. En utilisant un algorithme générique, cela est coûteux car tous les noeuds sont traversés pour pouvoir fusionner les noeuds équivalents. Si l'on considère que nous ajoutons/supprimons des tuples d'un MDD qui est déjà réduit, nous pouvons économiser des calculs pour l'application de la réduction appliquée après ces opérations. En effet seuls les noeuds isolés et ceux ayant un noeud voisin isolé ont besoin d'être reconsidéré pour tester leur équivalence car les autres ne sont pas modifiés. De plus, identifier les noeuds isolés est facile car ils appartiennent à la liste L de l'algorithme. L'avantage de cette approche est que l'étape de réduction n'augmente pas la complexité de l'opération de suppression.

4 Experimentations

Le but de ces experimentations est de montrer que lorsque nous appliquons des suppressions une approche basée sur les MDD est compétitive avec une approche basée sur les tables. Même si elle utilise des tuples compressés.

Machines MacBook Pro, Intel Core I7, 2,3GHz, 8GB memory.

Solveur or-tools 3158.

Algorithm 1: In-place Deletion Algorithm

```
DELETION(L, mdd_1, mdd_2)
  // Step 1 : first layer
  for each (root(mdd_1), v, y_1) ∈ ω+(root(mdd_1)) do
      if ∃(root(mdd_2), v, y_2) ∈ ω+(root(mdd_2)) then
          ADDARCANDNODE(L, 1, root(mdd_1), v, y_1, y_2)
          DELETEARC(root(mdd_1), v, y_1)
  // Step 2 : intermediate layers
  // L[i] is the set of nodes in layer i.
  for each i ∈ 1..r − 2 do
      L[i] ← ∅
      for each node x ∈ L[i − 1] do
          get x_1 and x_2 from x = (x_1, x_2)
          for each (x_1, v, y_1) ∈ ω+(x_1) do
              if ∃(x_2, v, y_2) ∈ ω+(x_2) then
                  ADDARCANDNODE(L, i, x, v, y_1, y_2)
              else CREATEARC(L, i, x, v, y_1)
  // Step 3 : last layer
  for each node x ∈ L[r − 1] do
      get x_1 and x_2 from x = (x_1, x_2)
      for each (x_1, v, tt) ∈ ω+(x_1) do
          if / ∃(x_2, v, y_2) ∈ ω+(x_2) then
              CREATEARC(L, r, x, v, tt)
  PREDUCE(L)
  return root

ADDARCANDNODE(L, i, x, y_1, v, y_2)
  if ∄y ∈ L[i] s.t. y = (y_1, y_2) then
      y ← CREATENODE(y_1, y_2)
      add y to L[i]
  CREATEARC(x, v, y)
```

Instances selectionnées Nous construisons des instances aléatoires pour avoir une vue globale des deux approches.

Impact des suppressions Nous étudions le nombre de modifications déclenchées par la suppression de tuples. L'ensemble de tuples contient 6 variables et 5 valeurs. La Figure 4 montre les résultats pour trois types d'ensemble de tuples, un ayant une petite densité (8%), une moyenne densité (15%) et une grande densité (25%). Nous observons une évolution linéaire pour les contraintes de table ce qui est normal. L'approche MDD a besoin de moins de modifications. Étonnement, les meilleurs résultats sont obtenus pour les faibles et hautes densités. Le pire des cas pour les MDDs semble pour une densité moyenne. Cela peut être expliqué par le fait que les modifications ont beaucoup à faire et qu'il y a beaucoup de noeuds. La haute densité est fortement compressée, alors que la faible densité n'a que

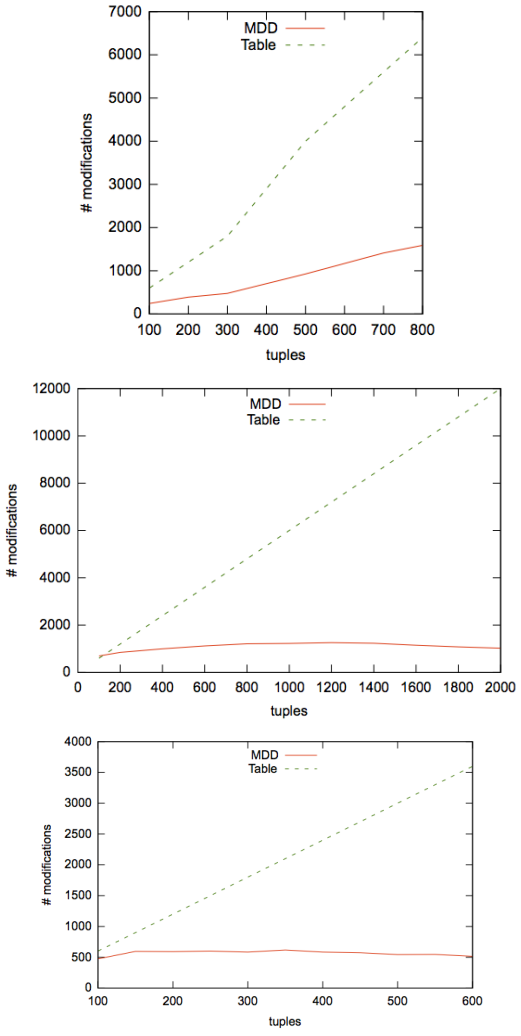

FIGURE 4 – Nombre de modifications pour la suppression de tuples. Graphe du haut : medium density. Graphe du bas : high density. Graphe du milieu : high density

Algorithme	#suppression	Domain size		
		10	12	14
GAC4R	0	1570	2123	2710
GAC4R	100 000	1274	1511	1763
MDD4RD	0	1115	1385	1815
MDD4RD	100 000	763	879	1064

TABLE 1 – temps (en ms) pour supprimer 100 000 tuples depuis un ensemble de tuples contenant 230 000 éléments durant la recherche de toutes les solutions.

peu de travail à effectuer.

Suppression de tuples durant la recherche Nous considérons un problème composé de contraintes d'arité 6. Chaque contrainte est définie depuis une table contenant 230 000 tuples. Nous cherchons toutes les solutions et appliquons la suppression de 100 000 tuples durant la recherche. Cela déclenche 600 000 modifications pour les contraintes de table, car nous avons besoin de vérifier si un tuple est un support pour chacune de ses valeurs. Il est intéressant de remarquer que le nombre de modifications (création/suppression d'arcs et de noeuds) déclenché pour le MDD est de 135 000. Ce nombre est donc plus petit, cela vient du fait que un MDD compresse les tuples. Par contre, plus d'opérations peuvent être requises quand un tuple est supprimé mais la structure de donnée reste compressés et reste donc plus performante. La Table 1 nous donne plusieurs informations sur le temps requis pour appliquer ces opérations. Une fois encore, l'approche MDD fonctionne bien.

5 Conclusion

Nous avons décrit des algorithmes sur place efficaces pour supprimer un ensemble de tuples d'un MDD. Nous avons aussi montré plusieurs experimentations. Ce travail contribue à la preuve que l'approche basée sur les MDD est compétitive avec une approche basée sur les tables pour représenter les contraintes en extension en programmation par contraintes.

Références

[1] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.

[2] D. Bergman, A. Cire, and W-J. van Hoeve. Mdd propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50 :697–722, 2014.

[3] David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, pages 20–35, 2011.

[4] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3) :293–318, 1992.

[5] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C35(8) :677–691, 1986.

[6] 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.

[7] Kenil C. K. Cheng and Roland H. C. Yap. Maintaining generalized arc consistency on ad hoc r-ary constraints. In *CP*, pages 509–523, 2008.

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

[9] G. Gange, P. Stuckey, and Radoslaw Szymanek. Mdd propagators with explanation. *Constraints*, 16 :407–429, 2011.

[10] 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.

[11] Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.

[12] Samid Hoda, Willem Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *CP*, pages 266–280, 2010.

[13] Olivier Lhomme. Practical reformulations with table constraints. In *ECAI*, pages 911–912, 2012.

[14] G. Perez and J-C. Régin. Improving GAC-4 for table and MDD constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 606–621, 2014.

[15] G. Perez and J-C. Régin. Efficient operations on mdds for building constraint programming models. In *International Joint Conference on Artificial Intelligence, IJCAI-15*, Argentina, 2015.

[16] G. Perez and J-C. Régin. Relations between mdds and tuples and dynamic modifications of mdds based constraints. *CoRR*, abs/1505.02552, 2015.

[17] J-C. Régin. Improving the expressiveness of table constraints. In *CP'11, proceedings workshop ModRef'11*, 2011.