Contraintes de Classement

Christian Bessiere, Emmanuel Hébrard, George Katsirelos, Zeynep Kiziltan, Toby Walsh

Contraintes de Classement

Christian Bessiere[1] **Emmanuel Hebrard**[2] **George Katsirelos**[3] **Zeynep Kiziltan**[4] **Toby Walsh**[5]

[1] LIRMM, CNRS, Université de Montpellier
[2] LAAS-CNRS, Université de Toulouse
[3] MIAT, INRA
[4] Université de Bologne
[5] Université de Nouvelle-Galles du Sud

bessiere@lirmm.fr hebrard@laas.fr george.katsirelos@toulouse.inra.fr zeynep.kiziltan@unibo.it tw@cse.unsw.edu.au

Résumé

La notion de classement se manifeste dans de nombreux domaines : la recherche d'information, la planification de tournois, la bibliométrie, ou encore l'analyse statistique. Nous proposons une contrainte pour modéliser ce concept, ainsi que deux décompositions et des algorithmes de filtrage efficaces. Les résultats expérimentaux sur la minimisation de la corrélation entre classements démontrent l'intérêt de l'approche choisie. Ce papier est un résumé d'un article présenté à IJCAI-16 [1].

1 Introduction

Supposons que nous voulions déterminer le classement de joueurs de tennis sur un ensemble de tournois. La méthode de Kemeny-Young est la seule méthode à la fois *neutre*, *cohérente* et *Condorcet*. Elle calcule le classement qui minimise la somme des distances de *Kendall tau*, i.e., du nombre de divergences, pour chaque paire de joueurs, avec le classement d'un tournoi. Déterminer ce classement est NP-complet pour 4 tournois, et le calculer par résolution d'un CSP nécessite de représenter la notion de classement.

Un classement peut avoir des ex æquo, au contraire d'une permutation. Par exemple, 12225 et 12345 sont des classements mais seul le deuxième est une permutation. Il existe une contrainte globale élégante et efficace pour la notion de permutation, pour laquelle Régin a proposé un algorithme établissant la cohérence d'arc en $O(n^2d^2)$ [5]. Pourtant, aucun outil de résolution ne propose d'algorithme pour la contrainte de classement. Nous comblons ce manque dans ce papier.

2 La contrainte de classement

Définition 1 *Une séquence R est un* classement *si et seulement si $R = (1)$, ou $R = (x_1, \ldots, x_{m+1})$ avec $x_{m+1} = x_m$ ou $x_{m+1} = m + 1$ et $(x_1, \ldots, x_m)$ est un classement. La contrainte* Classement$(X_1, \ldots, X_n)$ *est satisfaite si et seulement si il existe une permutation π telle que $(X_{\pi(1)}, \ldots, X_{\pi(n)})$ est un classement.*

Par exemple, Classement$([4, 1, 2, 2])$ est satisfaite, mais Classement$([3, 1, 4, 3])$ ne l'est pas.

Il est possible d'exprimer cette contrainte à l'aide de la contrainte Trié$(\mathbf{X}, \mathbf{Y})$ [3], qui impose que $\mathbf{Y} = Y_1, \ldots, Y_n$ soit une permutation croissante de la séquence $\mathbf{X} = X_1, \ldots, X_n$:

$$\begin{aligned} &\textsc{Trié}(\mathbf{X}, \mathbf{Y}), \quad Y_1 = 1 \\ &\quad Y_i = Y_{i-1} \vee Y_i = i \quad \forall i \in [2, n] \end{aligned}$$

Un second modèle utilise la contrainte Cgc$(\mathbf{X}, V, \mathbf{Y})$ [4], où $\mathbf{Y} = \{Y_v \mid v \in V\}$, qui impose à chaque valeur $v \in V$ un nombre d'occurrences dans $\mathbf{X}$ égal à Y_v.

$$\begin{aligned} \textsc{Cgc}(\mathbf{X}, \{1, \ldots, n\}, \mathbf{Y}), \quad Y_1 &= Z_1 \\ Z_i &= Z_{i-1} + Y_i \quad \forall i \in [2, n] \\ Z_i &\geq i \quad \forall i \in [1, n] \\ Y_i = 0 \iff Z_{i-1} &\geq i \quad \forall i \in [2, n] \end{aligned}$$

Établir la cohérence d'arc sur l'une ou l'autre de ces décompositions n'est pas suffisant pour établir la cohérence d'arc (de bornes) sur la contrainte Classement.

Arc cohérence. Le problème de l'existence d'un support peut être reformulé comme un problème de flot maximum lexicographique [2] (e.g., avec des coûts exponentiels). Un flot dans ce réseau est une séquence R qui respecte une condition proche de celle d'un classement : l'égalité $x_{m+1} = m + 1$ dans la définition 1 est remplacée par $x_{m+1} \geq m+1$. Si le flot maximum n'est

pas un classement, il existe un indice i dans la séquence tel que le préfixe de taille $i-1$ (i.e., $x_1, \ldots, x_{i-1}$, les $i-1$ plus petites valeurs) est un classement, et $x_i > i$. Or, la séquence $R = x_1, \ldots, x_n$ donnée par un flot maximum est maximale dans l'ordre lexicographique. Il s'en suit qu'il n'existe pas de classement préfixe dont le maximum est supérieur ou égal à x_i et de longueur $\leq i$. Cette contrainte peut être rajoutée au réseau en changeant la capacité d'un seul arc. Le nombre de révisions, et donc d'itérations est borné par $O(n^2)$.

Cohérence de bornes. L'algorithme glouton suivant calcule un support pour la cohérence de bornes en temps $O(n \log n)$. Il maintient une partition des variables entre assignées (A) et non-assignées (U) :

SupportCB : A chaque itération, si aucune variable dans U ne contient la valeur $|A|+1$, alors un échec est renvoyé. Sinon, la variable X_i avec la plus petite borne supérieure est choisie parmi celles-ci. Lors d'une itération, les trois étapes suivantes ont lieu :

- X_i est affectée à $|A|+1$ et déplacée de U vers A.
- Pour un ensemble F initialement vide, et tant qu'il existe des variables dans U dont la borne supérieure est inférieure à $|A|+|F|+1$ ces variables sont déplacées de U vers F.
- Toutes les variables dans F prennent la valeur $|A|$ et sont déplacées de F vers A. Un échec est renvoyé si ce n'est pas possible.

Théorème 1 *L'algorithme SupportCB retourne un support de bornes s'il en existe un, et renvoie un échec sinon, en temps* $O(n \log n)$.

Il existe trois raisons pour l'incohérence de bornes :

Définition 2 (Valeur saturée/échec) *Une valeur v est saturée s'il existe exactement v variables dont le domaine a une intersection non nulle avec l'intervalle* $[1, v]$. *De plus, si ce nombre est strictement inférieur à v, alors v est une valeur échec.*

S'il existe une valeur saturée v, alors le domaine de toute variable contenant v peut être restreint à son intersection avec $[1, v]$. S'il existe une valeur échec, alors la contrainte est incohérente.

Définition 3 ((super-)intervalles de Hall) *Soit* $V_{\sqsubseteq}(a,b) = \{X \mid D(X) \subseteq [a,b]\}$ *et* $S(a,b) = |V_{\sqsubseteq}(a,b)|$. $[a,b]$ *est un intervalle de Hall si* $S(a,b) = b-a+1$, *et un super-intervalle de Hall si* $S(a,b) > b-a+1$.

S'il existe un super-intervalle de Hall $[a,b]$, alors aucune variable hors de $V_{\sqsubseteq}(a,b)$ ne peut prendre de valeur dans l'intervalle $[b+1, a+S(a,b)-1]$.

Enfin, une affectation peut être incohérente parce qu'elle étendrait un ensemble de (super-)intervalles jusqu'à vider le domaine d'une autre variable. Par exemple, une variable $X_j \notin V_{\sqsubseteq}(a,b)$ participerait à cet intervalle si une valeur dans $[a,b]$ lui été affectée, avec pour conséquence l'extension des valeurs interdites $[b+1, a+S(a,b)-1]$, jusqu'à possiblement recouvrir le domaine d'une variable X_j.

3 Résultats expérimentaux

Nous avons évalué les différents algorithmes et décompositions sur le problème de la minimisation de la corrélation entre deux classements. La figure 1 montre, pour chaque taille de séquence dans $[5, 20]$, le temps CPU moyen et le ratio d'instances non résolues en moins de 30 minutes (en pointillé) sur des domaines générés de façon aléatoire et pour quatre méthodes : les deux décompositions de la section 2 (Trié, Cgc) ; un algorithme complet pour la cohérence de bornes "force brute" (singleton) ; et un algorithme incomplet basé sur les règles de filtrage de la section 2 (filtrage).

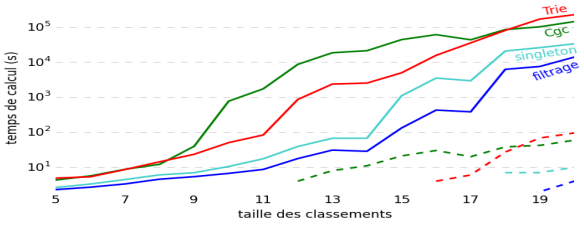

FIGURE 1 – Non-corrélation : Intervalles aléatoires

4 Conclusion

Nous avons proposé une contrainte globale de classement. Établir la cohérence d'arc pour cette contrainte est polynomial, mais coûteux. Nous avons donc proposé un algorithme pour la cohérence de bornes permettant un gain en performance d'environ un ordre de grandeur par rapport à une décomposition.

Références

[1] C. Bessiere, E. Hebrard, G. Katsirelos, Z. Kiziltan, T. Walsh. Ranking constraints. *IJCAI*, p. 705–711, 2016.

[2] D. Kozen. Lexicographic flow. Technical Report, Cornell University, 2009.

[3] W.J. Older, G.M. Swinkels, M.H. van Emden. Getting to the real problem : experience with BNR Prolog in OR. *PAP*, p. 465-478, 1995.

[4] A. Oplobedu, J. Marcovitch, Y. Tourbier. CHARME : Un langage industriel de programmation par contraintes, illustré par une application chez Renault. *Workshop on Expert Systems and their Applications*, p. 55-70, 1989.

[5] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *AAAI*, p. 362-367, 1994.