Algorithme Efficace pour la Fouille de Séquences Fréquentes avec la Programmation par Contraintes

John O.R. Aoga[1,2] * **Tias Guns**[3] **Pierre Schaus**[1]

[1] ICTEAM, UCLouvain, Belgique; [2] ED-SDI, UAC, Bénin
[3] VUB et KULeuven, Belgique
{john.aoga,pierre.schaus}@uclouvain.be tias.guns@vub.ac.be

Résumé

La programmation par contraintes (CP) a séduit la communauté de fouille de données grâce à son approche hautement déclarative et sa flexibilité. Cependant, ces avantages n'offrent pas une garantie d'efficacité. En témoigne la littérature où les méthodes basées sur la CP n'ont toujours pas réussi à être aussi performante que les méthodes spécialisées. Dans ce papier publié au *ECML-PKDD'16* [1], nous montrons comment en combinant les techniques des deux mondes on peut arriver à une méthode très robuste et efficace en plus d'être modulaire et flexible pour résoudre les problèmes de fouille de séquences fréquentes. Notre approche, intitulée *PPIC*, est une contrainte globale, conçue sur les méthodes de projection de base de données. Cette contrainte utilise une structure de données *auto-backtraquante* (*trailing*) pour stocker et restaurer efficacement les bases de données projetées. Le calcul des supports et le filtrage reposent sur des améliorations algorithmiques utilisant des données pré-calculées. Des expériences détaillées montrent comment cette approche, pour la première fois, surpasse à la fois les méthodes basées sur la CP et celles spécialisées. Ainsi, le moindre qu'on puisse dire est que le tandem fouille de données et CP a encore de beaux jours devant lui.

Problème. Etant donnée $I = \{1, \ldots, N\}$ un ensemble de N symboles, une séquence est une liste ordonnée d'éléments de I. Une séquence $\alpha = \langle \alpha_1 \alpha_2 \ldots \alpha_m \rangle$ est une sous-sequence d'une autre séquence $s = \langle s_1 s_2 \ldots s_n \rangle$ (ou inversement s est une super-séquence de α), noté $\alpha \preceq s$, ssi *(i)* $m \leq n$ et *(ii)* il existe une liste d'entiers $(j_1, \ldots, j_m)$, telle que $1 \leq j_1 \ldots \leq j_m \leq n$ avec $s_{j_i} = \alpha_i$. Une base de données de séquences est un ensemble de séquences : $SDB = \{(sid_i, s) \forall i \in [1, |SDB|]\}$ où sid_i est l'identifiant de la i^{eme} séquence s. La Table 1a est un exemple de SDB. Le nombre de séquences de SDB qui sont super-séquences de la séquence α est appelé *support*. Le problème de la fouille de séquence fréquente (SPM) est la recherche de toutes les séquences α qui ont un support supérieur à un seuil θ donné.

Ce problème est très étudié et est à l'origine de beaucoup de méthodes spécialisées dont les plus efficaces sont *PrefixSpan* [5] et *cSPADE* [6]. La méthode *PrefixSpan* doit son efficacité aux bases de données projetées (BDP). En effet, une BDP, notée $SDB_{|\alpha}$, est une partition de la SDB originale par rapport à α. C'est l'ensemble des suffixes des séquences dans lesquelles on a pu trouver une première correspondance de α. Considérons par exemple $\alpha = \langle A \rangle$. Cette sous-séquence se retrouve dans les séquences 1,2 et 3 avec les suffixes respectifs tels que présentés dans la Table 1b-suffixes. Cette BDP peut être stockée efficacement en ne conservant qu'un pointeur sur les positions dans chaque séquence (Table 1b-pos). Remarquez que le support de α est égale à la taille de sa BDP. Quand l'on étend α avec le symbole B, la nouvelle BDP se définit incrémentalement : $SDB_{|\langle AB \rangle} = (SDB_{|\langle A \rangle})_{|\langle B \rangle}$. Ainsi, en faisant grandir progressivement une sous-séquence, en commençant par la séquence vide, on construit les BDP on en déduit les supports et on peut vérifier lesquelles sont fréquentes.

a) SDB		b) $SDB_{\|\langle A \rangle}$		c) $SDB_{\|\langle AB \rangle}$		d) $SDB_{\|\langle ABC \rangle}$	
sid	sequence	*pos*	suffixes	*pos*	suffixes	*pos*	suffixes
sid_1	$\langle ABCBC \rangle$	2	$\langle \_BCBC \rangle$	3	$\langle \_\_CBC \rangle$	1	$\langle \_\_\_BC \rangle$
sid_2	$\langle BABC \rangle$	3	$\langle \_\_BC \rangle$	4	$\langle \_\_\_C \rangle$	1	$\langle \_\_\_\_ \rangle$
sid_3	$\langle AB \rangle$	2	$\langle \_B \rangle$	3	$\langle \_\_ \rangle$		
sid_4	$\langle BCD \rangle$						

TABLE 1 – Exemples de SDB et de BDP. (« _ » pour un symbole supprimé ; *pos* : les positions stockées).

*Papier doctorant : John O.R. Aoga[1,2] est auteur principal.

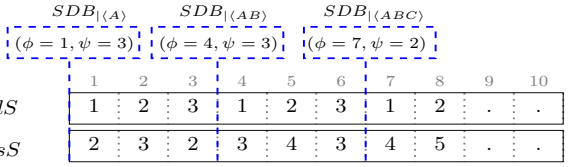

FIGURE 1 – Exemple de structure de données *auto-backtraquante* utilisant les BDP de la Table 1(-pos) .

En pratique, les solutions fréquentes générées sont souvent en grand nombre et ne sont pas forcément pertinentes. Afin d'affiner ou de superviser la fouille, d'autres contraintes sont souvent introduites. C'est là que les méthodes spécialisées (même les plus performantes) atteignent leur limite car manquant de flexibilité et de modularité. C'est ainsi, que les approches en CP ont été introduites [3, 2].

Objectif et contribution. Cependant, aucune de ces approches CP n'est aussi performante que les méthodes spécialisées. Ainsi, l'objectif de ce travail est de concevoir de nouvelles contraintes améliorant la littérature. La principale contribution de ce travail est la conception d'une *structure de données auto-backtraquante*.

Structure de données auto-backtraquante Le cœur de la méthode *PrefixSpan*, comme nous l'avons précédemment décrit, est la construction des BDP à chaque nœud de la recherche en profondeur. En effet, il faut remarquer que la BDP de α est utile pour construire l'extension de α à un symbole a donnée et en bactrackant la même BDP de α servira pour une nouvelle extension. Etant donnée qu'il est totalement inefficace de reconstruire les BDP à chaque *backtrack*, nous avons proposer de stocker et de restaurer les BDP en utilisant les techniques de *trailing* comme illustré à la Figure 1 (pour les valeurs voir la Table 1-pos) . On utilise deux vecteurs : $sidS$ et $posS$, pour maintenir respectivement les identifiants de séquences et la position dans chacune d'elles. Ces vecteurs sont des vecteurs « réversibles » : quand on étend une séquence, la BDP courante est lue. Une nouvelle est ensuite construite et stockée à sa suite en maintenant deux variables réversibles ϕ et ψ représentant la position courante dans les vecteurs et la taille de la BDP. Lors du *backtracking* les valeurs précédentes de ϕ et ψ sont restaurées et toutes celles après $\phi + \psi$ sont éventuellement écrasées. Cette structure permet non seulement d'optimiser l'utilisation de la mémoire mais rend plus rapide l'ensemble du processus de fouille. Plus encore, cette structure de données peut être utilisée dans n'importe quel problème utilisant la recherche en profondeur.

En utilisant cette structure de données et en la combinant avec des améliorations algorithmiques nous

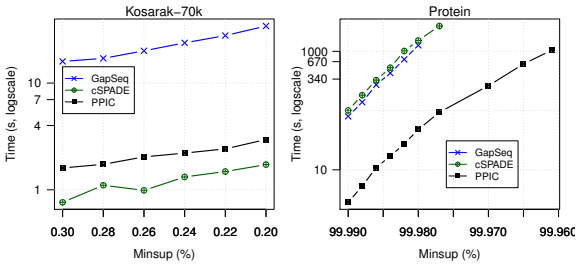

FIGURE 2 – Temps processeur pour PPIC avec *Gap-Seq* et *cSPADE* pour plusieurs θ (*minsup*).

avons proposés trois nouvelles *contraintes globales* dont la plus performante est *PPIC* (Prefix Projection Incremental Counting). De plus, ces contraintes peuvent être associées avec plusieurs contraintes additionnelles comme les contraintes sur la longueur des séquences, l'inclusion ou l'exclusion de certains symboles et les contraintes d'expression régulière.

Expériences et Résultats Comme illustré à la Figure 2, en utilisant deux bases de données (nombre de séquences/nombre de symboles) : l'une très éparse *Kosarak-70k* (69999/21144) et l'autre très dense *Protein* (103120/25), nous sommes clairement plus performant que *GapSeq* [2] sur tous les tableaux. Nous sommes aussi très compétitifs ou meilleurs que *cSPADE* [6]. C'est la toute première fois que les performances d'une approche CP dépassent à la fois celles des autres approches CP et spécialisées.

Nos approches sont implémentées en Scala dans le solveur OscaR [4]. Pour plus détail, le lecteur peut se référer au papier original [1] et/ou à notre site [1] où sont disponibles les données, le code et l'application.

Références

[1] J.O.R. Aoga, T. Guns, and P. Schaus. An efficient algorithm for mining frequent sequence with constraint programming. In *ECML PKDD 2016, Proceedings, Part II*. Springer International Publishing, 2016.

[2] A. Kemmar, S. Loudni, Y. Lebbah, P. Boizumault, and T. Charnois. A global constraint for mining sequential patterns with GAP constraint. In *CPAIOR 2016, Proceedings*, pages 198–215. Springer, 2016.

[3] B. Negrevergne and T. Guns. Constraint-based sequence mining using constraint programming. In *CPAIOR15, Proceedings*. Springer, 2015.

[4] OscaR Team. OscaR : Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and MC. Hsu. Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICCCN*, page 0215. IEEE, 2001.

[6] M.J. Zaki. Sequence mining in categorical domains : incorporating constraints. In *ICIKM*, pages 422–429. ACM, 2000.

1. `http://sites.uclouvain.be/cp4dm/spm/`