Échantillonnage d'ensemble de motifs diversifiés par compression locale

François Camelin*, Samir Loudni*, Gilles Pesant**, Charlotte Truchet***

*TASC – DAPI, IMT-Atlantique, LS2N – CNRS, Nantes, France
{francois.camelin, samir.loudni}@imt-atlantique.fr
**Polytechnique Montréal, Montreal, Canada
gilles.pesant@polymtl.ca
*** LS2N, 2 Chemin de la Houssinière, Nantes 44322, France
charlotte.truchet@univ-nantes.fr

Résumé. Les méthodes exhaustives d'extraction de motifs dans une base de données constituent de véritables obstacles pour la rapidité et le contrôle en sortie des motifs: un grand nombre de motifs sont extraits, dont beaucoup sont redondants. Les méthodes d'extraction de motifs par échantillonnage, qui permettent de contrôler la taille des sorties obtenues tout en garantissant des temps de réponse rapides, apportent une solution à ces deux problèmes. Toutefois, ces méthodes ne fournissent pas des motifs de bonne qualité : elles retournent des motifs très peu fréquents dans la base. Par ailleurs, elles ne passent pas à l'échelle. Pour garantir en sortie des motifs plus fréquents et diversifiés, nous proposons d'intégrer à l'échantillonnage des méthodes de compression pour sélectionner les motifs les plus représentatifs des transactions échantillonnées. Nous montrons que notre approche améliore l'état de l'art en termes de diversité des motifs produits.

1 Introduction

La fouille de motifs sous contraintes est une tâche fondamentale en fouille de données, qui extrait des modèles (ou motifs) localement intéressants pour être soit interprétés directement par des experts du domaine, soit utilisés comme descripteurs dans des tâches en aval, telles que la classification ou le clustering. Depuis la publication de l'article fondateur (Agrawal et Srikant, 1994), deux problèmes ont limité l'utilisation de cette approche : 1) comment produire rapidement des motifs variés, et 2) comment gérer les grands ensembles de résultats qui se comptent souvent en milliers, voire en millions de modèles. Remplacer le cadre original de support-confiance par d'autres mesures de qualité (Tan et al., 2002) ne résout pas l'explosion des motifs. Le post-traitement des résultats via les représentations condensées laisse encore généralement de nombreux motifs, dont beaucoup sont redondants.

Pour faire face à ces difficultés, des méthodes d'échantillonnage en sortie de motifs ont été récemment proposées (Boley et al., 2011; Dzyuba et al., 2017; Hasan et Zaki, 2009; Bendimerad et al., 2020; Diop et al., 2020). Ces méthodes permettent l'extraction rapide d'échantillons

de motifs représentatifs de l'ensemble des motifs du jeu de données proportionnellement à une mesure de qualité. Les méthodes d'échantillonnage en plusieurs étapes (Boley et al., 2011, 2012) forment la classe la plus rapide parmi les classes de méthodes d'échantillonnage en sortie de motifs. La méthode CFTP proposée par Boley et al. (2012) permet d'échantillonner des motifs en une procédure à deux étapes : la première étape tire une transaction du jeu de données proportionnellement au nombre de motifs qu'elle contient et la seconde étape tire un motif de cette transaction proportionnellement à son intérêt. Néanmoins, cette méthode ne fournit pas des motifs de bonne qualité, c'est-à-dire qu'elle retourne souvent des motifs longs et très peu fréquents, phénomène dit de la *longue traîne* (Diop et al., 2020). Par ailleurs, la plupart des méthodes de cette classe, et plus particulièrement CFTP, assurent une forme de diversité entre motifs échantillonnés par le biais d'une étape d'aléatoire souvent très coûteuse, ce qui limite très fortement le passage à l'échelle sur des bases de plus grande taille. Dans cet article, nous proposons d'exploiter le principe de la compression au sein de la méthode CFTP pour contourner le phénomène de la longue traîne, dans le but d'échantillonner des motifs diversifiés et de bonne qualité tout en garantissant un passage à l'échelle. L'idée de la compression consiste à trouver un ensemble de motifs permettant de décrire le mieux le jeu de données avec une taille de description réduite (principe du MDL (Vreeken et al., 2011)). En effet, les motifs extraits présentent alors peu (ou pas) de redondances étant donné qu'ils décrivent différentes parties du jeu de données. Leur recouvrement est donc faible. Nous montrons l'intérêt de notre approche sur les jeux de données de l'UCI, de FIMI et de CP4IM.

Cet article est organisé comme suit. La section 2 rappelle les préliminaires. La section 3 présente notre contribution d'échantillonnage par compression. La section 4 présente les résultats de nos expérimentations sur les jeux de données de l'UCI, de FIMI et de CP4IM. Enfin, nous concluons et donnons quelques perspectives de recherche.

2 Préliminaires

2.1 Motifs ensemblistes et échantillonnage

Soit $\mathcal{I}$ un ensemble de n *items*, un motif X est un sous-ensemble non vide de $\mathcal{I}$. Le langage des motifs correspond à $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \{\emptyset\}$. Un jeu de données transactionnel $\mathcal{D}$ est un ensemble de transactions, où chaque *transaction* $t \subseteq \mathcal{I}$; $\mathcal{T} = \{1, ..., m\}$ est un ensemble de m indices de *transaction*. Un motif X *apparaît* dans une transaction t, ssi $X \subseteq t$. Nous notons $\mathcal{L}(t)$ l'ensemble des motifs qui apparaissent dans t. La *couverture* de X dans $\mathcal{D}$ est l'ensemble des transactions dans lesquelles il apparaît : $\boldsymbol{t}(X) = \{t \in \mathcal{D} \mid X \subseteq t\}$. La fréquence de X dans $\mathcal{D}$ est la taille de sa couverture : $freq(\mathcal{D}, X) = |\boldsymbol{t}(X)|$. L'aire d'un motif est définie par $aire(\mathcal{D}, X) = freq(\mathcal{D}, X) \times |X|$.

Un motif X est dit *fréquent* dans $\mathcal{D}$ si $freq(\mathcal{D}, X) \geq \theta$, où θ est un seuil minimal fixé par l'utilisateur. Étant donné $T \subseteq \mathcal{D}$, $\boldsymbol{i}(T)$ est un ensemble d'items qui sont communs à toutes les transactions de T : $\boldsymbol{i}(T) = \{i \in \mathcal{I} \mid \forall t \in T, i \in t\}$. On définit par $clos$ un *opérateur de fermeture*, tel que $clos(X) = \boldsymbol{i} \circ \boldsymbol{t}(X) = \boldsymbol{i}(\boldsymbol{t}(X))$. La *fermeture* d'un motif X est l'ensemble des items qui sont contenus dans toutes les transactions de $\boldsymbol{t}(X)$: $clos(\mathcal{D}, X) = \{i \in \mathcal{I} \mid \forall t \in \boldsymbol{t}(X), i \in t\}$. Un motif X est dit *clos* ssi $clos(\mathcal{D}, X) = X$. La *diversité* d'une paire de motifs $\langle X, Y \rangle$ est reflétée par le nombre de transactions couvertes à la fois par X et Y. Pour mesurer la diversité, nous utiliserons l'indice de Jaccard. Il permet de mesurer la proportion de

chevauchement de couvertures entre deux motifs :

$$Jac(X,Y) = \frac{|\boldsymbol{t}(X) \cap \boldsymbol{t}(Y)|}{|\boldsymbol{t}(X) \cup \boldsymbol{t}(Y)|}$$

Cet indice varie entre 0 et 1, une valeur proche de 0 indiquant une grande diversité. Une mesure d'intérêt vise à sélectionner et à classer les motifs en fonction de leur intérêt potentiel selon un certain point de vue. Plus formellement : Étant donné un jeu de données $\mathcal{D}$, une mesure m est une fonction qui associe une valeur à un motif :$m : \mathcal{L}_\mathcal{I} \times \mapsto \mathbb{R}$. La fréquence est l'une des mesures de base utilisée en fouille de motifs. Il existe aussi d'autres mesures qui mettent l'accent sur la particularité (Zhong et al., 2003), la diversité (Hilderman et Hamilton, 2001), la surprise (Liu et al., 1999) ou encore l'utilité (Yao et al., 2004) des motifs.

L'échantillonnage de motifs est une technique de fouille de motifs permettant d'accéder aux motifs du langage $\mathcal{L}_\mathcal{I}$ par une procédure simulant une distribution $\pi : \mathcal{L}_\mathcal{I} \Rightarrow [0,1]$. Cette procédure est définie par rapport à une mesure d'intérêt m : $\pi(.) = \frac{m(.)}{Z}$, où Z est une constante de normalisation définie par : $Z = \sum_{X \in \mathcal{L}_\mathcal{I}} m(X)$. Formellement, l'échantillonnage de motifs du langage $\mathcal{L}_\mathcal{I}$ selon une distribution proportionnelle à la mesure d'intérêt m dans un jeu de données $\mathcal{D}$ peut être formulé par l'opérateur suivant (Diop, 2020) :

$$Echantillonner_k(\mathcal{L}_\mathcal{I}, \mathcal{D}, m) = \bigcup_{i=1}^{k} X_i \sim m(\mathcal{L}_\mathcal{I})$$

où k représente le nombre d'échantillons de motifs à extraire et $X \sim m(\mathcal{L}_\mathcal{I})$ signifie que le motif X est tiré avec une probabilité proportionnelle à la mesure d'intérêt m :
$X \sim m(\mathcal{L}_\mathcal{I}) \equiv \pi(X) = \frac{m(X)}{Z}$.

D'autres approches utilisent plusieurs étapes pour d'abord échantillonner une sous-partie de la base de données, puis depuis cet échantillon tirer un motif par échantillonnage direct. Nous présentons ci-dessous la méthode CFTP qui permet d'échantillonner des motifs au moyen d'une procédure en deux étapes.

2.2 Coupling From The Past (CFTP)

L'échantillonnage en deux étapes introduit par Boley et al. (2012) consiste en des tirages à deux étapes : la première étape consiste à tirer une transaction t du jeu de données proportionnellement à son poids, la seconde étape consiste à tirer un motif de la transaction proportionnellement à son intérêt dans la transaction échantillonnée.

Première étape de CFTP. Pour réaliser cette étape, il est nécessaire de calculer le poids de chaque transaction t. En supposant que chaque motif a la même utilité et que celle-ci vaut 1, alors le poids w d'un ensemble d'item I est égal au nombre de motifs de taille supérieure à 1 qu'il est possible de générer à partir de I : $w(I) = |\mathcal{L}_I| - |I| = 2^{|I|} - |I| - 1$. En remplaçant I par t cela donne le poids d'une transaction. Pour éviter de favoriser le tirage de transactions ayant le plus d'items, Boley et al. (2012) introduisent la notion de **facteur de fréquence**. Le facteur de fréquence détermine le nombre de transaction c dans un échantillon pour la première étape. Cette nouvelle approche permet de sélectionner à l'étape 1 un sous-ensemble de c transactions du jeu de données en simulant une CMMC où chaque état de la

Algorithm 1 Première étape de l'algorithme de CFTP

Entrée: une base de données $\mathcal{D}$, une distribution d, un facteur de fréquence c
Sortie: Un échantillon de c transactions

1: $i \leftarrow 1$, $E_{Actu} \leftarrow$ NULL ▷ L'état initial n'est pas défini
2: $listeEtats \leftarrow$ CREERLISTEETATS($\mathcal{D}, c, 2^i$) ▷ créer une liste de 2^i états
3: **tant que** (E_{Actu} == NULL) **faire**
4: AGRANDIR($listeEtats, 2^i$) ▷ ajoute des états tant que $|listeEtats| < 2^i$
5: **pour** $E \in listeEtats$ **faire**
6: **si** (METROPOLISHASTING(E, E_{Actu})) **alors** ▷ teste la condition de MH
7: $E_{Actu} \leftarrow E$
8: **fin si**
9: **fin pour**
10: $i \leftarrow i + 1$
11: **fin tant que**
12: **retourne** E_{Actu}

chaîne représente un ensemble de transactions. Pour construire cette CMMC, CFTP utilise l'algorithme de Metropolis-Hastings qui permet de la simuler avec le test suivant :

$$u_{E_{Nouv}}(0,1) < \frac{q(E_{Actu}) \times w(E_{Nouv})}{w(E_{Actu}) \times q(E_{Nouv})} \quad (1)$$

avec E_{actu} l'état actuel de la chaîne et E_{Nouv} le nouvel état proposé. $u_{E_{actu}}$ est un nombre aléatoire tiré uniformément dans [0,1], $q(E)$ désigne la fonction de potentiel et $w(E)$ la fonction de poids de l'échantillon représenté par E :

$$w(E) = w(\bigcap_{t \in E} t), \quad q(E) = \prod_{t \in E} \sqrt[|E|]{w(t)}$$

Les fonction q et w sont définies pour que les états E tel que si $\frac{w(E)}{q(E)}$ est proche de 1 alors ces états auront plus de hance d'être tirer. L'algorithme 1 présente le pseudo-code de l'étape 1 de CFTP basée directement sur l'algorithme de Metropolis-Hastings. Initialement, une liste d'états est créée puis initialisée par un ensemble de deux états ayant chacun c transactions tirées aléatoirement et uniformément en suivant le poids des transactions,i.e chaque transaction est tiré avec une probabilité suivant son poids divisé par la somme des poids de toutes les transactions (lignes 1-2). Pour faire une marche aléatoire, à chaque itération de boucle *tant que*, nous agrandissons la liste en tirant de nouveaux états de la même manière qu'à l'initialisation (ligne 4), ensuite, nous parcourons cette liste jusqu'à l'obtention d'un état dont la probabilité d'acceptation est supérieure à la valeur tirée uniformément entre 0 et 1 (lignes 5-9 de la boucle *pour*). Si aucun état n'a été accepté, alors on incrémente la valeur i et on recommence le processus.

Seconde étape de CFTP. Lors de cette étape, le tirage d'un motif se fait à partir des items communs aux c transactions échantillonnées à la première étape. L'échantillonnage se fait

selon la distribution $\mathcal{F} : \mathcal{L}(t) \rightarrow [0, 1]$ avec

$$\mathcal{F}(X) = \boldsymbol{w}(X) \prod_{i=1}^{c} \frac{\phi_i(\mathcal{D}, X)}{Z}$$

où ϕ_i correspond soit à la mesure de fréquence $freq$ ou à la mesure d'aire $aire$ qui sont des paramètres liés au facteur de fréquence, et $\boldsymbol{w}$ est une fonction de poids définie soit par $\prod_{e \in X} w(e)$, soit par $\sum_{e \in X} w(e)$ selon que le motif est tiré proportionnellement à la fréquence ou à l'aire.

2.3 Compression avec KRIMP

La compression de données est une tâche qui consiste à décrire un jeu de données avec des ensembles d'items permettant d'obtenir une représentation de taille plus réduite que celle du jeu de données initial. Cela revient à trouver l'ensemble de motifs qui décrit le mieux le jeu de données tout en minimisant la taille de la description. L'un des principes exploités pour trouver cet ensemble de motifs $\mathbf{X}$, appelé *description*, est celui de MDL (Minimum Description Length ou Longueur de Description Minimale) (Rissanen, 1978). Le principe de MDL est de trouver une description $\mathbf{X}$ de la base $\mathcal{D}$ de taille minimum :

$$L(\mathcal{D}, \mathbf{X}) = L(\mathbf{X}) + L(\mathcal{D}|\mathbf{X})$$

avec $L(\mathcal{D}, \mathbf{X})$, la taille totale de la description qui est calculée en utilisant la longueur en bits de la description $L(\mathbf{X})$ et la longueur en bits de la description du jeu de données $L(\mathcal{D}|\mathbf{X})$ lorsqu'il est codé avec $\mathbf{X}$. Les différentes approches utilisant MDL (Siebes et Kersten, 2011; Siebes et al., 2006; Vreeken et al., 2011) associent à chaque motif $X \in \mathbf{X}$ un code C de telle sorte que toutes les transactions $t \in \mathcal{D}$ soient décrites par un ensemble de codes $\{C_1, C_2, \ldots\}$. Une table de correspondance permet alors d'associer chaque motif X à un code C et à la liste des transactions dans lesquelles X est utilisé. Cette table des codes (ou « code table ») est alors exploitée pour réaliser la compression et minimiser la taille du jeu de données compressé. Vreeken et al. (2011) ont proposé une méthode exploitant le principe de MDL, dénommée KRIMP, qui prend en entrée un jeu de données et un ensemble de motifs préalablement extraits (les motifs fréquents ou fermés par exemple) et essaie de trouver le sous-ensemble de motifs qui compresse le mieux le jeu de données. Pour obtenir cette description, KRIMP commence par initialiser la table de codes avec les motifs singletons (contenant un seul item). Puis, des motifs candidats (de taille supérieure à 1) sont ordonnés par KRIMP afin de pouvoir déterminer l'ordre de sélection de ceux-ci. Dans l'algorithme de KRIMP, chaque motif candidat est testé pour être ajouté à la description. Si la nouvelle description obtenue est meilleure alors elle remplace la description actuelle.

3 Local Compression Sampler (LCS)

Dans cette section nous présentons notre contribution : l'approche LCS. De la même manière que pour CFTP nous procédons en deux étapes. Cependant, contrairement à ce dernier, dans la première étape, le tirage d'une transaction se fait proportionnellement à la taille de la

Algorithm 2 Local Compression Sampler

Entrée: une base de données $\mathcal{D}$, un facteur de fréquence c, un nombre d'itérations n
Sortie: un ensemble de motifs diversifiés
1: $CT \leftarrow$ STANDARDCODETABLE() ▷ ensemble des motifs de taille 1
2: **pour** $i = 1 \dots n$ **faire**
3: $sampleT \leftarrow$ CFTP ($\mathcal{D}$, c) ▷ Échantillonnage des transactions avec CFTP
4: $newCand \leftarrow$ FOUILLE-MOTIFS($sampleT$) ▷ Minage de motifs de $sampleT$
5: $cand \leftarrow CT \bigcup newCand$
6: $CT \leftarrow$ KRIMP ($\mathcal{D}$, $cand$) ▷ Calcul du nouveau code table avec KRIMP
7: **fin pour**
8: **retourne** CT

base. Par ailleurs, dans la deuxième étape, au lieu d'échantillonner un motif proportionnellement à une mesure d'intérêt, la fréquence, nous extrayons d'abord un ensemble de motifs candidats, puis sélectionnons parmi ces candidats un sous-ensemble de motifs à l'aide de KRIMP. L'intuition derrière LCS est que la compression réduit implicitement la redondance, en sélectionnant un ensemble de motifs qui compressent au mieux la base de données sous-jacente.

A) Première étape – Échantillonnage de transactions. Comme dans l'algorithme 2, nous utilisons la première étape de CFTP pour obtenir un sous-ensemble de la base de transactions. CFTP est utilisé comme base car les facteurs de fréquence introduits par Boley et al. (2012) permettent d'obtenir des motifs diversifiés. Toutefois, contrairement à ce dernier, le tirage d'une transaction se fait de manière uniforme par rapport à la taille de la base $\mathcal{D}$: la probabilité de tirer la transaction t dans notre algorithme est $P(t) = \frac{1}{|\mathcal{D}|}$. En utilisant cette distribution, nous espérons améliorer la diversité des transactions tirées et éviter ainsi de tirer trop souvent des transactions avec un poids élevé.

Pour constituer un état E, nous tirons c transactions de cette manière. Cette partie fait référence au petit a) dans la figure 1 : t_2 et t_5 ont chacune une probabilité de 1/8 d'être tirée alors que dans Boley et al. (2012) elles auraient eu une probabilité de 4/21.

B) Seconde étape – Extraction des motifs. La seconde différence se situe après avoir simulé la CMMC (ligne 3). Au lieu de tirer un motif suivant son poids, nous extrayons tous les motifs clos présents dans 50% des transactions échantillonnées (ligne 4), partie b) dans la figure 1 (le seul motif clos est $\{A, B, C\}$). Cet ensemble de motifs clos est ensuite ajouté à l'ensemble des candidats pour KRIMP (ligne 5). Cela permet d'identifier dans les données une collection de modèles qui donnent une bonne compression. De plus, l'utilisation de motifs clos évite de générer beaucoup plus de candidats que nécessaire et permet de réduire la redondance entre les candidats. Parce que les coûts de compression suite à l'ajout de motifs candidats sont souvent prohibitifs, l'utilisation de motifs clos permet de limiter ces coûts de manière efficace et précise.

C) Troisième étape – Compression. Bien que les motifs clos réduisent la redondance localement au sein d'un échantillon, cela ne signifie pas qu'ils sont moins redondants sur toute la base entière. La compression est utilisée ici comme un outil pour comparer les modèles. Nous utilisons KRIMP pour filtrer les motifs candidats après l'échantillonnage (ligne 6). Avec un ensemble de motifs candidats notés C, KRIMP sélectionnera de manière heuristique les motifs qui forment la table de codes de longueur minimale $\mathbf{X} \subseteq C$. L'ensemble des candidats com-

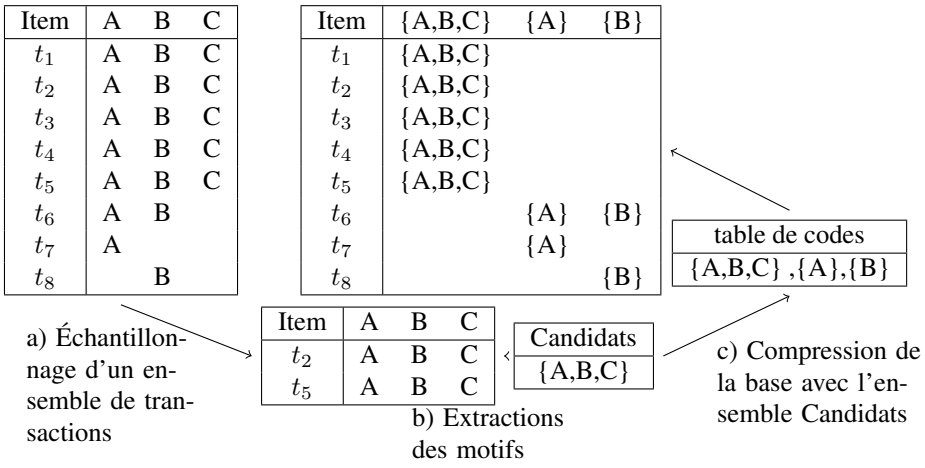

FIG. 1 – Exemple illustrant notre méthode.

prend toujours l'ensemble des motifs singletons. C'est pourquoi dans la partie c) de la figure 1, nous obtenons $\{\{A, B, C\}, \{A\}\{B\}\}$ comme table de codes.

L'intuition derrière l'utilisation de KRIMP vient du fait que trouver la longueur minimale de description (MDL) équivaut à identifier un ensemble de motifs avec de grandes valeurs d'aire, et ces aires ne doivent pas se chevaucher de manière significative, ni avoir un indice de Jaccard élevé. En effet, il est peu probable que deux motifs syntaxiquement très similaires soient choisis ensemble puisque leur contribution à la compression est probablement redondante. Nous démontrerons par des expérimentations que notre approche permet d'obtenir de meilleurs résultats que ceux rapportés dans la littérature en termes de diversité.

D) Répétition de la boucle. Nous produisons un ensemble de motifs en répétant les trois étapes présentées précédemment. L'algorithme 2 donne le pseudo-code de notre approche. Le processus d'échantillonnage se déroule de manière itérative pendant un nombre raisonnable d'itérations n. L'algorithme maintient une structure de données interne : la table de codes CT. Initialement, la table de codes est initialisée par la table de codes standard. A chaque itération, l'algorithme échantillonne un ensemble de c transactions. L'ensemble de tous les motifs clos avec une fréquence d'au moins $\lceil \frac{c}{2} \rceil$ est ensuite extrait à l'aide de l'algorithme LCM (ligne 4). Les motifs extraits sont ajoutés à ceux déjà présents dans CT, puis utilisés par KRIMP pour calculer la nouvelle table de codes qui donne une bonne compression. La table de code retournée par KRIMP deviendra alors une partie du prochain ensemble candidat (ligne 5). Ce processus est répété n fois. Dans nos expérimentations, nous avons évalué deux variantes : un appel à KRIMP à chaque itération, et un appel à KRIMP après x itérations, ce qui permet la construction d'un ensemble de candidats avec plusieurs échantillons.

4 Expérimentations

Nous évaluons les performances de notre méthode LCS en analysant la diversité des motifs échantillonnés et le temps d'exécution. Nous avons utilisé neuf bases de données issues

du FIMI repository [1], du UCI Machine Learning repository [2], du CP4IM repository [3] et d'un cas d'étude réel basé sur des levures biologiques. Ces bases ont toutes des caractéristiques différentes, leur taille diffère ainsi que la densité de la base de données. Nous comparonq notre méthode LCS à CFTP et KRIMP. Pour la comparaison à CFTP, nous analysons le temps d'exécution et la diversité des ensembles de motifs obtenus sur 100 itérations et nous calculons la moyenne sur 10 exécutions. Pour KRIMP, nous comparons la diversité et le taux de compression en utilisant différents ensemble des motifs fréquents extraits suivant différents seuils. Les algorithmes KRIMP et LCM sont en C++ alors que CFTP est en Java. Notre algorithme est en Python pour faire la passerelle. Les expérimentations sont faites sur un processeur 12th Gen Intel® Core™ i7-12700H avec le système d'exploitation Linux ayant 4 Go de mémoire RAM. KRIMP est lancé en parallèle avec quatre cœurs pour la compression. Le *time-out* est d'une heure.

4.1 Comparaison en termes de temps de calcul

Dans les tableaux qui suivent, LCS-x ($x \in \{1, 10, 100\}$ correspond à la variante de LCS qui fait appel à KRIMP après x itérations. Par exemple, LCS-10 indique qu'il faut 10 itérations pour construire l'ensemble de candidats, puis lancer KRIMP sur cet ensemble union la table de codes actuelle. Nous considérons deux valeurs pour le paramètre de fréquence c : 3 et 4. Le tableau 1 montre les résultats obtenus. Pour les bases de données relativement denses et très denses, CFTP est plus rapide, excepté pour splice1 où notre approche obtient de meilleurs temps de calcul avec $c = 3$. Les résultats pour splice1 indiquent que la seconde étape de CFTP est plus lente que l'extraction puis compression des motifs. Pour les bases dites rectangulaires, i.e. toutes les transactions ont le même nombre d'items, l'écart entre CFTP et LCS est uniquement lié à l'extraction et à la compression. Pour les bases non rectangulaires et peu denses (retail et pumsb*), LCS est plus rapide car la taille des intersections obtenues est trop petite par rapport à la taille moyenne des transactions d'un échantillon. Toutefois, malgré le fait que la taille moyenne des transactions dans retail soit de 10, chaque échantillon de transactions comporte beaucoup de transactions de (très) grande taille, ce qui induit une convergence lente de CFTP. Il est facile de remarquer que CFTP converge très vite si les bases sont denses car les intersections vont souvent être grandes. Fait intéressant à noter et que CFTP n'est pas capable de terminer après 1h de calcul même avec des petites valeurs de c.

4.2 Comparaison en termes de diversité

Pour une paire de motifs, l'indice de Jaccard est un bon indicateur de diversité. Pour autant, l'indice de Jaccard n'est pas utilisable pour un ensemble de motifs. Nous allons utiliser les fonctions de répartition cumulatives (ou Cumulative Distribution Function(CDF) sur les indices de Jaccard pour obtenir une forme visuelle de la distribution des paires de motifs en fonction de leurs indices de Jaccard. Pour un ensemble de motifs $\mathbf{X} = \{X_1, X_2, ..., X_k\}$ et une base de données $\mathcal{D}$: $CDF(\mathcal{D}, \mathbf{X}, \theta) = |\{(i, j) : Jac(X_i, X_j) \leq \theta, 1 \leq i < j \leq k\}| \times \frac{2}{k(k-1)}$. Par manque de place, nous ne montrons pas les graphiques pour toutes les bases. La figure 2 est constituée de deux graphiques, un pour la base chess et un autre pour pumsb. Ils ont plusieurs

1. http ://fimi.uantwerpen.be/data/
2. https ://archive.ics.uci.edu/datasets
3. https ://dtai.cs.kuleuven.be/CP4IM/datasets/

	c=3				c=4			
Intance	CFTP	LCS-1	LCS-10	LCS-100	CFTP	LCS-1	LCS-10	LCS-100
hepatitis	**0.2**	16.4	16.4	16.4	**3.5**	21.1	21.1	21.2
chess	**0.2**	17.5	17.5	17.5	**1.0**	21.0	21.0	21.0
eisen	-	379.6	379.6	379.6	-	-	-	-
splice1	911.6	865.17	**865,16**	**865,16**	-	-	-	-
mushroom	**0.17**	19.0	19.0	19.1	**0.77**	20.8	20.8	20.9
pumsb	**43.7**	110.8	110.8	111.0	**937.0**	1022.9	1023.0	1023.5
pumsb_star	133.3	95.5	**95.4**	95.6	2144.9	**693.5**	693.6	693.9
connect	**1.2**	50.3	50.2	55.8	**8.0**	73.3	76.3	113.4
retail	-	**61.9**	**61.9**	**61.9**	-	**70.7**	**70.7**	70.8

TAB. 1 – Résultats de temps d'exécution en secondes après 100 itérations.

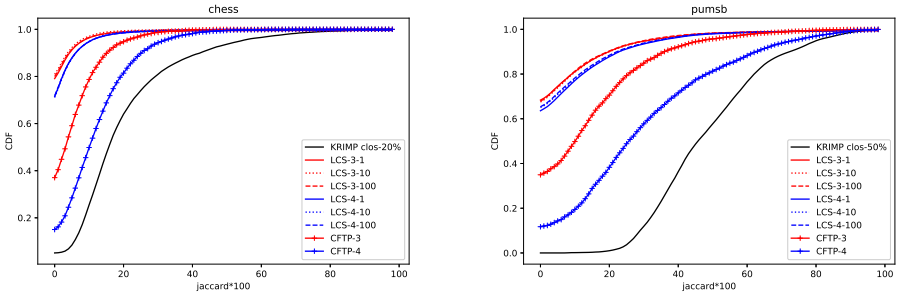

FIG. 2 – Fonctions cumulatives de l'indice de Jaccard pour *chess* et *pumsb*.

courbes, chacune représentant un paramétrage d'algorithmes. Les courbes pour KRIMP sont suivies du type de motifs extraits et du seuil utilisé pour leur extraction. Pour CFTP et LCS, le premier chiffre correspond au facteur de fréquence utilisé, alors que le deuxième nombre correspond au nombre d'itérations nécessaires avant l'appel à KRIMP dans LCS. Les résultats de KRIMP montrent que les motifs obtenus sont très peu diversifiés. Les résultats de KRIMP montrent également que plus le seuil utilisé est bas, plus les motifs sont diversifiés, qu'ils soient fréquents ou clos. Cela s'explique par le fait qu'en ne sélectionnant que des motifs très fréquents, il y a de grandes chances qu'ils aient des transactions en commun. Les résultats de CFTP respectent les conclusions données dans Boley et al. (2012) : plus le facteur de fréquence est grand, plus les motifs tirés sont fréquents. LCS n'échappe pas au problème, on observe aussi une petite baisse de diversité au début. Néanmoins, CFTP retourne un ensemble de motifs moins diversifiés que LCS quel que soit le facteur de fréquence.

4.3 Comparaison en termes de compression

Notre approche n'est pas une autre méthode de compression destinée à obtenir une meilleure compression. Ici, nous visons à évaluer la capacité de notre méthode à obtenir des modèles intéressants. Un meilleur taux de compression signifie que les motifs extraits doivent moins se chevaucher. La compression indiquée dans les tableaux 2 et 3 est le pourcentage de compres-

Instance	LCS-3-1	LCS-3-10	LCS-3-100	LCS-4-1	LCS-4-10	LCS-4-100
hepatitis	23.6	22.8	24.3	27.9	27.7	**28.6**
chess	36.4	37.2	37.5	**48.3**	**48.3**	47.8
eisen	3.0	**3.2**	2.9	-	-	-
splice1	**3.2**	3.1	3.0	-	-	-
mushroom	50.8	49.1	52.8	**57.2**	56.6	56.3
pumsb	14.4	14.7	14.9	16.1	**16.3**	16.2
pumsb*	15.6	15.7	15.7	16.6	16.7	**17.2**
connect	48.8	49.9	48.2	**62.1**	61.7	61.8
retail	1.7	**1.8**	1.7	1.6	1.7	1.7

TAB. 2 – Comparaisons en termes de taux de compression en % pour LCS.

sion. Pour une base de donnée $\mathcal{D}$ et une table de codes $\mathbf{C}$, nous affichons :

$$TauxComp(\mathcal{D}, \mathbf{C}) = 100 \times (1 - \frac{L(\mathcal{D}, \mathbf{C})}{L(\mathcal{D}, \mathcal{I})})\%$$

Le tableau 2 montre les taux de compression obtenus par LCS sur différents datasets après 100 itérations. En utilisant un facteur de fréquence plus élevé, le taux de compression est meilleur, ce qui s'explique par le fait que les motifs trouvés sont plus fréquents. Il est difficile de dire quel paramétrage est le meilleur pour obtenir la meilleure compression. Néanmoins, avoir un facteur de fréquence de 3 permet d'obtenir un bon compromis entre taux de compression et temps de calcul. Nous pouvons aussi constater que la fréquence d'appel à KRIMP a peu d'impact sur la compression.

Concernant KRIMP, le tableau 3 montre que plus le seuil utilisé est bas, meilleur est la compression. Bien que l'idée semble contre-intuitive, elle s'explique par la redondance des motifs avec de hautes fréquences. Baisser le seuil implique une explosion dans le nombre de motifs, ce qui rend KRIMP très lent sur les grosses bases de données comme connect, pumsb, pumsb* et retail. En comparant les tableaux 2et 3, nous observons que notre approche est en général moins bonne en termes de compression. Pour chess, nous obtenons un taux de compression de 48,3% pour un temps de 21 secondes contre 59,4% avec les clos à 30% pour un temps similaire avec KRIMP. Pour connect, nous avons une meilleure compression comparée à l'ensemble des fréquents à 60%, ce qui confirme notre hypothèse en plus d'être plus rapide.

5 Conclusion

Dans cet article, nous avons introduit une nouvelle approche qui utilise les principes d'échantillonnage de motifs et de compression de données afin d'échantillonner des motifs plus diversifiés. Notre approche exploite une méthode exacte, le Linear Closed Mining (LCM), pour extraire un ensemble de motifs clos à partir d'un ensemble candidat de transactions tiré en exploitant une approche basée sur les chaînes de Markov Monte Carlo. Une étape de compression est ensuite effectuée pour sélectionner parmi ces candidats un sous-ensemble de motifs à l'aide de l'algorithme KRIMP. L'intuition derrière notre approche est que la compression réduit implicitement la redondance, en sélectionnant un ensemble de motifs ou une nouvelle description qui compresse au mieux la base de données sous-jacente. Cette description est ensuite utilisée lors des itérations suivantes d'échantillonnage. Les résultats expérimentaux sur plusieurs

	Clos				Frequent			
Instance	Seuil	Tps extract	Tps comp	Taux Comp	Seuil	Tps extract	Tps comp	Taux Comp
hepatitis	10	1.7	10.9	52.8	30	1.7	1.4	47.6
	5	3.5	24.1	53.2	20	3.5	28.9	54.6
	1	4.6	43.5	53.2				
chess	30	5.8	16.4	59.4	50	5.8	4.4	54.8
	20	25.6	85.3	62.8	40	25.6	24.7	59.3
mushroom	1	0.2	0.3	69.3	5	0.2	11.2	64.3
	0.5	0.2	0.7	72.9				
	0.03	0.4	0.8	75.4				
pumsb	55	2.5	917.4	37.3	60	4.8	2411.5	34.6
	50	6.3	2122.8	41.6				
pumsb*	20	0.6	277.3	50.4	40	0.1	45.0	33.4
	15	1.4	833.4	53.2	30	0.3	311.7	41.3
	10	4.7	3221.5	56.6	25	0.7	1391.0	44.5
splice1	5	0.5	95.4	26.5	5	0.2	92.3	26.5
	2	3.9	302.1	30.4	2	1.5	291.6	30.4
	1	21.9	803.0	31.0	1	28.54	1474.4	31.0
connect	30	0.5	133.1	60.0	60	3.3	2758.9	55.9
	18	1.7	461.3	64.6				
	15	2.7	893.8	66.1				
	10	6.3	1717.8	68.9				
eisen	0.5	0.1	0.7	10.4	1	1.53	0.7	7.6
	0.1	1.3	5.0	21.2	0.5	1.3	13.77	10.8
retail	0.4	0.1	0.1	3.0	0.4	0.1	0.8	3.0
	0.003	0.1	0.1	7.2	0.003	4.1	1407.5	7.2

TAB. 3 – Résultats de KRIMP avec les temps en secondes

ensembles de données UCI ont montré que la diversité des motifs échantillonnés est considérablement améliorée tout en conservant des temps d'exécution comparables. Ces résultats confirment l'intérêt de notre approche pour obtenir des motifs diversifiés grâce à la compression de données. Notre travail est principalement basé sur la fréquence, mais notre méthode peut être adaptée à d'autres mesures d'intérêt, voire à des ensembles de mesures, permettant un échantillonnage multicritère. En intégrant les retours des utilisateurs, la méthode pourrait orienter l'échantillonnage vers des parties de la base de données plus susceptibles d'intéresser l'utilisateur. De nombreuses possibilités se présentent avec l'intégration potentielle de la programmation par contraintes dans le processus d'échantillonnage.

Références

Agrawal, R. et R. Srikant (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th VLDB*, Santiago de Chile, Chile, pp. 487–499.

Bendimerad, A., J. Lijffijt, M. Plantevit, C. Robardet, et T. D. Bie (2020). Gibbs sampling subjectively interesting tiles. In *Advances in Intelligent Data Analysis XVIII - 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27-29, 2020, Proceedings*, Volume 12080 of *Lecture Notes in Computer Science*, pp. 80–92. Springer.

Boley, M., C. Lucchese, D. Paurat, et T. Gärtner (2011). Direct local pattern sampling by efficient two-step random procedures. In C. Apté, J. Ghosh, et P. Smyth (Eds.), *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pp. 582–590. ACM.

Boley, M., S. Moens, et T. Gärtner (2012). Linear space direct pattern sampling using coupling from the past. In *Proceedings of KDD 12*, pp. 69–77. ACM.

Diop, L. (2020). *Echantillonnage sous contraintes de motifs Structures. (Constrained Sampling of Structured Patterns)*. Ph. D. thesis, Gaston Berger University, Saint-Louis, Senegal.

Diop, L., C. Diop, A. Giacometti, D. Li, et A. Soulet (2020). Sequential pattern sampling with norm-based utility. *Knowl. Inf. Syst. 62*(5), 2029–2065.

Dzyuba, V., M. van Leeuwen, et L. D. Raedt (2017). Flexible constrained sampling with guarantees for pattern mining. *Data Min. Knowl. Discov. 31*(5), 1266–1293.

Hasan, M. A. et M. J. Zaki (2009). Output space sampling for graph patterns. *Proc. VLDB Endow. 2*(1), 730–741.

Hilderman, R. et H. Hamilton (2001). *Knowledge Discovery and Measures of Interest*, Volume 638. Springer.

Liu, B., W. Hsu, L.-F. Mun, et H.-Y. Lee (1999). Finding interesting patterns using user expectations. *IEEE Transactions on Knowledge and Data Engineering 11*(6), 817–832.

Rissanen, J. (1978). Paper : Modeling by shortest data description. *Automatica 14*(5), 465–471.

Siebes, A. et R. Kersten (2011). A structure function for transaction data. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pp. 558–569. SIAM / Omnipress.

Siebes, A., J. Vreeken, et M. van Leeuwen (2006). Item sets that compress. In J. Ghosh, D. Lambert, D. B. Skillicorn, et J. Srivastava (Eds.), *Proceedings of SIAM DM 2006, April 20-22, 2006, Bethesda, MD, USA*, pp. 395–406. SIAM.

Tan, P.-N., V. Kumar, et J. Srivastava (2002). Selecting the right interestingness measure for association patterns. In *KDD*, pp. 32–41.

Vreeken, J., M. Van Leeuwen, et A. Siebes (2011). Krimp : mining itemsets that compress. *Data Mining and Knowledge Discovery 23*(1), 169–214.

Yao, H., H. J. Hamilton, et C. J. Butz (2004). A foundational approach to mining itemset utilities from databases. In *SDM*.

Zhong, N., Y. Yao, et M. Ohishima (2003). Peculiarity oriented multidatabase mining. *IEEE Transactions on Knowledge and Data Engineering 15*(4), 952–960.

Summary

Exhaustive methods of pattern extraction in a database pose real obstacles to speed and output control of patterns: a large number of patterns are extracted, many of which are redundant. Pattern extraction methods through sampling, which allow for controlling the size of the outputs while ensuring fast response times, provide a solution to these two problems. However, these methods do not provide high-quality patterns: they retu rn patterns that are very infrequent in the database. Furthermore, they do not scale. To ensure more frequent and diversified patterns in the output, we propose integrating compression methods into sampling to select the most representative patterns from the sampled transactions. We demonstrate that our approach improves the state of the art in terms of diversity of produced patterns.