SPAMS, une nouvelle approche incrémentale pour l'extraction de motifs séquentiels fréquents dans les *Data streams*

Lionel VINCESLAS*, Jean-Émile SYMPHOR*, Alban MANCHERON** et Pascal PONCELET***

*GRIMAAG, Université des Antilles et de la Guyane, Martinique, France.
{lionel.vinceslas,je.symphor}@martinique.univ-ag.fr

**alban@mancheron.infos.st
*** EMA-LG2IP/site EERIE, Parc Scientifique Georges Besse, 30035 Nîmes Cedex, France.
pascal.poncelet@ema.fr

Résumé. L'extraction de motifs séquentiels fréquents dans les data streams est un enjeu important traité par la communauté des chercheurs en fouille de données. Plus encore que pour les bases de données, de nombreuses contraintes supplémentaires sont à considérer de par la nature intrinsèque des streams. Dans cet article, nous proposons un nouvel algorithme en une passe : SPAMS, basé sur la construction incrémentale, avec une granularité très fine par transaction, d'un automate appelé SPA, permettant l'extraction des motifs séquentiels dans les streams. L'information du stream est apprise à la volée, au fur et à mesure de l'insertion de nouvelles transactions, sans pré-traitement a priori. Les résultats expérimentaux obtenus montrent la pertinence de la structure utilisée ainsi que l'efficience de notre algorithme appliqué à différents jeux de données.

1 Introduction

Concerné par de nombreux domaines d'application (e.g. le traitement des données médicales, le marketing, la sécurité et l'analyse financière), l'extraction de motifs séquentiels fréquents est un domaine de recherche actif qui intéresse la communauté des chercheurs en fouille de données. Initialement, les premiers travaux présentés traitent du cas des bases de données statiques et proposent des méthodes dites exactes d'extraction de motifs séquentiels. On peut citer, à titre d'exemple, les algorithmes GSP, SPADE, PrefixSpan et SPAM, respectivement proposés par Srikant et Agrawal (1996); Zaki (2001); Pei et al. (2001); Ayres et al. (2002). Plus récemment ces dernières années, de nouvelles applications émergentes, telles que l'analyse de traffic dans les réseaux, la fouille de données "clickstream"[1] ou encore la détection de fraudes et d'intrusions, induisent de nouvelles problématiques qui impactent les méthodes de fouilles. En

[1] clickstream : flot de requêtes d'utilisateurs sur des sites web.

effet, ces applications supposent la prise en compte d'un nouveau type de données plus connus sous l'intitulé *data streams*. Le traitement d'un data stream doit satisfaire de nouvelles contraintes. Les données sont générées rapidement de façon continue voire illimitée et donc ne peuvent être complètement stockées en mémoire. On ne peut, de plus, s'autoriser qu'un seul et unique passage sur les données qui doivent être traitées le plus rapidement possible. Ainsi, les premières méthodes d'extraction de motifs séquentiels ne sont plus du tout adaptées. Il a été montré dans (cf. Garofalakis et al. (2002)) que des méthodes approchées étaient tout à fait adaptées au contexte des streams. Toutefois, ces méthodes doivent trouver un compromis satisfaisant entre les temps de réponses, la consommation mémoire et la qualité des résultats tant en précision qu'en rappel. A l'heure actuelle, il existe peu de travaux qui ont abordé la problématique de l'extraction de motifs dans les streams et ils se répartissent en deux catégories. La première concerne les méthodes où le traitement des données s'effectue par batchs de transactions. On peut citer, les travaux de Kum et al. (2002); Marascu et Masseglia (2006) avec respectivement les algoritmes ApproxMap et SMDS. Ils effectuent un clustering du stream en plusieurs batchs, selon la similarité entre les motifs et procèdent par compression de séquences similaires selon une méthode d'alignement multiple, afin de réduire la consommation mémoire. Ces méthodes présentent l'inconvénient d'effectuer un traitement hors ligne du stream, car ne peuvent s'exécuter qu'après l'obtention d'un groupe de données. Elles requièrent donc des tailles critiques suffisantes pour les batchs, ce qui n'est pas complètement réaliste compte tenu des contraintes inhérentes aux streams. La seconde catégorie concerne les méthodes qui opèrent directement par transactions. Chang et Lee (2005) ont proposé l'algorithme eISeq, basé sur une structure d'arbre, effectuant un traitement en ligne des données par transaction parcourue en une seule passe. Plus les motifs sont longs à traiter, moins l'algorithme est performant du fait de la phase de génération de tous les sous motifs. Par exemple, si $< a_1, \cdots, a_i >$ est un motif, il y a $(2^i - 1)$ sous motifs à créer. Pour pallier cette difficulté, Li et Chen (2007) ont présenté l'algorithme GraSeq, développant une approche à partir d'une structure de type graphe orienté pondéré permettant de limiter la phase de génération de sous motifs. Toutefois, cette approche suppose un pré-traitement des transactions pour en effectuer un regroupement.

Dans cet article, nous proposons un nouvel algorithme *une-passe* : *SPAMS* (***S**equential **P**attern **A**utomaton for **M**ining **S**treams)*. Il est basé sur la construction et la mise à jour incrémentale par transaction d'une structure d'automate : *SPA (**S**equential **P**attern **A**utomaton)* et permet l'extraction des motifs séquentiels dans les data streams. Il ne nécessite pas d'effectuer un pré-traitement visant à opérer un regroupement de transactions. SPA est un automate dont, seuls sont parcourus, les états et les transitions qui doivent l'être, lors de l'insertion des nouvelles transactions du stream. On évite, ainsi les parcours multiples de la structure, prohibitifs et pénalisants pour les performances des algorithmes. De plus, notre approche permet de réduire sensiblement la génération des motifs candidats tout en conservant une qualité d'approximation très satisfaisante tant en rappel qu'en précision. La suite de l'article est organisée de la manière suivante. La section 2 présente formellement la problématique. La section 3 rapelle les concepts requis pour présenter notre approche qui est décrite à la section 4. Les expérimentations sont fournies à la section 5 et la conclusion est proposée dans la

dernière section.

2 Définition du problème

Dans cette section, nous donnons une définition formelle du problème de l'extraction des motifs séquentiels dans les data streams. Tout d'abord, nous résumons la description formelle présentée par Srikant et Agrawal (1996), classiquement utilisée dans le cas des bases de données statiques. Nous étendons ensuite la problématique au cas des data streams.

Soit $\mathbb{I} = \{i_1, i_2, \ldots, i_m\}$ un ensemble ordonné d'items utilisés dans une base de données *DB* de transactions, où chaque transaction t_r est identifiée de manière unique par un Cid, un temps et est associée à un ensemble d'items de $\mathbb{I}$. Un ensemble $\mathcal{X} \subseteq \mathbb{I}$ est appelé un *itemset*. Une séquence s est un ensemble d'itemsets ordonnés selon leur temps et est représentée par $< s_1 s_2 \cdots s_n >$, où s_j, pour $j \subseteq [1..n]$, est un itemset. Une k-séquence est une séquence de k items ou de longueur k. Une séquence $S' =< s'_1 s'_2 \cdots s'_n >$ est une sous séquence d'une autre séquence $S =< s_1\ s_2\ \cdots\ s_n >$, que l'on note $S' \prec S$ s'il existe des entiers $i_1 < i_2 < \cdots i_j \cdots < i_n$ tels que $s'_1 \subseteq s_{i_1}, s'_2 \subseteq s_{i_2}, \cdots, s'_n \subseteq s_{i_n}$.

Toutes les transactions relatives à un même Cid sont regroupées et triées selon leur ordre d'apparition, pour obtenir une séquence de données. Une séquence de données contient une séquence S, si S en est une sous-séquence. Le support d'une séquence S, noté supp(S), correspond au nombre d'occurences de S dans *DB*. Pour décider si une séquence est fréquente ou non, une valeur de support minimum, notée σ, est spécifiée par l'utilisateur. Une séquence S est dite θ-fréquente si $supp(S) \geq \sigma$, où $\sigma = \lceil \theta \times |DB| \rceil$ avec $\theta \in]0; 1]$ et $|DB|$ la taille de la base de données. Etant donnée une base de données de transactions de clients Cid, le problème de l'extraction de motifs séquentiels consiste à trouver toutes les séquences, dont le support est supérieur ou égal au support minimum fixé par l'utilisateur dans *DB*. Cette problématique, étendue au cas des data streams, peut s'exprimer simplement comme suit. Formellement, un data stream *DS* peut être défini par une suite de transactions $DS = (T_1, T_2, \cdots, T_j, \cdots)$. Chacune des transactions, identifiée par un Tid, est associée à un Cid (voir par exemple la table 1). L'extraction des motifs séquentiels fréquents revient à trouver toutes les séquences, dont le support est supérieur ou égal au support minimum fixé par l'utilisateur pour la partie connue du stream à un moment donné.

C_1			C_2		C_3	
T_1	T_2	T_3	T_4	T_5	T_6	T_7
bd	*abd*	*acd*	*bcd*	*bd*	*ab*	*c*

Tab. 1 – *Ensembles des transactions par Cid construites sur* $\mathbb{I} = \{a, b, c, d\}$.

3 Prérequis sur les couvertures statistiques

Nous reprenons ci-dessous le théorème proposé et prouvé par Laur et al. (2007) concernant les couvertures statistiques. Les auteurs proposent de biaiser la valeur du support fixé par l'utilisateur afin d'obtenir des ensembles de motifs qui permettent de maximiser les résultats tant en rappel qu'en précision.

Théorème 1 $\forall\theta, 0 < \theta \leq 1, \forall\delta, 0 < \delta \leq 1$, *soit m et* m^* *respectivement le nombre de motifs (*θ*-fréquent et* θ*-infrequent) dans la partie connue du stream et dans tout le stream, si on choisit* ϵ *tel que :*

$$\epsilon \quad \geq \quad \sqrt{\frac{1}{2\,m} \ln \frac{m^*}{\delta}} \ ,$$

cela implique que le `Rappel` $= 1$ *et respectivement la* `Précision` $= 1$ *avec une probabilité d'au moins* $1 - \delta$*, après suppression de tous les motifs qui ne sont pas* θ'*-frequent à partir de l'observation, avec* $\theta' = \theta - \epsilon$ *et respectivement* $\theta' = \theta + \epsilon$*. Le paramètre* δ *est le paramètre de risque statistique fixé par l'utilisateur et les valeurs* $\theta' = \theta \pm \epsilon$ *sont les supports statistiques.*

La **sup-**(θ, ϵ)**-couverture** représente le quasi-optimal plus petit ensemble de motifs avec une probabilité d'au moins $1 - \delta$ contenant tous les motifs qui sont θ-fréquent dans tout le stream (éventuellement infini). Il n'y a pas de résultats faux négatifs avec une forte probabilité. La **inf-**(θ, ϵ)**-couverture** représente le quasi-optimal plus grand ensemble de motifs avec une probabilité d'au moins $1 - \delta$ contenant tous les motifs qui sont θ-fréquent dans tout le stream (éventuellement infini). Dans cet ensemble, il n'y a pas de résultats faux positifs avec une forte probabilité, mais avec des faux négatifs. Par quasi-optimal, les auteurs expriment que toute technique obtenant de meilleures bornes est condamnée à se tromper (i.e. le critère à maximiser n'est plus égal à 1). Ils précisent aussi, qu'ils ne font aucune hypothèse concernant la distribution du stream.

4 L'approche SPAMS

Notre approche, repose sur la construction incrémentale d'une structure d'automate qui permet l'indexation des motifs séquentiels du stream. Le traitement du stream s'effectue avec une granularité très fine, par transaction et pour chaque transaction item par item. Il n'est point nécessaire de procéder à un pré-traitement pour regrouper les transactions par Cid. Aussi, nous ne pré-supposons au départ de l'algorithme ni la connaissance de l'alphabet des items ni la connaissance du nombre des Cid du stream. Cette information est apprise incrémentalement, à la volée, au fur et à mesure de l'insertion des nouvelles transactions du stream. Par ailleurs, afin d'obtenir une qualité d'approximation satisfaisante, tant en rappel qu'en précision, en plus des motifs séquentiels θ-fréquents, nous indexons également les motifs $(\theta - \epsilon)$ fréquents de la couverture statistique supérieure. On conserve donc ici, le nombre minimal de motifs candidats supplémentaires limitant ainsi l'explosion combinatoire.

4.1 SPA : l'automate des motifs séquentiels

Nous définissons formellement dans cette section l'automate SPA. Pour une information détaillée sur la théorie des automates, nous suggérons la présentation faite par Hopcroft et Ullman (1990). SPA est un automate fini déterministe, i.e. un quintuple tel que $SPA = (\mathcal{Q}, q_0, F, \mathbb{I}, \delta)$ où

- $\mathcal{Q}$ est un ensemble fini d'états dont chaque état est représenté par un identifiant unique associé à une valeur de support.
- $q_0 \in \mathcal{Q}$ est l'état initial, dont le support correspond au nombre de clients lu pour le stream en cours d'acquisition.
- $F \subseteq \mathcal{Q}$ est l'ensemble des états finaux, i.e. les états ayant un support supérieur ou égal au support seuil.
- $\Sigma \subseteq \mathbb{I}$ est l'alphabet des items reconnus.
- $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ est la fonction de transition non totale permettant d'indexer les motifs séquentiels fréquents du stream.

4.2 L'algorithme SPAMS

4.2.1 Présentation

Le principe de SPAMS repose sur l'idée centrale suivante concernant la construction de l'automate. Nous construisons SPA en imposant la condition suivante : un état ne peut être accessible que par un seul item, avec une valeur de support qui correspond au nombre d'occurences de chaque motif séquentiel reconnu à cet état. Si après lecture d'un item la condition précédente ne peut plus être vérifiée pour un état (i.e. le nombre d'occurences de certains motifs séquentiels reconnus à cet état doit être incrémenté), il convient de dupliquer l'état. Le nouvel état créé, recoit les seules transitions entrantes appartenant aux motifs séquentiels dont le support doit être incrémenté et recoit également les transitions sortantes de l'état dupliqué. Après initialisation de l'automate SPA, notre algorithme se décompose en trois modules principaux. Le premier, Inserer, permet la lecture et l'insertion d'un nouvel item acquis dans le *stream*. Le second, Supprimer, permet de supprimer, pendant l'exécution du module Inserer, les états et les transitions relatifs aux séquences devenues non $(\theta - \epsilon)$-fréquentes. Le dernier, Suivant, permet de terminer le traitement d'une transaction avant de passer à la transaction suivante. Nous utiliserons le symbole '-' comme séparateur des itemsets à l'intérieur des séquences.

- **Initialisation de l'automate** : création de l'état initial $\mathbf{q_0}$ et d'un état transitoire, noté q_∞, utilisé pendant la construction. Ce dernier n'appartient pas à l'automate final.
- **Lecture et Insertion d'un item α pour un client *cid* donné** : à chaque insertion d'un *item* α, il va s'agir dans un premier temps de déterminer tous les états derrières lesquels il faudra ajouter une transition libellée par l'item α, et cela afin d' indexer incrémentalement toutes les sous-séquences incluses dans la séquence en cours de lecture. Pour cela on maintient une liste $\mathcal{Q}_{\texttt{cid}} \subseteq \mathcal{Q}$ correspondant aux *états atteints* lors du traitement des séquences du client *cid*.

Σ	: Alphabet de l'automate.
$\mathcal{Q}$	: Ensemble des états de l'automate.
$\mathcal{T}$	: Ensemble des transitions de l'automate.
$\mathcal{T}_s$	: Ensemble des transitions entrantes sur l'état $s \in \mathcal{Q}$.
$\mathcal{Q}^-$	: Ensemble des états de l'automate accessibles par l'item '-'.
$\mathcal{Q}_{\texttt{cid}}$	: Ensemble des *états atteints* par le client `cid`.
$\mathbf{\Sigma}_{\texttt{cid}}$	: Alphabet du client `cid`.
$\mathcal{T}^a$	: Ensemble des *transitions atteintes* lors de l'insertion d'un item.
$\lvert s \rvert$	: Support de l'état $s \in \mathcal{Q}$.
$s \overset{\alpha}{\mapsto} s'$	: Transition de l'état s à l'état s', étiqueté par α.
$\mathcal{C}$	: Ensemble des identifiants client.

TAB. 2 – *Notations utilisées dans* SPAMS

1. Si $\texttt{cid} \notin \mathcal{C}$, alors $\mathcal{C} = \mathcal{C} \cup \{\ \texttt{cid}\ \}$, $\mathcal{Q}_{\texttt{cid}} = \{\ \texttt{q}_\emptyset\ \}$ et $\Sigma_{cid} = \emptyset$.
2. Ensuite, $\forall \texttt{s} \in \mathcal{Q}_{\texttt{cid}}$, si $\nexists\ \texttt{s}' \in \mathcal{Q} \mid \texttt{s} \overset{\alpha}{\longmapsto} \texttt{s}' \in \mathcal{T}$, alors $\mathcal{T} = \mathcal{T} \cup \{\texttt{s} \overset{\alpha}{\longmapsto} \texttt{q}_\infty\}$.
3. On récupère l'ensemble des *dernières transitions atteintes* $\mathcal{T}^{\texttt{a}} \subseteq \mathcal{T}$ tel que $\mathcal{T}^{\texttt{a}} = \left\{\ \texttt{s} \overset{\alpha}{\longmapsto} \texttt{s}' \in \mathcal{T} \mid \texttt{s} \in \mathcal{Q}_{\texttt{cid}} \text{ et } \texttt{s}' \in \mathcal{Q}\ \right\}$.
4. Pour tout état $\texttt{s}'$ tel que $\texttt{s} \overset{\alpha}{\longmapsto} \texttt{s}' \in \mathcal{T}^{\texttt{a}}$, on effectue les opérations suivantes :

   (i) Si $\texttt{s}' \neq \texttt{q}_\infty$ et $|\mathcal{T}_{\texttt{s}'}| = |\mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^{\texttt{a}}|$, alors $\mathcal{Q}_{\texttt{cid}} = \mathcal{Q}_{\texttt{cid}} \cup \{\ \texttt{s}'\ \}$.
   De plus, si $\alpha \notin \mathbf{\Sigma}_{\texttt{cid}}$ alors $\mathbf{\Sigma}_{\texttt{cid}} = \mathbf{\Sigma}_{\texttt{cid}} \cup \{\ \alpha\ \}$ et $|\texttt{s}'| = |\texttt{s}'| + 1$.

   (ii) Sinon, *i.e.* $\texttt{s}' = \texttt{q}_\infty$ ou $|\mathcal{T}_{\texttt{s}'}| \neq |\mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^{\texttt{a}}|$, on crée un nouvel état p et $\mathcal{Q}_{\texttt{cid}} = \mathcal{Q}_{\texttt{cid}} \cup \{\ \texttt{p}\ \}$. Si $\alpha \notin \mathbf{\Sigma}_{\texttt{cid}}$ alors $\mathbf{\Sigma}_{\texttt{cid}} = \mathbf{\Sigma}_{\texttt{cid}} \cup \{\ \alpha\ \}$ et $|\texttt{p}| = |\texttt{s}'| + 1$, sinon $|\texttt{p}| = |\texttt{s}'|$.
   Ensuite, $\forall \texttt{s}' \overset{\beta}{\mapsto} \texttt{z} \in \mathcal{T}$, avec $\beta \in \mathbf{\Sigma}$ et $\texttt{z} \in \mathcal{Q}$, on a $\mathcal{T} = \mathcal{T} \cup \left\{p \overset{\beta}{\mapsto} z\right\}$.
   Ensuite, $\forall \texttt{z} \overset{\alpha}{\mapsto} \texttt{s}' \in \mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^{\texttt{a}}$, on a $\mathcal{T} = \mathcal{T} \setminus \left\{\ \texttt{z} \overset{\alpha}{\mapsto} \texttt{s}'\ \right\} \cup \left\{\texttt{z} \overset{\alpha}{\mapsto} \texttt{p}\right\}$ avec $\texttt{z} \in \mathcal{Q}$.

- **Suppression d'un état** s : la suppression d'un état s de l'automate, est réalisée par le module SUPPRIMER. Elle consiste à supprimer l'état s et toute sa descendance, *i.e.* tous les états et transitions accessibles à partir de l'état s.
- **Transaction suivante** : quand tous les items α_i d'une transaction donnée (pour un client *cid*) ont été lus et insérés dans l'automate, le module SUIVANT est appelé. Ce module fonctionne en trois étapes :
  (i) réduction de l'ensemble des états atteints : $\mathcal{Q}_{\texttt{cid}} = \mathcal{Q}_{\texttt{cid}} \setminus \{\mathcal{Q}_{\texttt{cid}} \cap \mathcal{Q}^- \cup \{\texttt{q}_\emptyset\}\}$
  (ii) appel du module INSERER sur l'item '-'.
  (iii) mise à jour de l'ensemble des états atteints : $\mathcal{Q}_{\texttt{cid}} = \mathcal{Q}_{\texttt{cid}} \cup \{\ \texttt{q}_\emptyset\ \}$.

4.2.2 Exemple de construction & pseudo-code

Afin d'illustrer l'exécution de SPAMS, nous reprenons l'exemple de la table 1, traité sous forme de stream, représentée par le schéma de la figure 1. Nous conservons ici

toute la généralité du stream et ne présupposons pas d'ordonancement des transactions par Cid. Les schémas des figures 2, 3, 5, 6, 7 et 9 illustrent la lecture et l'insertion d'items (voir les explications de la section 4.2.1 pour les points sur l'initialisation et la lecture et l'insertion d'items). Les schémas des figures 4, 8 illustrent la fin du traitement de transactions (voir les explications de la section 4.2.1 pour la partie transaction suivante). Nous présentons à la figure 10 le pseudo-code de notre algorithme.

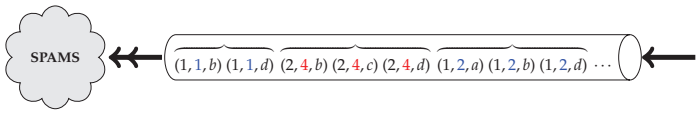
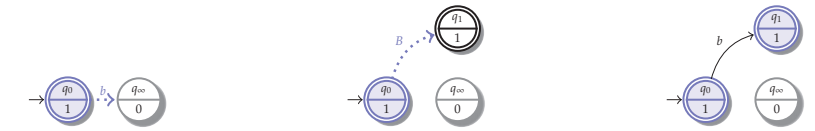
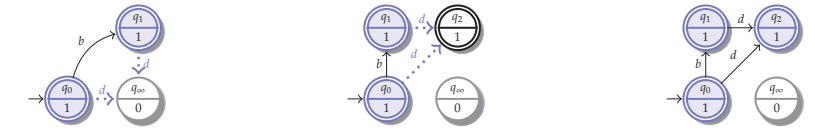


Fig. 1 – *Exemple de data stream non ordonné (voir table 1)*



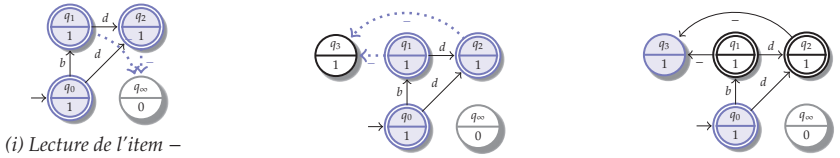
(i) Lecture de l'item b *(ii) Création de q_1 et déplacement des transitions atteintes* *(iii) Automate final*

Fig. 2 – *Lecture et insertion de l'item b (transaction 1)*



(i) Lecture de l'item d *(ii) Création de q_2 et déplacement des transitions atteintes* *(iii) Automate Final*

Fig. 3 – *Lecture et insertion de l'item d (transaction 1)*



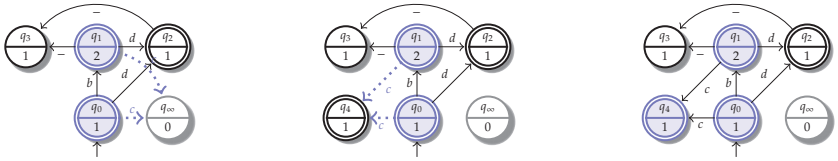
(i) Lecture de l'item – *(ii) Création de q_3 et déplacement des transitions atteintes* *(iii) Automate final*

Fig. 4 – *Fin du traitement de la transaction 1*

L'automate final comprend 21 états et 27 transitions indexant tous les motifs séquentiels de la table 1. Par manque de place, nous ne représentons pas cet automate.

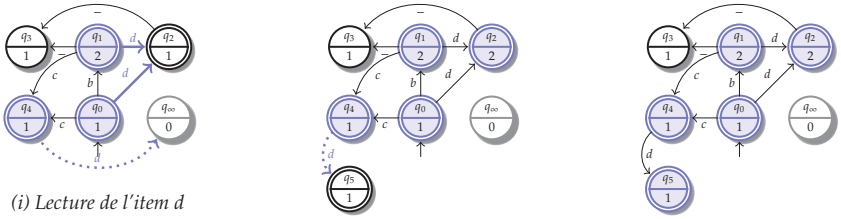
(i) Lecture de l'item b *(ii) Automate final*

FIG. 5 – *Lecture et insertion de l'item b (transaction 2)*



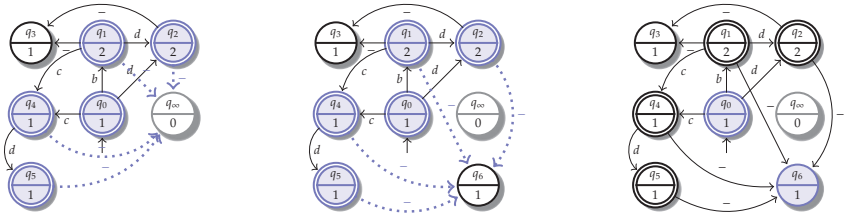
(i) Lecture de l'item c *(ii) Création de q_4 et déplacement des transitions atteintes* *(iii) Automate final*

FIG. 6 – *Lecture et insertion de l'item c (transaction 2)*



(i) Lecture de l'item d *(ii) Création de q_5 et déplacement des transitions atteintes* *(iii) Automate final*

FIG. 7 – *Lecture et insertion de l'item d (transaction 2)*

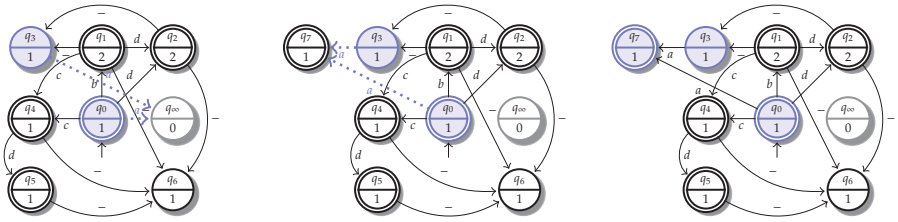


(i) Lecture de l'item – *(ii) Création de q_6 et déplacement des transitions atteintes* *(iii) Automate final*

FIG. 8 – *Fin de traitement de la transaction 2*

5 Expérimentations

Plusieurs expérimentations ont été réalisées afin de tester l'efficacité de notre approche. Des expérimentations empiriques ont été faites sur des jeux de données synté-



(i) Lecture de l'item a *(ii) Création de q_7 et déplacement des transitions atteintes* *(iii) Automate final*

Fig. 9 – *Lecture et insertion de l'item a (transaction 3)*

tiques générés à partir du simulateur IBM[2]. Nous avons fait varier le nombre de clients D, le nombre de transactions par client C, le nombre moyen d'items par transaction T, la taille moyenne des séquences maximales S et la taille moyenne des transactions des séquences maximales I. Nous avons utilisé une implémentation en C++ de SPAMS compilée avec l'option -03 du compilateur g++, sur un *Intel Pentium D*, *2Go* ram.
Nous illustrons sur les figures 11-(i) et 11-(ii) les performances en temps et en consommation mémoire de SPAMS, pour différentes valeurs de support, sur les jeux de données de petite, moyenne et grande taille, respectivement D7C7T7S7I7, D8C10T10S10I10 et D18C18T18S18I18. Les figures 11-(iii) à 11-(v) représentent l'évolution du temps, de la mémoire et du nombre de Cid en fonction du nombre de transactions sur D18C18T18S18I18 avec un support fixé à 20%.
La figure 11-(vi) permet d'illustrer sur le jeu de données D18C18T18S18I18 que le support statistique utilisé tend vers le support seuil au fur et à mesure de l'acquisition des transactions, diminuant ainsi l'ensemble des motifs $(\theta - \epsilon)$-fréquents de la couverture statistique. Pour le calcul de ϵ (voir section 3), nous avons choisi la valeur de 0.01 pour le risque statistique δ. Ces expérimentations montrent que nous trouvons un compromis très satisfaisant entre les performances temporelles, la consommation mémoire et la qualité des résultats de l'extraction tant en précision qu'en rappel. Elles montrent aussi, l'applicabilité et le passage à l'échelle de l'algorithme SPAMS.

6 Conclusion

Dans cet article, nous apportons une contribution originale en proposant un nouvel algorithme une-passe : SPAMS, basé sur l'élaboration d'un nouvel automate nommé SPA, qui permet de traiter de façon efficace la problématique de l'extraction des motifs séquentiels fréquents dans les data streams. SPA présente des propriétés incrémentales qui permettent son initialisation sans aucune information. Sa mise à jour, dans le cas des data streams, se fait avec une granularité très fine par transaction, sans qu'il soit nécessaire de procéder à un pré-traitement pour regrouper les transactions par client Cid. Par ailleurs, chaque transaction est acquise item par item et nous ne pré-supposons au départ de l'algorithme ni la connaissance de l'alphabet des items ni

[2]simulateur disponible à l'adresse `http://www.almaden.ibm.com/cs/quest`

Algorithme 1 – SPAMS

données : *Stream*, θ
sortie : SPA_θ
début
  Créer les états q_0 et q_∞
  $\mathcal{T} \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \{ q_0, q_\infty \}$
  $|q_0| \leftarrow 1$; $|q_\infty| \leftarrow 0$; $\texttt{cid} \leftarrow \texttt{null}$
  $\texttt{tid} \leftarrow \texttt{null}$; $\mathcal{C} \leftarrow \emptyset$; $\texttt{min\_sup} \leftarrow 0$
  pour chaque $(\texttt{cid}', \texttt{tid}', \alpha) \in \textit{Stream}$ **faire**
    si $(\texttt{cid} \neq \texttt{cid}')$ ou $(\texttt{tid} \neq \texttt{tid}')$ alors
      SUIVANT(cid, cid')
      $\texttt{cid} \leftarrow \texttt{cid}'; \texttt{tid} \leftarrow \texttt{tid}'$
    INSERER(α, cid)
fin

Algorithme 2 – SUIVANT

données : cid, cid'
début
  si $\texttt{cid} \neq \texttt{null}$ **alors**
    INSERER('$-$', cid)
    $\mathcal{Q}_{\texttt{cid}} \leftarrow \mathcal{Q}_{\texttt{cid}} \cap \mathcal{Q}^- \cup \{ q_0 \}$
  si $\texttt{cid}' \notin \mathcal{C}$ **alors**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{ \texttt{cid}' \}$
    $\mathcal{Q}_{\texttt{cid}'} \leftarrow \{ q_0 \}$
    $\epsilon \leftarrow \sqrt{\frac{1}{2.|\mathcal{C}|} \ln \frac{|\mathcal{C}|}{\delta}}$
    $\texttt{min\_sup} \leftarrow \lceil (\theta - \epsilon) \times |\mathcal{C}| \rceil$
fin

Algorithme 3 – SUPPRIMER

données : s
début
  · Supprimer les transitions entrantes de s
  pour chaque $\texttt{s} \xrightarrow{\beta} \texttt{s}' \in \mathcal{T}$ **faire**
    SUPPRIMER(s')
  · Supprimer l'état s
fin

Algorithme 4 – INSERER

données : α, cid
début
  $\mathcal{T}^a \leftarrow \emptyset$
  pour chaque $\texttt{s} \in \mathcal{Q}_{\texttt{cid}}$ **faire**
    si $\exists\, \texttt{s}' \in \mathcal{Q} \mid \texttt{s} \xrightarrow{\alpha} \texttt{s}' \in \mathcal{T}$ **alors**
      $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \left\{ \texttt{s} \xrightarrow{\alpha} \texttt{s}' \right\}$
    sinon
      $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \left\{ \texttt{s} \xrightarrow{\alpha} \texttt{q}_\infty \right\}$
  $\mathcal{Q}' \leftarrow \left\{ \texttt{s}' \in \mathcal{Q} \mid \texttt{s} \xrightarrow{\alpha} \texttt{s}' \in \mathcal{T}^a \right\}$
  pour chaque $\texttt{s}' \in \mathcal{Q}'$ **faire**
    si $|\texttt{s}'| + 1 >= \texttt{min\_sup}$ **alors**
      si $\texttt{s}' \neq \texttt{q}_\infty$ et $|\mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^a| = |\mathcal{T}_{\texttt{s}'}|$ **alors**
        si $\alpha \notin \Sigma_{cid}$ **alors**
          $|\texttt{s}'| \leftarrow |\texttt{s}'| + 1$
          $\Sigma_{\texttt{cid}} \leftarrow \Sigma_{\texttt{cid}} \cup \{\alpha\}$
        sinon
          · Créér un état s''
          · $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\texttt{s}''\}$
          si $\alpha \notin \Sigma_{\texttt{cid}}$ or $\texttt{s}' = \texttt{q}_\infty$ **alors**
            $|\texttt{s}''| \leftarrow |\texttt{s}'| + 1$
            $\Sigma_{\texttt{cid}} \leftarrow \Sigma_{\texttt{cid}} \cup \{\alpha\}$
          · Dupliquer les transitions sortantes de s' sur s''
          · Déplacer les transitions $\mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^a$ de s' vers s''
      $\mathcal{T}^a \leftarrow \mathcal{T}^a \setminus \{\mathcal{T}_{\texttt{s}'} \cap \mathcal{T}^a\}$
      si $|\texttt{s}'| < \texttt{min\_sup}$ **alors** SUPPRIMER(s')
fin

FIG. 10 – *Pseudo-code de l'algorithme* SPAMS

la connaissance du nombre des clients du stream. Cette information est apprise incrémentalement, à la volée, au fur et à mesure de l'insertion des nouvelles transactions du stream. Les expérimentations montrent que nous trouvons un compromis satisfaisant, en limitant la phase de génération des candidats avec la couverture statistique, tout en conservant de bons résultats tant en rappel qu'en précision avec également de bonnes performances en temps et en mémoire. Plusieurs perspectives peuvent être envisagées à la suite de ce travail. Cela concerne, par exemple, la suppression incrémentale de motifs séquentiels dans l'automate, ou encore la représentation de motifs tels que les

motifs fermés et maximaux.

Références

Ayres, J., J. Gehrke, T. Yiu, et J. Flannick (2002). Sequential pattern mining using a bitmap representation. pp. 429–435. ACM Press.

Chang, J. H. et W. S. Lee (2005). Efficient mining method for retrieving sequential patterns over online data streams. *J. Inf. Sci. 31*(5), 420–432.

Garofalakis, M., J. Gehrke, et R. Rastogi (2002). Querying and mining data streams : you only get one look a tutorial. In *SIGMOD '02 : Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp. 635–635. ACM.

Hopcroft, J. E. et J. D. Ullman (1990). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman Publishing Co., Inc.

Kum, H., J. Pei, W.Wang, et D. Duncan (2002). Approxmap : Approximate mining of consensus sequential patterns.

Laur, P.-A., R. Nock, J.-É. Symphor, et P. Poncelet (2007). Mining Evolving Data Streams for Frequent Patterns. *Pattern Recognition 40*(2), 492–503.

Li, H. et H. Chen (2007). Graseq : A novel approximate mining approach of sequential patterns over data stream. In *ADMA*, pp. 401–411.

Marascu, A. et F. Masseglia (2006). Extraction de motifs séquentiels dans les flots de données d'usage du web. In *EGC*, pp. 627–638.

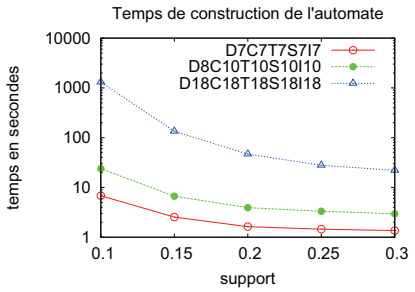
Pei, J., J. Han, B. Mortazavi-asl, H. Pinto, Q. Chen, U. Dayal, et M. chun Hsu (2001). Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. pp. 215–224.

Srikant, R. et R. Agrawal (1996). Mining sequential patterns : Generalizations and performance improvements. pp. 3–17.

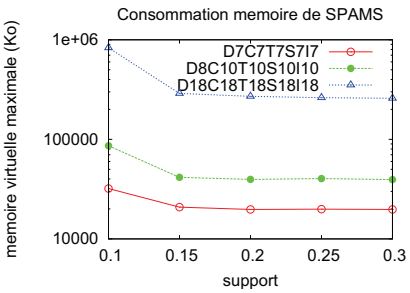
Zaki, M. J. (2001). Spade : an eficient algorithm for mining frequent sequences. In *Machine Learning Journal, special issue on Unsupervised Learning*, pp. 31–60.

Summary

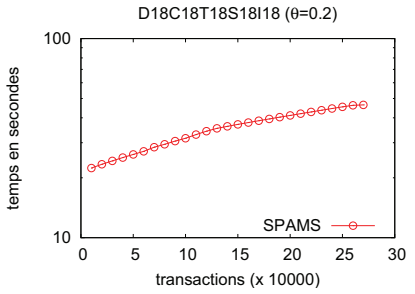
Mining sequential patterns on data streams is a new challenging problem for the datamining community since data arrives sequentially in the form of continuous rapid streams. More still than for databases, many additional constraints have to be considered due to the intrinsic nature of the streams. In this paper, we propose a new one-pass algorithm named: SPAMS, based on the incremental updating of an automaton structure: SPA, for mining sequential patterns in data streams. The information of the stream is learned progressively, from the insertion of new transactions, without preprocessing step a priori. The experimental results obtained show the relevance of the structure used, as well as the efficiency of our algorithm applied on datasets.



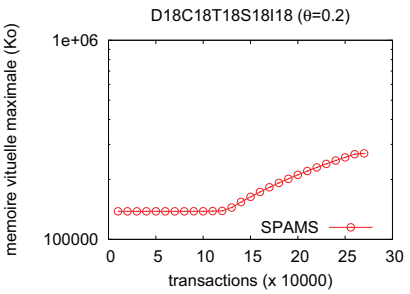
(i) temps



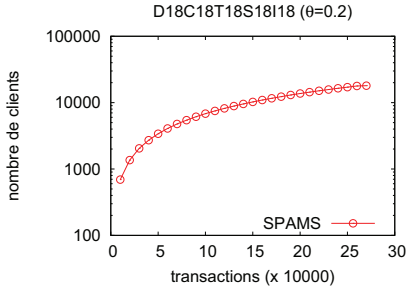
(ii) mémoire



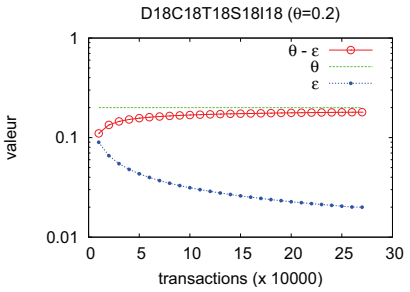
(iii) Temps sur D18C18T18S18I18



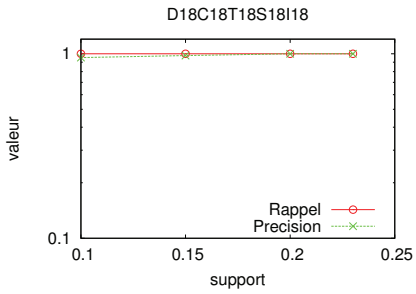
(iv) Mémoire sur D18C18T18S18I18



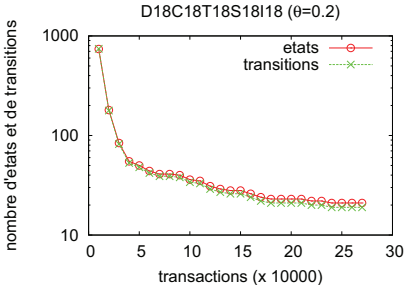
(v) Acquisition des clients sur D18C18T18S18I18



(vi) Évolution du support statistique sur D18C18T18S18I18



(vii) Rappel et précision sur D18C18T18S18I18



(viii) Nombre d'états et de transitions sur D18C18T18S18I18

FIG. 11 – *Expérimentations de SPAMS*