Echantillonnage pour l'extraction de motifs séquentiels : des bases de données statiques aux flots de données

Chedy Raïssi[*,**], Pascal Poncelet[**]

[*]LIRMM, 161 rue Ada,34392 Montpellier Cedex 5, France
raissi@lirmm.fr,
[**]EMA-LGI2P, Parc Scientifique Georges Besse, 30035 Nîmes Cedex, France
prénom.nom@ema.fr

Résumé. Depuis quelques années, la communauté fouille de données s'est intéressée à la problématique de l'extraction de motifs séquentiels à partir de grandes bases de données en considérant comme hypothèse que les données pouvaient être chargées en mémoire centrale. Cependant, cette hypothèse est mise en défaut lorsque les bases manipulées sont trop volumineuses. Dans cet article, nous étudions une technique d'échantillonnage basée sur des réservoirs et montrons comment cette dernière est particulièrement bien adaptée pour résumer de gros volumes de données. Nous nous intéressons ensuite à la problématique plus récente de la fouille sur des données disponibles sous la forme d'un flot continu et éventuellement infini ("*data stream*"). Nous étendons l'approche d'échantillonnage à ce nouveau contexte et montrons que nous sommes à même d'extraire des motifs séquentiels de flots tout en garantissant les taux d'erreurs sur les résultats. Les différentes expérimentations menées confirment nos résultats théoriques.

1 Introduction

La problématique de l'extraction de motifs séquentiels dans de grandes bases de données intéresse la communauté fouille de données depuis une dizaine d'années et différentes méthodes ont été développées pour extraire des séquences fréquentes. L'extraction de tels motifs est toutefois une tâche difficile car l'espace de recherche considéré est très grand. Afin de gérer au mieux cet espace de recherche, différentes stratégies ont été proposées. Les plus traditionnelles utilisent une approche à la *Apriori* Srikant et Agrawal (1996) et diffèrent principalement par les structures de données utilisées (vecteurs de bits, arbres préfixés, ...). Les approches les plus récentes considèrent, quant à elles, des projections multiples de la base de données selon le principe de *pattern-growth* proposé dans Pei et al. (2001) et évitent ainsi de générer des candidats. Outre ces différentes stratégies, les propositions les plus efficaces considèrent comme hypothèse que la base de données peut être chargée directement en mémoire centrale. Cependant, avec le développement des nouvelles technologies, ces dernières se trouvent de plus en plus mises en défaut dans la mesure où la quantité de données manipulées est trop volumineuse et qu'il devient irréaliste de stocker l'intégralité de la base en mémoire centrale.

Le développement des nouvelles technologies permet également de générer de très grands volumes de données issues de différentes sources : trafic TCP/IP, transactions financières, en-

registrements médicaux, capteurs. Les données apparaissent alors sous la forme d'un flot (*data stream*) de manière continue, à un rythme rapide et éventuellement de manière infinie. L'extraction de connaissances à partir de tels flots a récemment donné lieu à de nombreux travaux de recherche qui se sont focalisés sur la découverte d'itemsets fréquents (e.g., Chi et al. (2004); Giannella et al. (2003); Manku et Motwani (2002)) en utilisant des méthodes telles que le *landmark*, la *fenêtre glissante* ou les modèles de *pondérations temporelles* et peu de travaux se sont intéressés à l'extraction de motifs séquentiels dans les flots de données Raissi et al. (2006). Il est vrai qu'outre l'espace de recherche, les approches traditionnelles nécessitent de faire plusieurs passes sur la base ou de stocker cette dernière en mémoire.

Le reste de l'article est organisé de la manière suivante. Les travaux liés à l'extraction de motifs séquentiels sont présentés dans la section 2. Dans la section 3, nous introduisons plus formellement le problème et les concepts préliminaires de l'extraction de motifs. La section 4 présente l'échantillonnage dans le cadre d'une base de données statique avec les résultats théoriques sur la précision de l'échantillon et du seuil d'erreur. La section 5 étend ces résultats aux flots de données en présentant notre algorithme de maintien de synopsis. Les expérimentations sont décrites dans la section 6 et une conclusion est proposée dans la section 7.

2 Travaux antérieurs

Différentes approches efficaces d'extraction de séquences ont été proposées ces dernières années (e.g. PrefixSpan Pei et al. (2001), SPADE Zaki (2001), SPAM Ayres et al. (2002)). PrefixSpan est un algorithme basé sur la représentation de motifs au moyen d'un arbre préfixé qui dans son implémentation publique charge la base de données en mémoire centrale et utilise différentes projections afin d'éviter de générer des candidats. SPADE propose de transformer la base de données en une représentation verticale afin d'appliquer rapidement des opérations de jointure. Dans le cas de SPAM l'originalité réside dans la représentation des données sous la forme de bitmaps. Les expérimentations montrent que SPAM est plus efficace que PrefixSpan et SPADE sur de grandes bases de données. Cependant, ces trois algorithmes sont mis en défaut lorsqu'ils essayent de charger des bases trop volumineuses, de l'ordre de quelques gigaoctets.

Le volume de données important et croissant dans les bases de données statiques ou dans les flots de données impose de nouvelles contraintes à prendre en compte par les algorithmes d'extraction. Dans ce cas, il devient acceptable d'obtenir des réponses avec des *approximations* Aggarwal (2007). En d'autres termes, il devient indispensable de trouver un équilibre entre l'efficacité et la précision des résultats. De nombreuses structures de synopsis ont été développées ces dernières années comme les sketches, l'échantillonnage, les wavelets et les histogrammes. Toutes ces différentes structures possèdent les mêmes propriétés de grandes applicabilités (elles peuvent être utilisées pour répondre à divers problèmes), d'efficacité mémoire (elles sont capables de résumer de manière significative de grandes quantité de données) et de robustesse. En outre, toutes ces méthodes ne nécessitent qu'une passe sur la base de données ce qui les rend particulièrement utiles dans le cas des flots de données. Dans les travaux que nous avons menés, nous nous sommes focalisés sur les classes d'échantillonnage par réservoir dans la mesure où elles sont aisées à mettre en œuvre et garantissent un échantillon représentatif de bonne qualité. Cette approche a initialement été introduite dans Vitter (1985). Le principe est le suivant : nous maintenons un réservoir de taille n, les premiers n points dans l'ensemble des données sont stockés lors de l'étape d'initialisation. Lorsque le (t+1)^{ème}

Client	Date	Items
C_1	T_1	a, b, c, d
C_1	T_2	a, c
C_2	T_1	a, b
C_3	T_1	a, d
C_3	T_2	c

FIG. 1 – *Une base de données exemple $\mathcal{D}$*

point est traité, il remplace aléatoirement l'un des points du réservoir avec une probabilité de $\frac{n}{t+1}$. Ainsi, plus la taille de l'ensemble de données augmente plus la probabilité d'inclusion se réduit. Ceci est bien entendu un inconvénient majeur dans le cas des flots de données qui considèrent souvent que les informations les plus récentes dans le flot sont les plus pertinentes Giannella et al. (2003). Une solution pour répondre à ce problème est d'utiliser des fonctions biaisées pour réguler l'échantillon du flot de données Aggarwal (2006).

3 Concepts préliminaires

Dans cette section nous présentons la problématique de l'extraction des motifs séquentiels Srikant et Agrawal (1996); Srikant (1995) ainsi que le principe d'échantillonnage par réservoir biaisé.

Une transaction constitue, pour un client C, l'ensemble des items achetés par C à une même date. Une transaction s'écrit sous la forme d'un ensemble d'un triplet (id-client, id-date, itemset) où itemset est un ensemble d'items. Une séquence est une liste ordonnée, non vide, d'itemsets notée $\langle s_1 s_2 ... s_n \rangle$ o$\grave{u}$ s_j est un itemset. Une séquence de données est une séquence représentant les achats d'un client. Soit $T_1, T_2, ..., T_n$ les transactions d'un client, ordonnées par dates d'achat croissantes et soit itemset(T_i) l'ensemble des items correspondant à T_i, alors la séquence de données de ce client est $\langle\ itemset(T_1)\ itemset(T_2)\ ...\ itemset(T_n)\rangle$.

Un client supporte une séquence S si S est incluse dans la séquence de données de ce client. Le support d'une séquence S dans $\mathcal{D}$, noté $Support(S, \mathcal{D})$, représente le pourcentage des clients qui supportent S. Soit σ un seuil de support minimal défini par l'utilisateur, le problème de l'extraction des motifs séquentiels consiste à extraire toutes les séquences S dans $\mathcal{D}$ telles que $Support(S, \mathcal{D}) \geq \sigma$.

Exemple 1 *Considérons la base de données $\mathcal{D}$ de la figure 1. Avec un support minimum de $\sigma = \frac{2}{3}$, les séquences fréquentes dans $\mathcal{D}$ sont les suivantes :* $\langle(a)\rangle$, $\langle(b)\rangle$, $\langle(c)\rangle$, $\langle(d)\rangle$, $\langle(a\ b)\rangle$, $\langle(a\ d)\rangle$, $\langle(a)\ (c)\rangle$ *et* $\langle(d)\ (c)\rangle$.

L'échantillonnage par réservoir biaisé fut introduit dans Aggarwal (2006). L'idée principale est de réguler l'introduction des points dans le réservoir et ce afin de maîtriser la *fraîcheur* de l'échantillon produit. En d'autres termes, la fonction de biais permet de moduler l'échantillon de manière à se focaliser sur des comportements récents ou anciens en fonction des contraintes de l'application. La fonction de biais est définie de la manière suivante : $f(r,t) = e^{\lambda(t-r)}$ où le paramètre λ correspond au taux de biais. Cette fonction est proportionnelle à $p(r,t)$ avec $r < t$ qui est la probabilité qu'un point introduit dans le réservoir à l'instant r soit encore présent à l'instant t. En outre, l'inclusion d'une fonction de biais exponentielle rend possible l'utilisation d'algorithmes de remplacement simples et surtout cette classe spéciale de fonctions de biais

implique aussi une borne supérieure sur la taille du réservoir qui est indépendante de la longueur du flot. Pour un flot de longueur t, soit $R(t)$ la taille maximale du réservoir qui satisfait la fonction de biais exponentielle, nous avons $R(t) \leq \frac{1}{\lambda}$.

4 Echantillonnage et base de données statique

Etant donné que l'un des facteurs clés pour l'extraction est la taille de la base considérée, l'intuition sous-jacente est que l'algorithme d'extraction pourrait être lancé sur un échantillon de la base de données originale afin d'avoir des résultats de manière plus rapide et plus facile.

La première question à laquelle nous devons répondre si nous voulons extraire des motifs à partir d'un échantillon est : *à quel point l'échantillon est-il pertinent par rapport au jeu de données original ?* Nous répondons à cette question en exhibant une garantie sur le taux d'erreur du support d'une séquence. Notons qu'une approche similaire a été proposée pour l'extraction d'itemsets fréquents dans Toivonen (1996).

Définition 1 (Taux d'erreur) *Soit $\mathcal{D}$ une base de données transactionnelle de clients et posons $\mathcal{S}_\mathcal{D}$ l'échantillon aléatoire généré à partir de $\mathcal{D}$. Soit s une séquence présente dans $\mathcal{D}$. Le taux d'erreur absolu en terme d'estimation du support, noté $e(s, \mathcal{S}_\mathcal{D})$, est défini tel que :*

$$e(s, \mathcal{S}_\mathcal{D}) = |Support(s, \mathcal{S}_\mathcal{D}) - Support(s, \mathcal{D})|$$

Posons $X_{i,s}$ une variable aléatoire indépendante définie telle que :

$$\begin{cases} Pr[X_{i,s} = 1] = p_i & \text{si le } i^{eme} \text{ client supporte la séquence } s, \\ Pr[X_{i,s} = 0] = 1 - p_i & \text{sinon.} \end{cases}$$

et $X(s, \mathcal{S}_\mathcal{D}) = \sum_i^{|\mathcal{S}_\mathcal{D}|} X_{i,s}$.

$X(s, \mathcal{S}_\mathcal{D})$ représente le nombre de clients supportant la séquence s présents dans l'échantillon $\mathcal{S}_\mathcal{D}$. Ces clients peuvent être réécrits de la manière suivante : $X(s, \mathcal{S}_\mathcal{D}) = Support(s, \mathcal{S}_\mathcal{D}).|\mathcal{S}_\mathcal{D}|$. De même, l'espérance de la variable $X(s, \mathcal{S}_\mathcal{D})$ est $E[X(s, \mathcal{S}_\mathcal{D})] = Support(s, \mathcal{D}).|\mathcal{S}_\mathcal{D}|$.

Nous voulons estimer la probabilité que le taux d'erreur $e(s, \mathcal{S}_\mathcal{D})$ dépasse le seuil ε défini par l'utilisateur, $Pr[e(s, \mathcal{S}_\mathcal{D}) > \varepsilon]$.

Pour répondre à cette estimation, nous utilisons une méthode connue en statistiques : les inégalités de concentration (et plus précisément les inégalités de Hoeffding ; Hoeffding (1963)) qui permettent de borner la valeur réelle d'une variable aléatoire par rapport à son espérance et un terme d'erreur. Le théorème suivant exhibe une borne inférieure sur la taille de l'échantillon (ou réservoir) :

Théorème 1 *Soit s une séquence et $|\mathcal{S}_\mathcal{D}|$ la taille du réservoir, alors*

$$Pr[e(s, \mathcal{S}_\mathcal{D}) > \varepsilon] \leq \delta \; si \; |\mathcal{S}_\mathcal{D}| \geq \ln(\frac{2}{\delta})\frac{1}{2\varepsilon^2}$$

Preuve 1

$$\begin{aligned} Pr[e(s, \mathcal{S}_\mathcal{D}) > \varepsilon] &= Pr[\ |Support(s, \mathcal{S}_\mathcal{D}) - Support(s, \mathcal{D})| > \varepsilon] \\ &= Pr[\ |Support(s, \mathcal{S}_\mathcal{D}).|\mathcal{S}_\mathcal{D}| - Support(s, \mathcal{D}).|\mathcal{S}_\mathcal{D}|\ | > \varepsilon.|\mathcal{S}_\mathcal{D}|] \qquad (1) \\ &= Pr[\ |X(s, \mathcal{S}_\mathcal{D}) - E[X(s, \mathcal{S}_\mathcal{D})]\ | \ > \varepsilon.|\mathcal{S}_\mathcal{D}|] \end{aligned}$$

L'inégalité de Hoeffding énonce que pour n variables aléatoires indépendantes, $X_1, X_2, \ldots, X_n$ tel que $X_i \in [a, b]$, alors pour tout $t > 0$: $Pr[S - E[S] \geq nt] \leq exp\left(\frac{-2n^2t^2}{n(b-a)^2}\right)$ et $Pr[\ |S - E[S]\ | \geq nt] \leq 2exp\left(\frac{-2n^2t^2}{n(b-a)^2}\right)$ avec S la somme des variables X_i, d'où

$$Pr[\ |X(s, \mathcal{S}_\mathcal{D}) - E[X(s, \mathcal{S}_\mathcal{D})]\ |\ > \varepsilon.|\mathcal{S}_\mathcal{D}|\ ] \leq \delta \text{ avec } \delta = 2e^{-2\varepsilon^2.|\mathcal{S}_\mathcal{D}|} \qquad \square$$

Il est à noter que les inégalités de Hoeffding sont souvent considérées comme plus générales et moins précises que les inégalités de concentration de Chernoff, mais le choix de variables aléatoires indépendantes dans l'intervalle $[0, 1]$ donne des résultats similaires pour ces deux inégalités. De plus, il est à remarquer que la taille de l'échantillon $\mathcal{S}_\mathcal{D}$ est indépendante de la taille de la base de données initiale et reste conditionnée uniquement par ε et δ.

La table 1 illustre quelques exemples de tailles d'échantillons en termes de clients pour différentes valeurs de ε et δ. Il est à noter que lorsque les valeurs de ε et δ sont trop strictes, l'échantillon peut atteindre une taille assez importante.

ε	δ	$\lvert\mathcal{S}_\mathcal{D}\rvert$
0.01	0.01	26492
0.01	0.001	38005
0.01	0.0001	49518
0.001	0.0025	3333333

TAB. 1 – *Différentes tailles d'échantillons pour ε et δ donnés*

5 Motifs séquentiels, échantillonnage et flots de données

Un des problèmes majeurs qui rend la pratique de l'échantillonnage difficile sur les flots est que l'on ne sait pas à l'avance la taille du flot. Il faut donc développer des algorithmes d'échantillonnages dynamiques qui prennent en compte l'évolution et les changements dans la distribution des données transitant sur le flot. Dans cette section, nous étendons les résultats précédents et présentons un algorithme de maintien d'échantillon (biaisé ou non) de manière dynamique et qui prend en compte les différentes évolutions du flot de données. L'algorithme présenté peut être vu comme une étape de pré-traitement nécessaire afin de permettre l'extraction de séquences fréquentes. Afin que cette étape de pré-traitement soit pertinente, elle doit respecter les conditions suivantes : (i) L'échantillon doit avoir une borne inférieure sur sa taille afin de minimiser le taux d'erreur absolu en terme d'estimation du support. (ii) A cause de la nature même des séquences, les opérations d'insertions et d'enlèvements, nécessaires pour la mise à jour de l'échantillon, doivent se faire au niveau des clients, mais aussi de leurs itemsets. Pour s'en convaincre, il suffit de considérer un ensemble fini de clients avec pour chacun d'eux un grand nombre d'itemsets qui se rajoutent à chaque instant t. Cet ensemble ne peut bien sûr pas être considéré comme un échantillon ou un réservoir car il n'est pas borné et ne fait qu'augmenter avec le flot.

Dans notre modèle de flot de données, un point de données apparaissant à chaque instant t est défini comme un couple constitué d'un identifiant de client et d'une transaction.

Partant de ces contraintes et des résultats théoriques obtenus dans la section 4, nous proposons un algorithme simple de remplacement issu de l'approche de réservoir biaisé proposé dans Aggarwal (2006) qui permet de réguler l'échantillonnage des transactions des clients sur le flot grâce à une fonction de biaisage temporelle exponentielle. L'idée est la suivante : nous commençons avec un réservoir vide, de capacité maximale $\frac{1}{\lambda}$ (nous discutons un peu plus tard de la valeur du taux de biais λ) et chaque itemset d'un client apparaissant sur le flot est inséré de manière probabiliste dans le réservoir après une opération de *lancer de pièce* : soit par un remplacement des itemsets d'un client déjà présent dans le réservoir, soit par un ajout direct dans une des places encore vacantes. Comme discuté précédemment, nous devons aussi bien contrôler la taille du réservoir en terme de nombre de clients qu'en nombre d'itemsets. Cette opération de contrôle est appliquée grâce à une approche de *fenêtre glissante* (e.g. Aggarwal (2007), Babu et Widom (2001)) qui permet de garder uniquement les itemsets les plus récents pour un client donné dans le réservoir. Une fenêtre glissante peut être définie soit comme une fenêtre basée sur les séquences de taille k, contenant les k points les plus récents apparus sur le flot, soit comme une fenêtre basée sur un intervalle de temps de taille t contenant tous les points apparus sur le flot sur une durée de temps t. Dans notre approche nous utilisons des fenêtres glissantes basées sur des séquences afin de garder uniquement les transactions les plus récentes pour les clients présents dans l'échantillon. Ce type de fenêtre glissante permet l'extraction de séquences sur un horizon récent du flot. De plus, la fonction exponentielle de biais permet à l'utilisateur de choisir la taille de son réservoir (avec des contraintes sur l'(ε, δ)-approximation) et ainsi, un échantillon représentatif du flot peut être construit et mis à jour en mémoire selon les besoins de l'application et de l'utilisateur. Le corollaire suivant, issu du théorème 1 exhibe le lien qui existe entre le taux de biais λ et les seuils d'erreurs ε et δ :

Corollaire 1 *Soient λ le taux de bias, ε le seuil d'erreur et δ la probabilité maximale telle que $e(s, \mathcal{S}_\mathcal{D}) > \varepsilon$, alors :*

$$\lambda \leq \frac{2\varepsilon^2}{ln(2/\delta)}$$

Preuve 2 *D'après Aggarwal (2006), pour un flot de taille t, supposons $R(t)$ la taille maximale possible du réservoir qui satisfait la fonction de biaisage exponentielle, on a alors par définition : $R(t) \leq \frac{1}{\lambda}$. Nous considérons le réservoir entier comme l'échantillon cible pour l'extraction de motifs, on a donc : $R(t) = |\mathcal{S}_\mathcal{D}|$. Par substitution dans le théorème 1, nous avons le résultat énoncé.* □

La table 2 montre quelques valeurs du taux de biaisage λ et la taille minimale du réservoir nécessaire pour la bonne approximation du support des séquences.

ε	δ	λ	$R(t)$
0.01	0.01	0.0000377	26492
0.01	0.001	0.00002631	38005
0.01	0.0001	0.00002019	49518

TAB. 2 – *Différents taux de biaisage λ pour différentes valeurs de ε et δ*

Le fonctionnement général de l'algorithme est le suivant :

1. A l'arrivée d'un itemset d'un client C_i, voir si le client est dans la liste noire, auquel cas ignorer cet itemset, sinon passer à l'étape 2.

2. Voir si le client C_i est déjà dans le réservoir, si oui, rajouter l'itemset dans la fenêtre et voir s'il y a un décalage nécessaire à faire sinon passer à l'étape 3.
3. Faire un lancer de pièce (tir aléatoire) en cas de succès remplacer un des clients déjà présent dans le réservoir de manière aléatoire (le client remplacé est alors mis dans la liste noire). En cas d'échec, rajouter le client et son itemset dans le réservoir sans rien remplacer.

La partie la plus importante dans l'algorithme est la gestion des décalages des fenêtres glissantes et la mise à jour de la liste noire. Un problème peut apparaître lors de l'étape de remplacement des clients présents dans le réservoir : nous devons détecter si un client était déjà présent dans le réservoir, car le réintroduire sans aucun test préalable rendrait les résultats de l'extraction des motifs séquentiels inconsistants avec la réalité du flot. Le problème d'inconsistance apparaît lorsqu'un client est remplacé par un autre et qu'il revient dans le réservoir à l'instant suivant. Au moment de l'extraction, ce client n'aura pas tous les itemsets qu'il aurait dû avoir dans la fenêtre glissante actuelle. Les points de certains clients doivent donc être ignorés. Mais d'un autre côté, ignorer des points sur le flot peut introduire un nouveau biais, puisque seuls les clients ayant remplacé d'autres clients seront présents dans l'échantillon. Afin de résoudre ce problème d'inconsistance entre les itemsets des différents clients nous introduisons un système de *liste noire* qui permet d'interdire l'échantillonnage à un certain nombre de clients indésirables. Cette liste noire n'est pas irréversible et est réactualisée à chaque glissement de la fenêtre.

Algorithme 1 : Algorithme d'échantillonnage par réservoir pour l'extraction de motifs séquentiels

Data : Réservoir $\mathcal{S}_\mathcal{D}$;Taille du réservoir : n, Nombre de clients déjà présents dans le réservoir q ;Taille de la fenêtre glissante $|W|$; point T
Result : Réservoir mis à jour après le traitement du point T

```
1  begin
2      // BL : la liste noire
3      if T.C_i ∉ BL then
4          if T.C_i ∉ S_D then
5              // Insertion avec lancer
6              Coin ← Random(0,1);
7              if Coin ≤ q/n then
8                  pos ← Random(0,q) ; Replace(C_pos, T.C_i) ; BL.add(T.C_pos);
9              else
10                 Add(T, S_D) ;q++;
11         else
12             //Insertion directe
13             Insert(C_i,T);
14             if C_i.window.size > |W| then
15                 slideAllWindows();
16                 Update(BL);
17 end
```

Notons que cet algorithme peut être très facilement implémenté. Cependant, dans la section 5 nous avons supposé implicitement que l'algorithme 1 construisait un réservoir biaisé respectant la fonction de biaisage exponentielle avec le paramètre λ. Nous allons maintenant présenter une preuve formelle de cette proposition. Comme dans Aggarwal (2006), nous allons démontrer que la politique de remplacement appliquée dans notre algorithme permet de construire un réservoir biaisé de taille $|\mathcal{S}_\mathcal{D}| = n$ avec $\lambda = \frac{1}{n}$.

Proposition 1 *L'algorithme 1 construit un réservoir biaisé respectant la fonction de biaisage temporel $f(r,t) = e^{-\lambda(t-r)}$ avec $\lambda = \frac{1}{\mathcal{S}_\mathcal{D}}$.*

Preuve 3 *Soient K le nombre total de clients, q_t le nombre de points déjà insérés dans le réservoir à l'instant t et b_t la taille de la liste noire à l'instant t. Posons $n = |\mathcal{S}_\mathcal{D}|$, le nombre possible de points dans l'échantillon.*

La probabilité que le client arrivant sur le flot soit dans la liste noire est $\frac{b_i}{K}$ et la probabilité que le client soit déjà dans le réservoir est $\frac{q_i}{K}$. De plus, la probabilité que le lancer de pièce soit un succès après le test d'appartenance à la liste noire est $\frac{q_i}{n}$, la probabilité dans le cas d'un lancer fructueux est $\frac{1}{q_i}$ donc la probabilité qu'un point dans le réservoir soit éjecté à l'instant donné i est $\frac{q_i}{n} \times \frac{1}{q_i} = \frac{1}{n}$ (la probabilité qu'un point soit encore à l'instant donné i dans le réservoir est alors $(1 - \frac{1}{n})$, étape 2 de l'algorithme). De plus, sachant que $r < t$ nous devons calculer toutes les combinaisons possibles (r,t) avec t fixé de la probabilité qu'un client inséré à l'instant r soit encore dans le réservoir à l'instant t :

$$\prod_{i=r}^{t}\left(\frac{b_i}{K} + (1-\frac{b_i}{K})\left[\frac{q_i}{K} + (1-\frac{q_i}{K})(1-\frac{1}{n})\right]\right) = \prod_{i=r}^{t}\left(1 + \frac{b_i + q_i}{K.n} - \frac{b_i.q_i}{K^2.n} - \frac{1}{n}\right)$$
$$= \prod_{i=r}^{t}\left(1 + \frac{q_i}{K.n} + \frac{b_i.(K-q_i)}{K^2.n} - \frac{1}{n}\right)$$

Pour de très grandes valeurs de K, $\frac{q_i}{K.n}$ est à peu près égal à 0 et $K - q_i \simeq K$, donc $\frac{b_i.(K-q_i)}{K^2.n} \simeq 0$. D'ou,

$$\simeq \prod_{i=r}^{t}\left(1 - \frac{1}{n}\right) = \left[\left(1 - \frac{1}{n}\right)^n\right]^{(t-r)/n} \quad (2)$$

Pour de grandes valeurs de n, $\left(1 - \frac{1}{n}\right)$ est approximativement égal à e^{-1}. Par substitution dans l'équation 2, nous avons le résultat énoncé.

6 Expérimentations

Data Set	Items	Taille moy. trans.	Trans. par Client	# Clients	Taille (Go)
CL1MTR2.5SL50IT10K	10000	2.5	50	1000000	1.05
CL1MTR10SL20IT10K	10000	10	20	1000000	0.797
CL0.5MTR20SL20IT10K	10000	20	20	500000	0.767
CL6MTR2.5SL10IT20K	20000	2.5	10	6000000	0.686

TAB. 3 – *Différents jeux de données synthétiques utilisés pour les experimentations*

Nom	# de motifs seq. extraits	Support	Traitement	Mémoire ut.
CL1MTR10SL20IT10K	715	7%	537.165 s.	578.587 Mo
CL0.5MTR20SL20IT10K	582	15%	538.560 s.	502.034 Mo
CL6MTR2.5SL10IT20K	5503	0.3%	523.969 s.	685.821 Mo

TAB. 4 – *Résultats de l'extraction de motifs sur les jeux de données statiques*

CL0.5MTR20SL20IT10K					
$\lvert\mathcal{S}_\mathcal{D}\rvert$	# de motifs. séq.	Erreurs	Temps requis	Mémoire requise	Taille de l'échantillon
25000	582	2	10.341 s.	25.128 MB	38 MB
38005	579	3	13.640 s.	38.18 MB	57.9 MB
50000	582	1	20.367 s.	50.419 MB	76.2 MB
100000	580	3	44.410 s.	100.001 MB	152.1 MB
200000	582	1	158.566 s.	199.823 MB	205.4 MB
CL1MTR10SL20IT10K					
$\lvert\mathcal{S}_\mathcal{D}\rvert$	# de motifs. séq.	Erreurs	Temps requis	Mémoire requise	Taille de l'échantillon
25000	704	14	4.878 s.	14.291 MB	19.9 Mo
38005	707	9	5.899 s.	21.752 MB	30.2 Mo
50000	716	3	7.592 s.	28.982 MB	39.9 Mo
100000	717	2	15.339 s.	57.984 MB	79.8 Mo
200000	715	1	36.265 s.	115.579 MB	159.2 Mo
CL6MTR2.5SL10IT20K					
$\lvert\mathcal{S}_\mathcal{D}\rvert$	# de motifs. séq.	Erreurs	Temps requis	Mémoire requise	Taille de l'échantillon
25000	5830	342	1.6 s.	3.070 MB	3 Mo
38005	5239	271	1.608 s.	4.503 MB	4.4 Mo
50000	5321	184	1.903 s.	5.997 MB	5.9 Mo
100000	5432	75	2.725 s.	11.831 MB	11.8 Mo
500000	5531	31	9.854 s.	58.947 MB	58.8 Mo

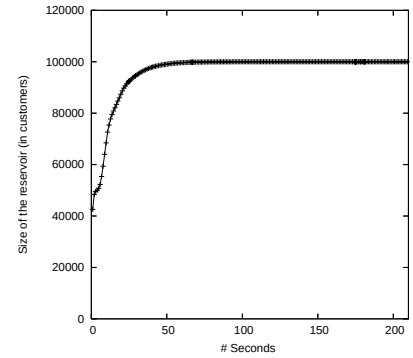
TAB. 5 – *Résultats d'extractions pour les différents jeux de données statiques*

Toutes les expérimentations ont été lancées sur un ordinateur MacBookPro Core-Duo 2.16 Ghz doté de 1Go de mémoire, tournant sous Mac OS X 10.4.6. Tous les algorithmes ont été codés dans le langage de programmation C++. Pour comparer nos résultats, nous utilisons une implémentation de PrefixSpan[1] Pei et al. (2001), qui permet l'extraction des motifs séquentiels. Les tests ont été lancés sur plusieurs jeux de données synthétiques qui sont générés par le logiciel QUEST[1]. Les différents jeux de données utilisés dans ces expérimentations et leurs caractéristiques sont regroupés dans le tableau 3. Afin d'échantillonner les bases de données statiques nous avons implémenté l'algorithme d'échantillonnage par réservoir de Vitter (1985).

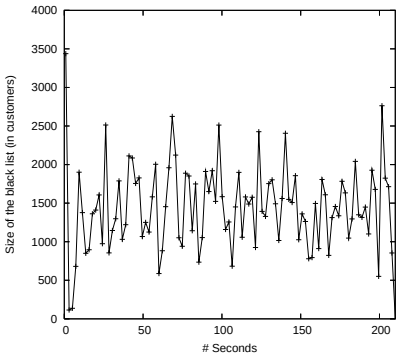
Nous avons testé la pertinence de nos calculs théoriques pour l'échantillonnage des bases de données statiques (section 4). Nous avons utilisé dans nos expérimentations les jeux de données CL6MTR2.5SL10IT20K, CL1MTR10SL20IT10K et CL0.5MTR20SL20IT10K afin de comparer les résultats aux estimations de l'(ε, δ)-approximation du théorème 1. Les résultats des différentes expérimentations sont listés dans les tableaux 4 et 5. Nous avons construit des échantillons de différentes tailles allant de 25000 à 500000 clients en utilisant l'approche d'échantillonnage par réservoir. Les expérimentations ont été répétées 5 fois pour chaque échantillon et les valeurs présentées sont les moyennes des résultats. La colonne *Erreur* dans

[1] http ://illimine.cs.uiuc.edu/

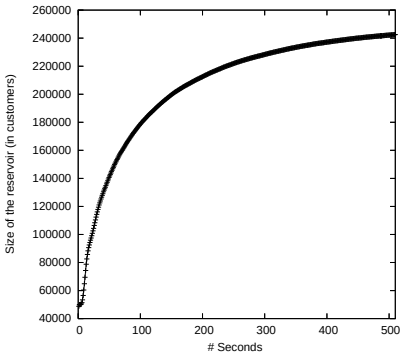
[1] http ://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html



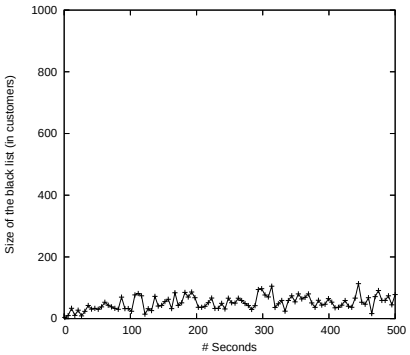
(a) Taille du réservoir en terme de clients



(b) Taille de la liste noire en terme de clients

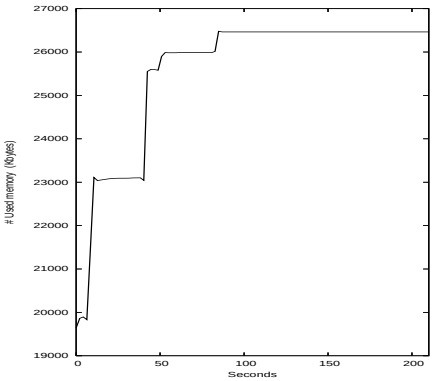


(c) Taille du réservoir en terme de clients

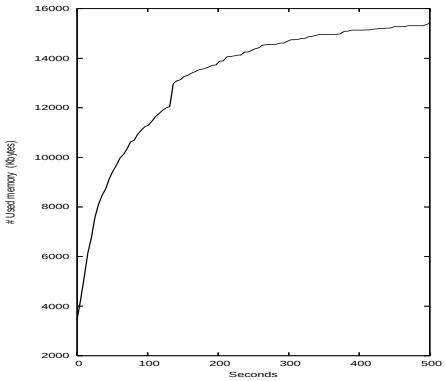


(d) Taille de la liste noire en terme de clients

FIG. 2 – *Expérimentations sur le jeu de données CL1MTR2.5SL50IT10K avec* λ*=0.00001,* $|W| = 10$ *et* λ*=0.00004,* $|W| = 2$*.*



(a) λ=0.00001 et $|W| = 10$



(b) λ=0.00004 et $|W| = 2$

FIG. 3 – *Besoin mémoire pour le jeu de données CL1MTR2.5SL50IT10K*

les tableaux décrit le nombre de séquences extraites dont le support dépasse nos (ε, δ)-approximations. On remarque que l'échantillonnage sur les bases de données statiques reste très précis, comme cela a été présenté dans les résultats théoriques de la section 4 et ce même avec de très petites tailles de résumés. Le temps d'extraction de motifs et l'utilisation mémoire pour ce processus est diminué de plusieurs ordres de magnitudes ce qui permet de pousser le processus global d'extraction vers des supports très bas.

L'expérimentation sur les flots de données utilise le jeu de données CL1MTR2.5SL50IT10K en faisant varier différents paramètres : la valeurs de biais (λ) et la taille de la fenêtre glissante. Dans ces expérimentations nous mettons en valeur l'efficacité de l'échantillonnage et montrons empiriquement que la liste noire reste bornée dans le temps. La figure 2 montre que la liste noire, qui est la garante de la consistence de l'échantillon pour le processus d'extraction de motifs reste assez limitée en terme d'espace mémoire utilisé et ce grâce aux différentes mises à jours faites à chaque décalage des fenêtres glissantes des clients présents dans le réservoir. De plus, si la taille de la fenêtre glissante est petite, la liste noire tend à être très petite aussi, ceci est du aux fréquents décalages qui permettent de mettre à jour les clients ignorés dans le réservoir. La figure 2 montre que le réservoir se remplit très vite, l'étape d'extraction de motifs séquentiels peut donc être lancé à partir de la 50^{eme} seconde. De plus, comme on peut le voir sur la figure 3, le réservoir reste borné et stable en terme d'occupation mémoire.

7 Conclusion

Dans cet article, nous nous sommes intéressés à de nouvelles techniques de résumés pour représenter des bases de données de motifs séquentiels. Nous avons montré qu'une approche basée sur des échantillons était tout à fait adaptée pour des bases de données statiques et que nous étions capables de maîtriser les taux d'erreur dans les résultats d'extraction de motifs séquentiels. A notre connaissance, ce travail est le premier à utiliser des techniques d'échantillonnage pour extraire des motifs séquentiels dans des bases de données. Nous avons également montré qu'une approche d'échantillonnage basée sur des réservoirs pouvait être adapté au contexte des flots de données et avons proposé un algorithme de remplacement des éléments du réservoir qui permet de réguler l'échantillonnage des transactions des clients sur le flot via une fonction de biaisage temporelle exponentielle.

Nous avons vu précédemment que les approches d'extraction de motifs sur les flots nécessitent, pour des raisons de capacités mémoire, de supprimer des connaissances acquises préalablement (par exemple, une séquence n'est plus fréquente sur un petit intervalle de temps). Au travers de notre approche, il n'est plus besoin d'exécuter l'algorithme d'extraction en continu mais plutôt à la demande pour fournir au décideur toute la connaissance extraite du flot en lui garantissant une marge d'erreur. Les perspectives associées à ces travaux sont nombreuses. Dans un premier temps, nous souhaitons offrir la possibilité de stocker les évolutions des données dans le réservoir et lors des étapes d'extraction de connaissances afin de déterminer les tendances dans le flot. Pour cela, nous souhaitons étendre la notion de *tilted-time window* introduite dans Giannella et al. (2003) pour ne stocker que les variations de fréquences et non plus toutes les fréquences des motifs agrégées sur un intervalle de temps. Dans un second temps, nous souhaitons intégrer cette approche dans un outil complet de suivi de flots de données permettant ainsi de pouvoir effectuer non seulement de l'extraction mais aussi des requêtes sur les données du flot.

Références

Aggarwal, C. (Ed.) (2007). *Data Streams : Models and Algorithms*. Springer.

Aggarwal, C. C. (2006). On biased reservoir sampling in the presence of stream evolution. In *Proc. of VLDB 06*, pp. 607–618.

Ayres, J., J. Flannick, J. Gehrke, et T. Yiu (2002). Sequential pattern mining using a bitmap representation. In *Proc. of KDD 02*, pp. 429–435.

Babu, S. et J. Widom (2001). Continuous queries over data streams. *SIGMOD Record 30*(3), 109–120.

Chi, Y., H. Wang, P. Yu, et R. Muntz (2004). Moment : Maintaining closed frequent itemsets over a stream sliding window. In *Proc. of ICDM 04*, pp. 59–66.

Giannella, G., J. Han, J. Pei, X. Yan, et P. Yu (2003). Mining frequent patterns in data streams at multiple time granularities. In *Next Generation Data Mining, MIT Press*.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association 58*(301), 13–30.

Manku, G. et R. Motwani (2002). Approximate frequency counts over data streams. In *Proc. of VLDB 02*, pp. 346–357.

Pei, J. et al. (2001). Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of ICDE 01*, pp. 215–224.

Raissi, C., P. Poncelet, et M. Teisseire (2006). Speed : Mining maximal sequential patterns over data streams. In *Proc. of IS 06*.

Srikant, R. et R. Agrawal (1996). Mining sequential patterns : Generalizations and performance improvements. In *Proc. of EDBT 96*, pp. 3–17.

Srikant, R. A. R. (1995). Mining sequential patterns. In *Proc. of ICDE 95*.

Toivonen, H. (1996). Sampling large databases for association rules. In *Proc. of VLDB 96*.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*(1), 37–57.

Zaki, M. J. (2001). Spade : An efficient algorithm for mining frequent sequences. *Machine Learning 42*(1/2), 31–60.

Summary

Sequential pattern mining is an active field in the domain of knowledge discovery. Since the database size is the most influential factor for mining algorithms we examine the use of sampling over static databases to get approximate mining results with an upper bound on the error rate. Moreover, we extend these sampling analysis and present an algorithm based on reservoir sampling to cope with sequential pattern mining over data streams. We demonstrate with empirical results that our sampling methods are efficient and that sequence mining remains accurate over static databases and data streams.