# Extraction de motifs séquentiels dans les flots de données d'usage du Web

Alice Marascu, Florent Masseglia

INRIA Sophia Antipolis, 2004 route des Lucioles - BP 93, 06902 Sophia Antipolis, France
{Alice.Marascu,Florent.Masseglia}@sophia.inria.fr

**Résumé.** Ces dernières années, de nouvelles contraintes sont apparues pour les techniques de fouille de données. Ces contraintes sont typiques d'un nouveau genre de données : les "*data streams*". Dans un processus de fouille appliqué sur un data stream, l'utilisation de la mémoire est limitée, de nouveaux éléments sont générés en permanence et doivent être traités le plus rapidement possible, aucun opérateur bloquant ne peut être appliqué sur les données et celles-ci ne peuvent être observées qu'une seule fois. A l'heure actuelle, la majorité des travaux relatifs à l'extraction de motifs dans les data streams ne concernent pas les motifs temporels. Nous montrons dans cet article que cela est principalement dû au phénomène combinatoire qui est lié à l'extraction de motifs séquentiels. Nous proposons alors un algorithme basé sur l'alignement de séquences pour extraire les motifs séquentiels dans les data streams. Afin de respecter la contrainte d'une passe unique sur les données, une heuristique gloutonne est proposée pour segmenter les séquences. Nous montrons enfin que notre proposition est capable d'extraire des motifs pertinents avec un support très faible.

## 1 Introduction

Le problème de l'extraction de motifs séquentiels dans un grand ensemble de données statiques a été largement étudié ces dernières années (Agrawal et Srikant (1995), Masseglia et al. (1998), Pei et al. (2001), Wang et Han (2004), Kum et al. (2003)). Les schémas extraits sont utiles dans de nombreuses applications comme le marketing, l'aide à la décision, l'analyse des usages, etc. Depuis peu, des applications émergentes comme (entre autres) l'analyse du trafic réseaux, la détection de fraude ou d'intrusion, la fouille de clickstream[1] ou encore l'analyse des données issues de capteurs ont introduits de nouveaux types de contraintes pour les méthodes de fouille. Ces applications ont donné lieu à une forme de données connues sous le nom de "data streams". Dans le contexte des data streams l'utilisation de la mémoire doit être réduite, les données sont générées de manière continue et très rapide, les opérations bloquantes ne sont pas envisageables et, enfin, les nouvelles données doivent être prises en compte aussi vite que possible. Ainsi, de nombreuses méthodes ont été proposées pour extraire des items ou des motifs dans les data streams (Datar et al. (2002), Chang et Lee (2003), Cormode et Muthukrishnan

[1] clickstream : flot de requêtes d'un utilisateur sur un site Web

(2005)). Dans ce domaine, l'approximation a rapidement été reconnue comme un facteur clé pour fournir des motifs à la vitesse imposée par l'application Garofalakis et al. (2002). Ensuite, des méthodes récentes (Chen et al. (2002), Giannella et al. (2003), Teng et al. (2003)) ont introduit différents principes pour gérer l'historique des motifs extraits. L'idée principale est que l'on est généralement plus interessé par les changements récents que par les changements plus anciens. Giannella et al. (2003) a ainsi introduit la notion de *logarithmic tilted time window* pour stocker les fréquences des motifs avec une granularité fine pour les changements récents et une granularité plus large pour les changements plus anciens. Dans Teng et al. (2003), une technique de regression est utilisée pour représenter les fréquences et une technique permettant de régler la finesse de représentation est introduite. Enfin, dans Raissi et al. (2005), les auteurs proposent une nouvelle structure de données destinée à extraire les motifs séquentiels fréquents d'un data stream. Cependant, dans cet article, nous montrons que les phénomènes combinatoires liés à l'extraction des motifs séquentiels rendent toute méthode exhaustive potentiellement bloquante. En effet, si dans le cas des règles d'association le nombre de possibilité est fini, ce n'est pas le cas des motifs séquentiels, pour lesquels un item peut se répéter à l'infini. Dans cet article, nous proposons l'algorithme SMDS (Sequence Mining in Data Streams) qui est basé sur l'alignement de séquences (comme Kum et al. (2003), Hay et al. (2002) l'ont déjà utilisée pour la fouille de bases de données statiques). SMDS est implémenté et a été testé sur des données réelles et des données synthétiques. Les données réelles sont issues du serveur Web de l'Inria Sophia Antipolis. Ces données collectent les informations sur les usages qui sont faits d'un site Web. Les techniques d'analyse de ces usages (WUM ou Web Usage Mining) fournissent des informations sur le comportement des utilisateurs du site. Parmi les méthodes existantes, celles qui extraient les motifs séquentiels sont particulièrement bien adaptées. Elles ont pour but l'extraction de motifs du genre : « *Sur le site de l'Inria, 10% des utilisateurs visitent la page d'accueil, puis la page des emplois, la page des offres d'emplois ITA, la description du métier d'ITA et enfin les annales des concours passés* ». Notre objectif est d'extraire ce type de comportement à partir des flots de données d'usage d'un site Web. Nous montrerons que SMDS satisfait les contraintes liés à la rapidité du data stream et peut être inclus dans un environnement temps réel.
Cet article est organisé de la manière suivante : tout d'abord nous présentons en section 2 les concepts d'extraction de motifs séquentiels et de Web Usage Mining. Ensuite nous proposons d'étudier un cadre de travail qui considère le data stream par batches et qui intègre l'extraction de motifs séquentiels dans la section 3. Enfin nous présentons nos expérimentations en section 4 avant de conclure cet article.

# 2 Définitions

## 2.1 Motifs séquentiels

Ce paragraphe expose et illustre la problématique liée à l'extraction de motifs séquentiels dans de grandes bases de données. Il reprend les différentes définitions proposées dans (Agrawal 1995). La notion de séquence est définie de la manière suivante :

**Définition 1** *Une transaction constitue, pour un client $C$, l'ensemble des items achetés par $C$ à une même date. Dans une base de données client, une transaction s'écrit sous forme d'un triplet : <id-client, id-date, itemset>. Un itemset est un ensemble non vide d'items noté*

$(i_1 i_2 \dots i_k)$ *où* $i_j$ *est un item (il s'agit de la représentation d'une transaction non datée). Une séquence est une liste ordonnée, non vide, d'itemsets notée* $< s_1 s_2 \dots s_n >$ *où* $s_j$ *est un itemset (une séquence est donc une suite de transactions avec une relation d'ordre entre les transactions). Une séquence de données est une séquence représentant les achats d'un client. Soit* $T_1, T_2, \dots, T_n$ *les transactions d'un client, ordonnées par date d'achat croissante et soit itemset(*$T_i$*) l'ensemble des items correspondants à* $T_i$*, alors la séquence de données de ce client est* $< itemset(T_1)\ itemset(T_2) \dots\ itemset(T_n) >$.

**Exemple 1** *Soit* $C$ *un client et* $S$=$< (3)\ (4\ 5)\ (8) >$*, la séquence de données représentant les achats de ce client.* $S$ *peut être interprétée par "*$C$ *a acheté l'item 3, puis en même temps les items 4 et 5 et enfin l'item 8".*

**Définition 2** *Le* $support$ *de* $s$*, noté* $supp(s)$*, est le pourcentage de toutes les séquences dans* $D$ *qui supportent (contiennent)* $s$*. Si* $supp(s) \geq minsupp$*, avec une valeur de support minimum* $minsupp$ *fixée par l'utilisateur, la séquence* $s$ *est dite fréquente.*

## 2.2 Adapter la problématique des motifs séquentiels

Pour les méthodes traditionnelles de Web Usage Mining, le principe gérénal est similaire à celui de Masseglia et al. (2000). Les données brutes sont collectées dans des fichiers logs par les serveurs. Chaque entrée dans le fichier log représente une requête faite par une machine cliente au serveur.

**Définition 3** Soit $Log$ un ensemble d'entrées dans le fichier access log. Une entrée $g$, $g \in Log$, es t un tuple $g =< ip_g, \{(l_1^g.URL, l_1^g.time), ..., (l_m^g.URL, l_m^g.time)\} >$ tel que pour $1 \leq k \leq m$, $l_k^g.URL$ représente l'objet demandé par le client $g$ à la date $l_k^g.time$, et pour tout $1 \leq j < k$, $l_k^g.time > l_j^g.time$.

La figure 1 illustre un exemple de fichier obtenu après la phase de pré-traiteme nt pour une classe de navigations. A chaque client correspond une suite de "dates" (événem ents) et la traduction de l'URL demandée par ce client à cette date.

| | Date1 | Date2 | Date3 | Date4 | Date5 |
|---|---|---|---|---|---|
| Client1 | 10 | 30 | 40 | 20 | 30 |
| Client2 | 10 | 30 | 60 | 20 | 50 |
| Client3 | 10 | 70 | 30 | 20 | 30 |

FIG. 1 – *Exemple de fichier résultat issu de la phase de pré-traitement*

L'objectif est alors de déterminer, grâce à une phase d'extraction, les séquences de ce jeu de données, qui peuvent être considérées comme fréquentes selon la définition 2. Les résultats obtenus sont du type `< ( 10 ) ( 30 ) ( 20 ) (30 ) >` (ici avec un support minimum de 66% et en appliquant les algorithmes de fouille de données sur le fichier représenté par la figure 1). Ce dernier résultat, une fois re-traduit en termes d'URLs, confirme la découverte d'un comportement commun à $minSup$ utilisateurs et fournit l'enchaînement des pages qui constituent ce comportement fréquent. Enfin, l'exploitation par l'utilisateur des résultats obtenus est facilitée par un outil de requête et de visualisation.

# 3 SMDS : motivation et principe général

Notre méthode est basée sur un environnement de découpage du data stream en “batches” (inspiré de celui proposé par Giannella et al. (2003)) et par la structure d’arbre préfixé de PSP Masseglia et al. (1998) pour gérer les séquences extraites. Nous proposons d’abord une étude des limites d’une approche intégrant un algorithme exhaustif d’extraction de motifs séquentiels. Nous présentons ensuite notre méthode, basée sur le principe de l’alignement de séquences.

## 3.1 Limites de l’extraction de motifs séquentiels

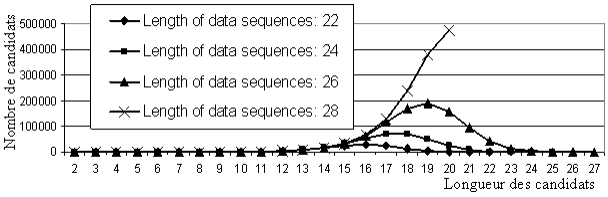


FIG. 2 – *Limites d’un environnement intégrant PSP*

Dans SMDS, le data stream est traité sous forme de batches de taille fixe. Soient $B_1, B_2,$ ... $B_n$, les batches et $B_n$, le batch courant. Le principe de SMDS est d’extraire les motifs sequentiels représentatifs de chaque batch $b$ de $[B_1..B_n]$ et de stocker les motifs extraits dans une structure d’arbre préfixé. Considérons que les motifs sont extraits par une méthode exhaustive (comme celles conçues pour les données statiques). Une telle méthode présentera au moins un type d’opération bloquante. Considérons par exemple le cas de l’algorithme PSP Masseglia et al. (1998). Nous avons testé cet algorithme sur des bases de données ne contenant que deux séquences ($s_1$ et $s_2$). Les deux séquences sont égales et contiennent des répétitions d’itemsets de taille 1. Plus précisément, la première base de test contenait onze répétitions des itemsets $(1)(2)$ (*i.e.* $s_1 =< (1)(2)(1)(2)...(1)(2) >$, longueur($s_1$)=22 et $s_2 = s_1$). Le nombre de candidats générés à chaque passe est reporté dans la figure 2. La figure 2 reporte aussi le nombre de candidats générés pour les bases contenant des séquences de longueur 24, 26 et 28. On peut observer que, pour la base contenant des séquences de longueur 28, PSP est incapable de fournir les résultats (la mémoire est saturée par le nombre de candidats). Nous avons fait la même observation pour l’algorithme prefixSpan[2] (Pei et al. (2001)) pour lequel ce cas de figure conduirait à un blocage du data stream. Dans le contexte des flots de données issus des usages d’un site Web, il n’est pas rare de trouver de nombreuses répétitions d’un ou plusieurs items (fichiers pdf, php, etc.).

[2]http ://www-sal.cs.uiuc.edu/-hanj/software/prefixspan.htm

| | | | | |
|---|---|---|---|---|
| Etape 1 : | | | | |
| $S_1$ : | <(a,c) | (e) | () | (m,n)> |
| $S_2$ : | <(a,d) | (e) | (h) | (m,n)> |
| $SA_{12}$ : | (a :2, c :1, d :1) :2 | (e :2) :2 | (h :1) :1 | (m :2, n :2) :2 |
| Etape 2 : | | | | |
| $SA_{12}$ : | (a :2, c :1, d :1) :2 | (e :2) :2 | (h :1) :1 | (m :2, n :2) :2 |
| $S_3$ : | <(a,b) | (e) | (i,j) | (m)> |
| $SA_{13}$ : | (a :3, b :1, c :1, d :1) :3 | (e :3) :3 | (h :1, i :1, j :1) :2 | (m :3, n :2) :3 |
| Etape 3 : | | | | |
| $SA_{13}$ : | (a :3, b :1, c :1, d :1) :3 | (e :3) :3 | (h :1, i :1, j :1) :2 | (m :3, n :2) :3 |
| $S_4$ : | <(b) | (e) | (h,i) | (m)> |
| $SA_{14}$ : | (a :3, b :2, c :1, d :1) :4 | (e :4) :4 | (h :2, i :2, j :1) :3 | (m :4, n :2) :4 |

FIG. 3 – *Etapes de l'alignement de séquences*

## 3.2 Principe général

Dans les grandes lignes, SMDS fonctionne de la manière suivante : classification de l'ensemble des séquences de chaque batch de transactions suivi d'un alignement pour chaque cluster ainsi obtenu. Cela permet d'obtenir des clusters de comportements qui représentent les usages du site en temps réel. Pour chaque cluster dont la taille est supérieure à $minSize$ (spécifié par l'utilisateur) SMDS ne stocke donc que le résumé du cluster. Ce résumé est donné par l'algorithme d'alignement de séquences appliqué sur chaque cluster.

**Algorithme glouton de classification des séquences**

Notre schéma classificatoire est basique. Il repose sur le fait que les navigations sur un site sont souvent : soit plutôt proches, soit très éloignées. De manière empirique, on peut constater que les utilisateurs qui demandent les pages concernant les offres d'emplois d'ITA ne vont probablement pas consulter (dans la même session) les pages relatives aux prochains séminaires organisés par l'unité de recherche de Sophia Antipolis. Dans le but d'obtenir une classification des navigations aussi rapide que possible, notre approche gloutonne fonctionne de la manière suivante : l'algorithme est initialisé avec une seule classe, qui contient la première navigation. Ensuite, pour chaque navigation $n$ dans le batch, $n$ est comparée avec chaque cluster $c$. Aussitôt que $n$ est similaire à une séquence de $c$ alors $n$ est insérée dans $c$. Si $n$ n'est insérée dans aucun cluster, alors un nouveau cluster est crée et $n$ est insérée dans ce nouveau cluster. La similitude entre deux séquences ($sim(s_1, s_2)$) est donnée dans la définition 4. $s$ est insérée dans $c$ si la condition suivante est respectée : $\exists s_c \in c / sim(s, s_c) \leq minSim$, avec $minSim$ la similitude minimum, spécifiée par l'utilisateur.

**Définition 4** Soient $s_1$ et $s_2$ deux motifs séquentiels. Soit $|LCS(s_1, s_2)|$ la longueur de la plus longue sous-séquence commune entre $s_1$ et $s_2$. La similitude $sim(s_1, s_2)$ entre $s_1$ et $s_2$ est définie de la manière suivante : $sim = \frac{2 \times |LCS(s_1, s_2)|}{longueur(s_1) + longueur(s_2)}$.

**Alignement de séquences**

L'algorithme de classification permet d'obtenir des classes de séquences similaires, ce qui est un élément clé pour l'alignement des séquences. L'alignement des séquences renvoie une séquence alignée du type : $SA =< I_1 : n_1, I_2 : n_2, ..., I_r, n_r >: m$. Dans cette représentation, $m$ représente le nombre total de séquences impliquées dans l'alignement. $I_p$ $(1 \leq p \leq r)$ est un itemset représentée sous la forme $(x_{i_1} : m_{i_1}, ...x_{i_t} : m_{i_t})$, où $m_{i_t}$ est le nombre de séquences qui contiennent l'item $x_i$ à la $p^{eme}$ position dans la séquence alignée. Enfin, $n_p$ est le nombre d'occurrences de l'itemset $I_p$ dans l'alignement. L'exemple 2 décrit le processus d'alignement de quatre séquences. À partir de deux séquences, l'alignement commence par insérer des itemsets vides (au début, au milieu ou à la fin des séquences) jusqu'à ce que les deux séquences contiennent le même nombre d'itemsets.

**Exemple 2** *considérons les séquences suivantes : $S_1 =<$ (a,c) (e) (m,n) $>$, $S_2 =<$ (a,d) (e) (h) (m,n) $>$, $S_3 =<$ (a,b) (e) (i,j) (m) $>$ et $S_4 =<$ (b) (e) (h,i) (m) $>$. Les étapes conduisant à l'alignement de ces séquences sont détaillées dans la figure 3. Tout d'abord, un itemset vide est inséré dans $S_1$. ensuite $S_1$ et $S_2$ sont alignées dans le but de produire $SA_{12}$. Le processus d'alignement est alors appliqué entre $SA_{12}$ et $S_3$. La méthode d'alignement continue à traiter les séquences deux par deux jusqu'à la dernière séquence.*

À la fin du processus d'alignement, la séquence alignée ($SA_{14}$ dans la figure 3) est un résumé du cluster correspondant. Le motif séquentiel correspondant peut être obtenu en spécifiant $k$ : le nombre minimum d'occurrences d'un item pour que celui-ci soit considéré comme fréquent. Par exemple, avec la séquence $SA_{14}$ de la figure 3 et $k = 2$, la séquence alignée filtrée sera : <(a,b)(e)(h,i)(m,n)> (ce qui correspond aux items qui ont un nombre d'occurrences supérieur ou égal à $k$).

**Stockage et gestion des séquences**

Les séquences alignées obtenues à la fin de l'étape précédente sont stockée dans un arbre préfixé similaire à celui de Masseglia et al. (1998). Si une nouvelle séquences $s$ est découverte, alors l'arbre est modifié pour stocker cette nouvelle séquence. Sinon, $s$ est déjà dans l'arbre et son support est mis à jour. La figure 4 donne un exemple d'arbre préfixé. Chaque chemin de la racine à un nœud de l'arbre représente une séquence extraite. L'arbre de la figure 4 contient 6 séquences (<(a c)>, <(a d)>, <(b)>, <(c d)>, <(c)(e)>, <(d)(a)>). Tout chemin de la racine à une feuille représente une séquence et le nœud de profondeur $l$ représente le $l^{eme}$ item de la séquence. Le changement d'itemset est représenté par des branches de différents types. Par exemple, le lien pointillé entre les nœuds $c$ et $e$ de la figure 4 illustre le fait que $e$ n'est pas dans le même itemset que $c$. À chaque nœud est associé $k$, le filtre utilisé pour obtenir cette séquence alignée dans le cluster correspondant. L'exemple 3 donne une illustration de la gestion des séquences et de leur support.

**Exemple 3** *Considérons la séquence $s_1 =<$(a)$>$ de la figure 4. Le support de la séquence alignée $s_1$ est 5 et le nœud $(a)$ possède deux fils. Cela signifie que $s_1$ a été extraite dans plus d'un cluster et que sont support est de 5. Considérons maintenant la séquence $s_2 =<$(d)(a)$>$. Le support de la séquence alignée $s_2$ est 3. Cela signifie que le cluster correspondant contient cette séquence avec un filtre $k = 3$.*

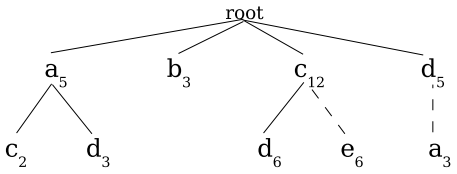


FIG. 4 – *Exemple d'arbre préfixé*

L'algorithme SMDS décrit dans cet article est le suivant :

**Algorithm 1** (SMDS)
**Input :** $B = \cup_{i=0}^{\infty} B_i$ : un ensemble infini de batches de transactions ; $minSize$ : la taille minimum des classes à synthétiser ; $minSim$ : la similitude minimum entre deux séquences pour qu'un cluster soit mis à jour ; $k$ : le filtre dans la méthode d'alignement.
**Output :** L'arbre préfixé mis à jour avec les séquences alignées.
**While** ($B$) **Do**
$b \leftarrow$ NextBatch() ;
$C \leftarrow$ Clustering($b$, $minSize$, $minSim$) ;// 1) Obtenir des cluster de taille $> minSize$
**Foreach** ($c \in C$) **Do** // 2) Résumer chaque cluster avec un filtre $k$ ;
$SA_c \leftarrow$ Alignment($c$, $k$) ;
**If** ($SA_c$) **Then** PrefixTree $\leftarrow$ PrefixTree+$SA_c$ ; //3) Stocker les séquences alignées
**Done**
// Séquences obsolètes
TailPruning(PrefixTree) ; // 4) Mettre à jour les Tilted Time Windows.
**Done** (fin Algorithm SMDS) ;

**Complexité**
Soit $n$ le nombre de séquences du batch. Dans le pire des cas, l'algorithme de classification a une complexité en temps de $O(n^2)$. En fait, dans le pire des cas, LCS est appliqué une fois pour la première séquence, deux fois pour la deuxième, et ainsi de suite. La complexité est donc $O(\frac{n.(n+1)}{2}) = O(n^2)$. Cependant, même si cette complexité peut être améliorée (pour le pire des cas) SMDS est très adapté pour les motifs de navigation sur le Web et nos expérimentations (C.f. section 4) sur des données réelles (access logs de l'Inria Sophia Antipolis) ont montré sont efficacité. En ce qui concerne l'alignement, considérons que toutes les séquences d'un cluster $C$ ont une longueur $m$ et $|C| = p$. La complexité de l'alignement pour un batch est de $O(p.m^2)$.

# 4 Expérimentations

La méthode SMDS a été implémentée en Java sur un Pentium (2,1 Ghz) exploité par un système Linux Fedora. Nous avons évalué notre proposition sur des données synthétiques et

des données réelles[3].

## 4.1 Temps de réponse et robustesse de SMDS

Dans le but de montrer l'efficacité de SMDS, nous reportons dans la figure 5 le temps nécessaire pour extraire les motifs séquentiels les plus longs sur chaque batch correspondant à des données d'usage du Web (à gauche de la figure 5) et des données synthétiques (à droite de la figure 5). Pour le site Web de l'Inria, les données ont été collectées sur une période de 14 mois et représentent 14 Go. Le nombre total de navigations est de 3,5 millions pour $300000$ navigations. Nous avons découpé le fichier log en batches de $4500$ transactions (soit environ 1500 séquences en moyenne). Pour ces expérimentations, le filtre $k$ est fixé à 30% (notons que ce filtre a un impact sur les temps d'exécution, dans la mesure où il modifie la taille des séquences à gérer dans l'arbre préfixé). De plus, nous avons "injecté" dans les données des séquences parasites. Le premier batch ne subit pas de modification. Dans le second, nous ajoutons dix séquences contenant deux répétitions de deux items (C.f. les séquences $s_1$ et $s_2$ décrites en section 3.1). Dans le troisième batch, nous ajoutons dix séquences de trois répétitions, et ainsi de suite jusqu'au trentième batch qui contient 10 séquences de trente répétitions. L'objectif est de montrer que les méthodes d'extraction traditionelles (PSP, prefixSpan, ...) risquent de bloquer le data stram alors que SMDS continuera sa tâche d'extraction. Nous pouvons observer que le temps de réponse de SMDS varie de 1800 ms à 3100 ms. PSP propose des motifs avec de très bonnes performances pour les premiers batches et se trouve pénalisé par le bruit ajouté par les séquences parasites (voir le batch 19). Le test a également été fait avec prefixSpan et le comportement exponentiel est similaire. Pour PSP comme pour prefixSpan, le support minimum spécifié était le maximum possible tout en assurant que les séquences "parasites" (répétitions) seraient trouvées. Nous avons ajouté à la figure 5 le nombre de séquences de chaque batch pour expliquer les différences de temps d'exécution d'un batch à un autre. On peut observer, par exemple, que le batch 1 contient 1500 séquences et que SMDS demande 2700 ms pour en extraire les motifs séquentiels. Pour les données synthétiques, nous avons généré des batches de $10000$ transactions (qui correspondent à environ 500 séquences en moyenne). La longueur moyenne des séquences était de 10 pour $200000$ items. Le filtre $k$ est fixé à environ 30%. Nous indiquons dans la figure 5 (partie droite) les temps de réponse et le nombre de séquences correspondant à chaque batch. Nous pouvons observer que SMDS traite $10000$ transactions en moins de 4 secondes (par exemple pour le batch 2).

## 4.2 Motifs extraits sur les données réelles

La liste des comportements découverts par SMDS couvre plus de 100 objectifs de navigation (classes de séquences de navigations) sur le site Web de l'Inria Sophia Antipolis. La plupart des motifs découverts peut être considérée comme rare (support faible) et pertinente (haute confiance, car le filtre utilisé est élevé). Nous reportons ici quelques exemples de ces comportements.
**A)** $k = 30\%$, taille de la classe = 13, préfixe="http ://www-sop.inria.fr/omega/" :
< (MC2QMC2004) (personnel/Denis.Talay/moi.html) (MC2QMC2004/presentation.html) (MC2QMC2004/dates.html) (MC2QMC2004/Call_for_papers.html)>

[3]Le générateur de données synthétiques peut être téléchargé sur http ://www.almaden.ibm.com/cs/quest.

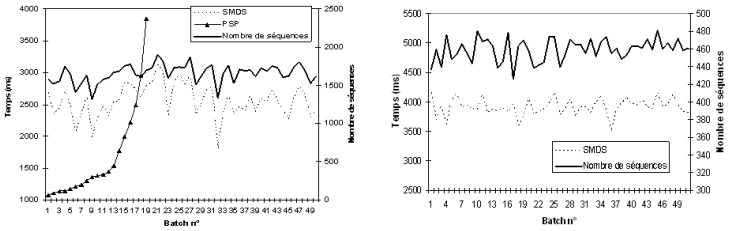


FIG. 5 – *Temps d'exécution de SMDS.*

*Cette séquence est fréquente en juin 2004, lors de l'organisation de la conférence MCQMC par une équipe de l'Inria Sophia Antipolis. Ce comportement était partagé par au plus 13 utilisateurs en même temps (taille de la classe = 13).*
**B)** $k = 30\%$, taille de la classe = 10, préfixe="http ://www-sop.inria.fr/acacia/personnel/itey /Francais/Cours/" : < (programmation-fra.html) (PDF/chapitre-cplus.pdf) (cours-programmation-fra.html) (programmation-fra.html) >
*Ce comportement correspond aux requête faites sur un document pédagogique sur la programmation écrit par un membre d'une équipe de l'Inria Sophia Antipolis. Il correspond à la période d'avril 2004.*

Pour les navigations sur le site de l'Inria Sophia Antipolis, nous avons également constaté que SMDS est capable de détecter les séquences parasites qui avaient été injectées dans les batches. Ces séquences sont simplement regroupées par SMDS dans un cluster qui ne contient qu'elles, sans impact sur les temps d'exécution.

## 4.3 Impact de la taille des batches

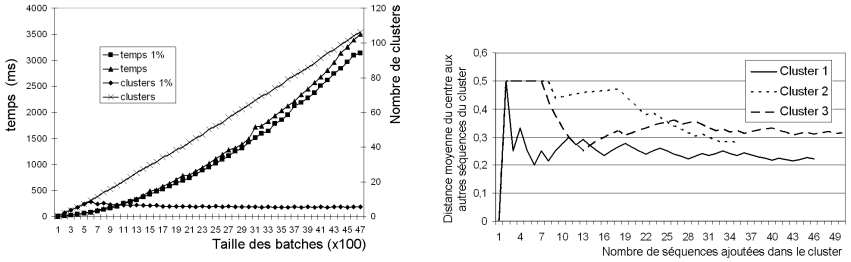


FIG. 6 – *Taille des batches et pire cluster, pas à pas.*

Puisque la complexité de SMDS dépend de la taille de batches, nous avons mené une étude concernant l'impact de cette taille sur les temps de réponse. Nous reportons dans la

figure 6 (partie gauche) les temps de réponses quand $S$ (le nombre de séquences du batch) varie de 100 à 5000 séquences. La courbe $temps$ représente les temps d'exécution, $cluster$ représente les nombre de clusters extraits, $cluster\ 1\%$ représente le nombre de clusters tels que : $|c| > 0.1 \times S$. En effet, nous pensons qu'il ne faut considérer que les clusters dont la taille est supérieure à une certaine proportion du nombre total de séquences (on ne garde que les clusters les plus grands). Avec 1% et un batch de 1000 séquences, par exemple, un cluster $c$ tel que $|c| < 10$ ne sera pas considéré. $time\ 1\%$ représente le temps d'exécution quand on ne garde que les clusters de taille supérieure à 1% de $S$. On peut observer que le nombre de clusters augmente de façon linéaire. Le temps de réponse est de toute évidence lié à la taille des batches, mais il est raisonnable de dire que l'utilisateur final peut choisir la taille de ses batches en fonction du degré de rapidité souhaité.

## 4.4 Analyse de la qualité des clusters

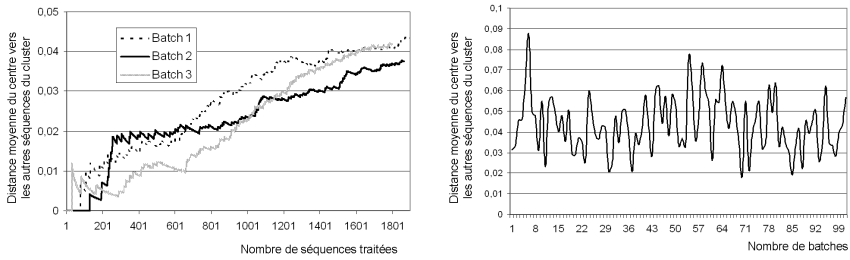


FIG. 7 – *Distance globale étape par étape et batch par batch*

Afin de mesurer la qualité des classes produites par SMDS, notre principal outil sera la distance entre deux séqunces. Soit $s_1$ et $s_2$, deux séquences, la distance $dist(s_1, s_2)$ entre $s_1$ et $s_2$ est basée sur $sim(s_1, s_2)$, la mesure de similitude donnée par la définition 4 et telle que $dist(s_1, s_2) = 1 - sim(s_1, s_2)$. On a donc $dist(s_1, s_2) \in [0..1]$ et $dist(s_1, s_2)$ proche de 0 signifie que les séquences sont proches (similaire si cette valeur est nulle) alors que $dist(s_1, s_2)$ proche de 1 signifie que les séquences sont éloignées (ne partagent aucun item si cette valeur est 1). Nous avons utilisé deux mesures pour la qualité des classes. La première est le diamètre d'une classe $C$. Il s'agit de la plus grande distance entre deux séquences de $C$. Un diamètre de $0\%$ montre que la classe est constitué uniquement de séquences égales alors qu'un diamètre de $100\%$ montre que la classe contient au moins deux séquences qui ne partagent aucun item. Lors de nos expérimentations, le diamètre moyen a varié entre 2% et 3%. La seconde mesure est la "double moyenne". Elle est basée sur le centre de la classe. Soit $C$ une classe, le centre de $C$ est une séquence $c$ telle que : $\forall s \in C, \sum_{x \in C} dist(s, x) \geq \sum_{y \in C} dist(c, y)$. Nous sommes donc en mesure de donner, pour $C$, la distance moyenne ($DM$) entre $c$ et toutes les autres séquences de $C$ : $DM = \frac{\sum_{x \in C} dist(x,c)}{|C|}$. Nous reportons dans la figure 6 (partie droite) quelque distances moyennes parmi les pires obtenues durant nos expérimentations. Pour chaque séquences ajoutée dans une classe, nous donnons la valeur $DM$ de cette classe.

Par exemple, après l'ajout de la dernière séquence dans le cluster 1, la valeur de $DM$ pour ce cluster est 22%. On peut observer que $DM$ varie de 0 (quand $|C| = 1$ l'unique séquence est le centre) à 50%. $DM$ décroit alors rapidement jusqu'à des valeurs comprises entre 20% et 35%, ce qui est un bon résultat compte tenu du fait que la figure 6 (partie droite) ne comprend que les résultats des clusters les moins homogènes. Les autres clusters sont homogènes et offrent à l'algorithme d'alignement un cadre adéquat. Nous reportons dans la figure 7 (partie gauche) la double moyenne $DBM$ après avoir traité chaque séquence d'un batch. $DBM$ est calculée de la manière suivante : soit $C$ l'ensemble des classes, $DBM = \frac{\sum_{i \in C} \sum_{x \in C_i} dist(x, c_i)}{|C|}$ avec $c_i$ le centre de $C_i$ (la $i^{eme}$ classe). On peut observer dans la figure 7 (partie gauche) que pour le second batch, $DBM$ augmente rapidement jusqu'à 2% (séquence 220), puis augmente lentement jusqu'à 3,7%. La valeur finale de $DBM$ à la fin du batch est donnée par la figure 7 (partie droite). On peut y observer que $DBM$ est toujours comprise entre 2% et 9%. A la fin du processus, la valeur moyenne de $DBM$ est de 4,3% (une qualité moyenne des classes de 95,7%).

# 5 Conclusion

Dans ce papier, nous avons proposé la méthode SMDS, conçue pour extraire rapidement les séquences d'un data stream et en proposer un résumé significatif. Notre algorithme repose sur une technique de classification combinée avec un alignement des séquences. Le processus d'alignement repose sur un algorithme de classification glouton qui considère que dans le contexte du Web les navigations ont certaines caractéristiques qu'il faut prendre en compte. De plus nous avons proposé une solution adaptée pour gérer efficacement les séquences et leur historique dans un arbre préfixé. Grâce à cette façon de traiter le data stream, SMDS est capable de détecter des comportement partagé par un nombre relativement faible d'utilisateurs (*e.g.* 13 utilisateurs ou encore $0,5\%$) ce qui est proche du difficile problème de l'extraction de motifs séquentiels avec un support très faible. De plus, nos expérimentations ont montré que SMDS traite le data stream assez rapidement pour être intégré dans un contexte temps réel. Nous avons également montré que SMDS propose des classes de très bonne qualité, ce qui permet d'extraire les motifs les plus pertinents et de façon exhaustive.

# Références

Agrawal, R. et R. Srikant (1995). Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, Taiwan.

Chang, J. H. et W. S. Lee (2003). Finding recent frequent itemsets adaptively over online data streams. In *KDD '03 : Proceedings of the ninth international conference on Knowledge discovery and data mining*, pp. 487–492.

Chen, Y., G. Dong, J. Han, B. Wah, et J. Wang (2002). Multidimensional regression analysis of time-series data streams.

Cormode, G. et S. Muthukrishnan (2005). What's hot and what's not : tracking most frequent items dynamically. *ACM Trans. Database Syst. 30*(1), 249–278.

Datar, M., A. Gionis, P. Indyk, et R. Motwani (2002). Maintaining stream statistics over sliding windows. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pp. 635–644.

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*.

Giannella, C., J. Han, J. Pei, X. Yan, et P. Yu (2003). *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining. AAAI/MIT.

Hay, B., G. Wets, et K. Vanhoof (2002). Web Usage Mining by Means of Multidimensional Sequence Alignment Method. In *WEBKDD*, pp. 50–65.

Kum, H., J. Pei, W. Wang, et D. Duncan (2003). ApproxMAP : Approximate mining of consensus sequential patterns. In *Proceedings of SIAM Int. Conf. on Data Mining*, San Francisco, CA.

Masseglia, F., F. Cathala, et P. Poncelet (1998). The PSP Approach for Mining Sequential Patterns. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, Nantes, France.

Masseglia, F., P. Poncelet, et R. Cicchetti (April 2000). An efficient algorithm for web usage mining. *Networking and Information Systems Journal (NIS)*.

Pei, J., J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, et M. Hsu (2001). PrefixSpan : Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In *17th International Conference on Data Engineering (ICDE)*.

Raissi, C., P. Poncelet, et M. Teisseire (2005). Need for SPEED : Mining Sequential Patterns in Data Streams. In *Actes des 21ièmes Journées Bases de Données Avancées (BDA'05)*, Saint-Malo, France.

Teng, W.-G., M.-S. Chen, et P. S. Yu (2003). A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *VLDB*, pp. 93–104.

Wang, J. et J. Han (2004). BIDE : Efficient Mining of Frequent Closed Sequences. In *Proceedings of the International Conference on Data Engineering (ICDE'04)*, Boston, M.A.

## Summary

In recent years, emerging applications introduced new constraints for data mining methods. These constraints are typical of a new kind of data: the "*data streams*". In a data stream processing, memory usage is restricted, new elements are generated continuously and have to be considered as fast as possible, no blocking operator can be performed and the data can be examined only once. We argue that the main issue is the combinatory phenomenon related to sequential pattern mining. In this paper, we propose an algorithm based on sequences alignment for mining approximate sequential patterns in data streams. To meet the constraint of one scan, a greedy clustering algorithm associated to an alignment method are proposed. We will show that our proposal is able to extract relevant sequences with very low thresholds.