

Mining Data Streams for Frequent Sequences Extraction

Alice Marascu and Florent Masegla
 INRIA Sophia Antipolis
 AxIS Project/Team
 2004 route des Lucioles - BP 93
 06902 Sophia Antipolis, France
 Email: {amarascu,fmasegl}@sophia.inria.fr

Abstract— In recent years, emerging applications introduced new constraints for data mining methods. These constraints are particularly linked to new kinds of data that can be considered as complex data. One typical kind of such data is known as *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. At this time and to the best of our knowledge, no method has been proposed for mining sequential patterns in data streams. We argue that the main reason 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 Web usage 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.

Keywords: data streams, sequential patterns, web usage mining, clustering, sequences alignment.

I. INTRODUCTION

The problem of mining sequential patterns from a large static database has been widely addressed [1], [10], [13], [15], [9]. The extracted relationship is known to be useful for various applications such as decision analysis, marketing, usage analysis, etc. In recent years, emerging applications such as network traffic analysis, intrusion and fraud detection, web clickstream mining or analysis of sensor data (to name a few), introduced new constraints for data mining methods. Beside those applications appeared the concept of *complex data*. Among those new kinds of complex data, an increasing amount of work is being done on the *data streams*. A data stream processing has to satisfy the following constraints: 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. Hence, many methods have been proposed for mining items or patterns from data streams [5], [2], [4]. One potential characteristic of complex data is their low reliability and the approximation that has to be accepted in some cases. For the field of mining data streams, the main problem was at first to satisfy the constraints of the data stream environment and provide efficient methods for extracting patterns as fast as possible. For this purpose, approximation has been recognized as a key feature [6]. Then, recent methods [3], [7], [14] introduced different principles

for managing the history of the extracted patterns frequencies. The main idea is that people are often more interested in recent changes. [7] introduced the *logarithmic tilted time window* for storing patterns frequencies with a fine granularity for recent changes and a coarse granularity for long term changes. In [14] the frequencies are represented by a regression-based scheme and a particular technique is proposed for segment tuning and relaxation (merging old segments for saving main memory). However, at this time and to the best of our knowledge, no method has been proposed for mining sequential patterns in data streams. We argue that the main reason is the combinatory phenomenon related to sequential pattern mining. Actually, if itemset mining relies on a finite set of possible results (the set of combinations between items recorded in the data) this is not the case for sequential patterns where the set of results is infinite. In fact, due to the temporal aspect of sequential patterns, an item can be repeated without limitation leading to an infinite number of potential frequent sequences. FTP-DS [14], for instance, is designed for mining inter-transaction patterns from data streams. Actually, the patterns extracted in their framework are itemsets and this work does not address the extraction of sequences as we propose to do.

In this paper, we propose the SMDS (Sequence Mining in Data Streams) algorithm which is based on sequences alignment (such as [9], [8] have already proposed for static databases) for mining approximate sequential patterns in data streams. The goal of this paper is first to show that classic sequential pattern mining methods cannot be included in a data stream environment because of their complexity and then to propose a solution. The proposed algorithm is implemented and tested over real and synthetic datasets. Our data come from the access log files of Inria Sophia-Antipolis. We will thus show the efficiency of our mining scheme for Web usage data streams, though our method might be applied to any kind of sequential data. Analyzing the behavior of a Web site's users, also known as Web Usage Mining, is a research field which consists in adapting the data mining methods to the records of access log files. These files collect data such as the IP address of the connected host, the requested URL, the date and other information regarding the navigation of the user. Web Usage Mining techniques provide knowledge about the behavior of the users in order to extract relationships in the recorded data. Among available techniques, the sequential patterns are

particularly well adapted to the log study. Extracting sequential patterns on a log file, is supposed to provide this kind of relationship: “On the Inria’s Web Site, 10% of users visited consecutively the homepage, the available positions page, the ET¹ offers, the ET missions and finally the past ET competitive selection”. We want to extract typical behaviours from clickstream data and show that our algorithm meets the time constraints in a data stream environment and can be included in a data stream process at a negligible cost. The rest of this paper is organized as follows. The definitions of Sequential Pattern Mining and Web Usage Mining are given in Section II.

II. DEFINITIONS

In this section we define the sequential pattern mining problem in large databases and give an illustration. Then we explain the goals and techniques of Web Usage Mining with sequential patterns.

A. Sequential Pattern Mining

The problem of mining sequential patterns from static databases is defined as follows [1]:

Definition 1: Let $I = \{i_1, i_2, \dots, i_m\}$, be a set of m literals (items). I is a k -itemset where k is the number of items in I . A sequence is an ordered list of itemsets denoted by $\langle s_1 s_2 \dots s_n \rangle$ where s_j is an itemset. The *data-sequence* of a customer c is the sequence in D corresponding to customer c . A sequence $\langle a_1 a_2 \dots a_n \rangle$ is a *subsequence* of another sequence $\langle b_1 b_2 \dots b_m \rangle$ if there exist integers $i_1 < i_2 < \dots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$.

Example 1: Let C be a client and $S = \langle (c) (d e) (h) \rangle$, be that client’s purchases. S means that “ C bought item c , then he bought d and e at the same moment (i.e. in the same transaction) and finally bought item h ”.

Definition 2: The *support* of a sequence s , also called $supp(s)$, is defined as the fraction of total data-sequences that contain s . If $supp(s) \geq minsupp$, with a minimum support value $minsupp$ given by the user, s is considered as a *frequent sequential pattern*.

The problem of sequential pattern mining is thus to find all the frequent sequential patterns as stated in definition 2.

B. From Web Usage to Data Streams

For classic Web usage mining methods, the general idea is similar to the principle proposed in [11]. Raw data is collected in access log files by Web servers. Each input in the log file illustrates a request from a client machine to the server (*http daemon*).

Definition 3: Let Log be a set of server access log entries. An entry g , $g \in Log$, is a tuple $g = \langle ip_g, ([l_1^g.URL, l_1^g.time] \dots [l_m^g.URL, l_m^g.time]) \rangle$ such that for $1 \leq k \leq m$, $l_k^g.URL$ is the item asked for by the user g at time $l_k^g.time$ and for all $1 \leq j < k$, $l_k^g.time > l_j^g.time$.

The structure of a log file, as described in definition 3, is close to the “Client-Time-Item” structure used by sequential

Client	d1	d2	d3	d4	d5
1	a	c	d	b	c
2	a	c	b	f	c
3	a	g	c	b	c

TABLE I

FILE OBTAINED AFTER A PRE-PROCESSING STEP

pattern algorithms. In order to extract frequent behaviors from a log file, for each g in the log file, we first have to transform ip_g into a client number and for each record k in g , $l_k^g.time$ is transformed into a time number and $l_k^g.URL$ is transformed into an item number. Table I gives a file example obtained after that pre-processing. To each client corresponds a series of times and the URL requested by the client at each time. For instance, the client 2 requested the URL “f” at time $d4$. The goal is thus, according to definition 2 and by means of a data mining step, to find the sequential patterns in the file that can be considered as frequent. The result may be, for instance, $\langle (a)(c)(b)(c) \rangle$ (with the file illustrated in table I and a minimum support given by the user: 100%). Such a result, once mapped back into URLs, strengthens the discovery of a frequent behavior, common to n users (with n the threshold given for the data mining process) and also gives the sequence of events composing that behavior.

Nevertheless, most methods which were designed for mining patterns from access log files cannot be applied to a data stream coming from web usage data (such as clickstreams). In our context, we consider that large volumes of usage data are arriving at a rapid rate. Sequences of data elements are continuously generated and we aim at identifying representative behaviours. We assume that the mapping of URLs and clients as well as the data stream management are performed simultaneously. Furthermore, as stated by [12], sequential pattern extraction, when applied to Web access data, is effective only if the support is very low. A low support means long response time and the authors proposed a divisive approach to extract sequential patterns on similar navigations (in order to get highly significant patterns). Our goal with this work is close to that of [12] since we will provide a navigation clustering scheme designed to facilitate the discovery of interesting sequences, while meeting the needs for rapid execution times involved in data stream processing.

III. SMDS: MOTIVATION AND PRINCIPLE

Our method relies on a batch environment (widely inspired from [7]) and the prefix tree structure of PSP [10] (for managing frequent sequences). We first study the limitations of a sequential pattern mining algorithm that would be integrated in a data stream context. Then, we propose our framework, based on a sequences alignment principle.

A. Limits of Sequential Pattern Mining

Our method will process the data stream as batches of fixed size. Let B_1, B_2, \dots, B_n be the batches, where B_n is the most recent batch of transactions. The principle of SMDS will be to extract frequent sequential patterns from each batch b in

¹ET: Engineers, Technicians

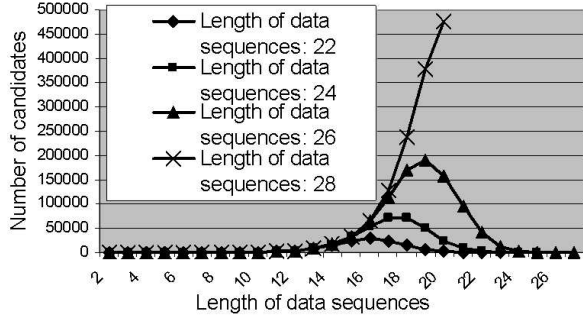


Fig. 1. Limits of a framework involving PSP

$[B_1..B_n]$ and to store the frequent approximate sequences in a prefix tree structure (inspired from [10]). Let us consider that the frequent sequences are extracted with a classic exhaustive method (designed for a static transaction database). We argue that such a method will have at least one drawback leading to a blocking operator. Let us consider the example of the PSP [10] algorithm. We have tested this algorithm on databases containing only two sequences (s_1 and s_2). Both sequences are equal and contain repetitions of itemsets having length one. The first database contains 11 repetitions of the itemsets (1)(2) (i.e. $s_1 = \langle (1)(2)(1)(2)...(1)(2) \rangle$, $length(s_1)=22$ and $s_2 = s_1$). The number of candidates generated at each scan is reported in figure 1. Figure 1 also reports the number of candidates for databases of sequences having length 24, 26 and 28. For the base of sequences having length 28, the memory was exceeded and the process could not succeed. We made the same observation for PrefixSpan² [13] where the number of intermediate sequences was similar to that of PSP with the same mere databases. If this phenomenon is not blocking for methods extracting the whole exact result (one can select the appropriate method depending on the dataset), the integration of such a method in a data stream process is impossible because the worst case can appear in any batch³.

B. Principle

The outline of our method is the following: for each batch of transactions, discovering clusters of users (grouped by behavior) and then analyzing their navigations by means of a sequences alignment process. This allows us to obtain clusters of behaviours representing the current usage of the Web site. For each cluster having size greater than $minSize$ (specified by the user) we store only the summary of the cluster. This summary is given by the aligned sequence obtained on the sequences of that cluster.

A Greedy Approach for Clustering Sequences

The clustering scheme we have developed is a straight forward, naive algorithm. It is based on the fact that navigations on a

Step 1 :				
S_1 :	$\langle (a,c) \rangle$	$\langle (e) \rangle$	$\langle (h) \rangle$	$\langle (m,n) \rangle$
S_2 :	$\langle (a,d) \rangle$	$\langle (e) \rangle$	$\langle (h) \rangle$	$\langle (m,n) \rangle$
SA_{12} :	$\langle (a,2, c:1, d:1):2 \rangle$	$\langle (e):2 \rangle$	$\langle (h:1):1 \rangle$	$\langle (m:2, n:2):2 \rangle$
Step 2 :				
SA_{12} :	$\langle (a:2, c:1, d:1):2 \rangle$	$\langle (e):2 \rangle$	$\langle (h:1):1 \rangle$	$\langle (m:2, n:2):2 \rangle$
S_3 :	$\langle (a,b) \rangle$	$\langle (e) \rangle$	$\langle (i,j) \rangle$	$\langle (m) \rangle$
SA_{13} :	$\langle (a:3, b:1, c:1, d:1):3 \rangle$	$\langle (e):3 \rangle$	$\langle (h:1, i:1, j:1):2 \rangle$	$\langle (m:3, n:2):3 \rangle$
Step 3 :				
SA_{13} :	$\langle (a:3, b:1, c:1, d:1):3 \rangle$	$\langle (e):3 \rangle$	$\langle (h:1, i:1, j:1):2 \rangle$	$\langle (m:3, n:2):3 \rangle$
S_4 :	$\langle (b) \rangle$	$\langle (e) \rangle$	$\langle (h,i) \rangle$	$\langle (m) \rangle$
SA_{14} :	$\langle (a:3, b:2, c:1, d:1):4 \rangle$	$\langle (e):4 \rangle$	$\langle (h:2, i:2, j:1):3 \rangle$	$\langle (m:4, n:2):4 \rangle$

Fig. 2. Different steps of the alignment

Web site are usually very similar or very different from each other. Basically, users interested in the pages related to job opportunities are usually not likely to request pages related to the next seminar organized by the Inria's unit of Sophia-Antipolis. In order to cluster the navigations as fast as possible, our greedy approach is thus based on the following scheme: the algorithm is initialized with one cluster containing the first navigation. Then, for each navigation n in the the batch, n is compared to each cluster c . As soon as n is found to be similar to at least one sequence of c then s is inserted in c . If s has been inserted in no cluster, then a new cluster is created and s is inserted in this new cluster. The similarity between two sequences is given in definition 4 (s is inserted in c if the following condition holds: $\exists s_c \in c / sim(s, s_c) \leq minSim$ with $minSim$ given by the user).

Definition 4: Let s_1 and s_2 be two sequential patterns. Let $|LCS(s_1, s_2)|$ be the length of the longest common subsequences between s_1 and s_2 . The similarity $sim(s_1, s_2)$ between s_1 and s_2 is defined as follows: $sim = \frac{2 \times |LCS(s_1, s_2)|}{length(s_1) + length(s_2)}$.

Sequences Alignment

The clustering algorithm ends with clusters of similar sequences, which is a key element for sequences alignment. Several algorithms ([9], [8]) have been proposed at this time for the alignment of sequences. In [9], the authors introduced the notion of weighted sequence and consensus pattern in their alignment method. The weighted sequences notion of [9] is used as is in SMDS. It leads to a weighted sequence represented as follows: $SA = \langle I_1 : n_1, I_2 : n_2, \dots, I_r, n_r \rangle : m$. In this representation, m stands for the total number of sequences involved in the alignment. I_p ($1 \leq p \leq r$) is an itemset represented as $(x_{i_1} : m_{i_1}, \dots, x_{i_t} : m_{i_t})$, where m_{i_t} is the number of sequences containing the item x_i at the p^{th} position in the aligned sequences. Finally, n_p is the number of occurrences of itemset I_p in the alignment. Example 2 describes the alignment process on 4 sequences. Starting from two sequences, the alignment begins with the insertion of empty items (at the beginning, the end or inside the sequence) until both sequences contain the same number of itemsets.

Example 2: Let us consider the following sequences: $S_1 = \langle (a,c) \rangle$, $S_2 = \langle (a,d) \rangle$, $S_3 = \langle (a,b) \rangle$, $S_4 = \langle (b) \rangle$. The steps leading to the alignment of those sequences are detailed in Figure 2. At first, an empty itemset is inserted in S_1 . Then S_1 and S_2 are aligned in order to provide SA_{12} . The alignment process

²<http://www-sal.cs.uiuc.edu/~hanj/software/prefixspan.htm>

³In a web usage pattern, for instance, numerous repetitions of requests for pdf or php files are usual.

is then applied to SA_{12} and S_3 . The alignment method goes on processing two sequences at each step.

At the end of the alignment process, the aligned sequence (SA_{14} in figure 2) is a summary of the corresponding cluster. The approximate sequential pattern can be obtained by specifying k : the number of occurrences of an item in order for it to be displayed. For instance, with the sequence SA_{14} from figure 2 and $k = 2$ the filtered aligned sequence will be: $\langle(a,b)(e)(h,i)(m,n)\rangle$ (corresponding to items having a number of occurrences greater or equal to k).

Sequences Storage and Management

The aligned sequences obtained from the previous step are stored in a prefix tree similar to that of [10]. If a new sequence s has been discovered, then the tree is modified to store this new sequence. Otherwise, if s is already in the tree, then the support of s is updated. Figure 3 gives an example of a prefix tree where transaction cutting is captured by using labelled edges. Each path from the root to any node in the tree stands for an extracted sequence. The tree from figure 3 contains 6 sequences ($\langle(a\ c)\rangle$, $\langle(a\ d)\rangle$, $\langle(b)\rangle$, $\langle(c\ d)\rangle$, $\langle(c)(e)\rangle$, $\langle(d)(a)\rangle$). Any path, from the root to a leaf stands for a sequence and considering a single branch each node at depth l ($k \geq l$) captures the l^{th} item of the sequence. Transaction cutting is captured by using labelled edges. For instance, the dashed link between nodes c and e in figure 3 illustrates the fact that e is not in the same itemset as c . Each node is provided with k , the filter used to obtain this aligned sequence from the corresponding cluster. Example 3 gives an illustration of the sequences support management.

Example 3: Let us consider the sequence $s_1 = \langle(a)\rangle$ from figure 3. The support of the aligned sequence s_1 is unknown (-1). This means that s_1 has been extracted in more than one cluster. Let us consider the sequence $s_2 = \langle(d)(a)\rangle$. The support of the aligned sequence s_2 is 3 (meaning that the corresponding cluster contains this aligned sequence with a filter $k = 3$).

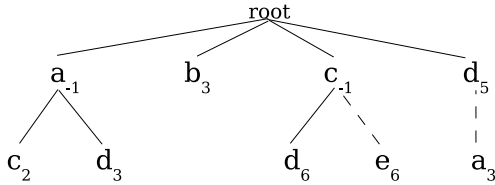


Fig. 3. Example of a prefix tree

As we wrote in section I, the first challenge of mining data streams was to extract patterns as fast as possible in order to get adapted to the speed of the streams. Then the history of frequencies has been considered and tilted time windows were proposed [7], [3]. However, no particular effort has been made for extracting temporal relationships between items in data streams (sequences, sequential patterns). Even if our main goal was to show that such patterns could be extracted with SMDS, we have provided our method with logarithmic

tilted time windows. Let $f_S(i, j)$ denote the frequency of a sequence S in $B_{(i,j)} = \cup_{k=i}^j B_k$, with B_k the k^{th} batch of transactions. Let B_n be the current batch, the logarithmic tilted time windows allow to store the set of frequencies $[f(n, n); f(n-1, n-1); f(n-2, n-3); f(n-4, n-7), \dots]$ and to save main memory. Frequencies are shifted in the tilted time window when updating with a new batch B . For this purpose, $f_S(B)$ replaces $f(n, n)$, $f(n, n)$ replaces $f(n-1, n-1)$ and so on. When two windows are old enough, their frequencies are merged in a single window. This principle allows to give more importance to recent events (their frequencies are stored at a fine level of granularity) and less importance to past events (their frequencies are stored with a precision getting less and less fine). Tail pruning is also implemented. Actually, in our version, tail frequencies (oldest records) are dropped when their timestamp is greater than a fixed timestamp given by the user (e.g. we only store frequencies for the last 100,000 batches, which will require $\log_2(100,000) \approx 17$ units of time).

The SMDS algorithm described in this paper is given below.

Algorithm 1 (SMDS)

Input : $B = \cup_{i=0}^{\infty} B_i$: an infinite set of batches of transactions ; $minSize$: the minimum size of a cluster that has to be summarized ; $minSim$: the minimum similarity between two sequences in order to consider growing a cluster ; k : the filter for the sequences alignment method.

Output : The updated prefix tree structure of approximate frequent sequences.

While (B) Do

$b \leftarrow NextBatch()$;

// 1) Obtain clusters of size $> minSize$

$C \leftarrow Clustering(b, minSize, minSim)$;

// 2) Summarize each cluster with filter k ;

Foreach (c ∈ C) Do

$SA_c \leftarrow Alignment(c, k)$;

//3) Store frequent sequences

If (SA_c) **Then** PrefixTree \leftarrow PrefixTree+ SA_c

Done

// 4) Update Tilted Time Windows and delete

// obsolete sequences

TailPruning(PrefixTree);

Done (end Algorithm SMDS);

Complexity

Let n be the number of sequences in the batch. In the worst case, the clustering algorithm has a time complexity of $O(n^2)$. Actually, in the worst case, LCS is called once for the first sequence, twice for the second, and so on. The complexity is thus $O(\frac{n \cdot (n+1)}{2}) = O(n^2)$. Even if our algorithm's complexity (in the worst case) might be improved, it is well adapted for Web navigation patterns and the results obtained (see Section IV) on real datasets (access logs of Inria Sophia-Antipolis) show its effectiveness. Let us consider that all the sequences in a cluster C have length m and $|C| = p$. The complexity of the alignment algorithm for a batch is $O(p \cdot m^2)$.

IV. EXPERIMENTS

The SMDS algorithm is written in Java on a Pentium (2.1 Ghz) PC running a Linux Fedora system. We evaluated our proposal on both real and synthetic data (available at <http://www.almaden.ibm.com/cs/quest>).

A. Feasibility and Scalability of SMDS

In order to show the efficiency of the SMDS algorithm, we report in figure 4 the time needed to extract the longest approximate sequential pattern on each batch corresponding to the Web usage data (up figure 4) and the synthetic data (down figure 4). For the Inria's web site, the data were collected over a period of 14 months for a size of 14 Gb. The total amount of navigations is 3.5 millions and the total number of items is 300,000. We cut down the log into batches of 4500 transactions (an average amount of 1500 navigation sequences). For those experiments, the filter k was fixed to 30 % (please note that this filter has an impact on response time, since the sequences managed in the prefix tree will be longer when k is low). In our experiment we have injected "parasitic" navigations into the batches. The first batch was not modified. The second batch was added with ten sequences containing repetitions of 2 items and having length 2 (as s_1 and s_2 described in Section III-A). The third batch was added with ten such sequences having length 3 and so on up to ten sequences having length 30 in batch number 30. The goal is to show that a classic method (PSP, prefixSpan) will block the data stream whereas SMDS will go on performing the mining task. We can observe that the response time of SMDS varies from 1800 ms to 3100 ms. PSP performs very well for the first batches and finally is penalized by the noise added to the data stream (*i.e.* batch 19). The test has also been conducted with prefixSpan. The execution times were greater than 6000 ms and prefixSpan had the same exponential behaviour because of the noise injected in the data streams. For both PSP and prefixSpan the specified minimum support was just enough to find the injected sequences of repetitions (10 sequences). We added to figure 4 the number of sequences involved in each batch in order to explain the different execution times of SMDS. We can observe, for instance, that batch number 1 contains 1500 sequence and SMDS needs 2700 ms in order to extract the approximate sequential patterns. For the synthetic data we generated batches of 10,000 transactions (corresponding to 500 sequences in average). The average length of sequences was 10 and the number of items was 200,000. The filter k was fixed to 30 %. We report in figure 4 (down) the response time and the number of sequences corresponding to each batch. We can observe that SMDS is able to handle 10,000 transactions in less than 4 seconds (*e.g.* batch number 2).

B. Patterns Extracted on Real Data

The list of behaviours discovered by SMDS covers more than 100 navigation goals (clusters of navigation sequences) on the Web site of Inria Sophia-antipolis. Most of the discovered patterns can be considered as "rare" but very "confident" (their support is low with respect to the global number of

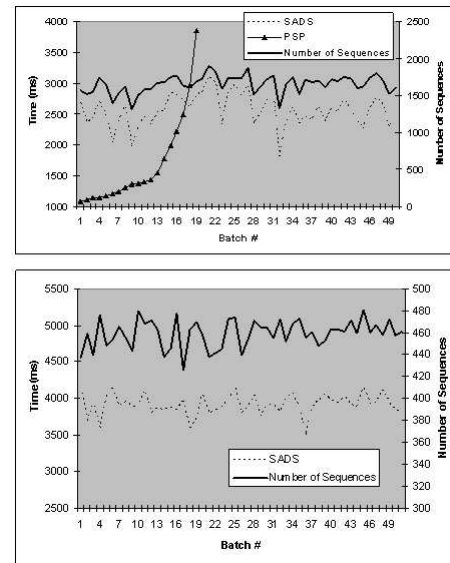


Fig. 4. SMDS execution time

sequences in the batch, but the filter k used for each cluster is high). We report here a sample of two discovered behaviours: **A)** $k = 30\%$, cluster size = 13, prefix="http://www-sop.inria.fr/omega/":

```
< (MC2QMC2004)
(personnel/Denis.Talay/moi.html)
(MC2QMC2004/presentation.html)
(MC2QMC2004/dates.html)
(MC2QMC2004/Call_for_papers.html)>
```

This sequence has been found on batches corresponding to june 2004, when the conference MCQMC has been organized by a team of Inria Sophia Antipolis. This behaviour was shared by up to 13 users (cluster size).

B) $k = 30\%$, cluster size = 10, prefix="http://www-sop.inria.fr/acacia/personnel/itey/Francais/Cours/":

```
< (programmation-fra.html) (PDF/chapitre-cplus.pdf) (cours-
programmation-fra.html)
(programmation-fra.html) >
```

This behaviour corresponds to requests that have been made for a document about programming lessons written by a member of a team from Inria Sophia Antipolis. It has been found on a batch corresponding to april 2004. For the usage sequences of Inria sophia Antipolis, we also observed that SMDS is able to detect the parasitic sequences that we added to the batches (long sequences containing multiple repetitions of 2 items). Those sequences are gathered together in one cluster.

C. Size of the Batches

Since the complexity of our algorithm depends on the number of sequences in the batch, we conducted a study on the impact of the size of the batches on the response time. For this purpose we report in figure 5 the response time when S , the size of the batch, varies from 100 to 5000 sequences. *time* stands for the response time. *clusters* stands for the number of clusters extracted by SMDS. *clusters 1%* stands for the number of

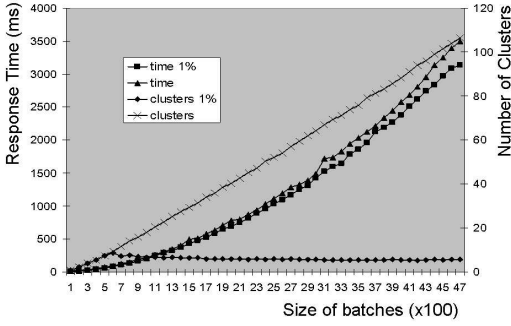


Fig. 5. Size of the batches

clusters c such that $|c| > 0.1 \times S$. Indeed, we argue that the number of clusters has to be filtered. We thus propose to consider only the clusters such that their size is greater than a particular ratio of the number of sequences (large clusters). With 1% and a batch of 1000 sequences, for instance, a cluster c such that $|c| < 10$ will not be considered. *time 1%* stands for the response time needed to process only the large clusters. We can observe that the number of clusters grows linearly, but the number of large clusters remains the same after a short number of iterations. The response time is obviously linked to the size of the batches, but it is reasonable to say that the final user can choose the size of the batches depending on the desired response time.

D. Analyzing the Quality of the Clusters

In order to give a measure of the quality of our clusters, our main tool will be the distance between two sequences. Let s_1 and s_2 be two sequences, the distance $dist(s_1, s_2)$ between s_1 and s_2 is based on $sim(s_1, s_2)$, the similarity given in definition 4, and is such that $dist(s_1, s_2) = 1 - sim(s_1, s_2)$. Thus, $dist(s_1, s_2) \in [0..1]$ and $dist(s_1, s_2) = 0$ means that the sequences are the same whereas $dist(s_1, s_2) = 1$ means that s_1 and s_2 do not share any item. We used two main measures. The first one is the diameter of a cluster C . It stands for the largest distance between two sequences of C . A diameter of 0% shows that the cluster contains only equal sequences, whereas a diameter of 100% shows that the cluster contains at least two sequences that do not share any item. During our experiments the average diameter at the end of each batch varied from 2% to 3%.

The second measure is the “double average”. It is based on the center of the cluster, which is given as follows. Let C be a cluster, the center of C is a sequence c such that: $\forall s \in C, \sum_{x \in C} dist(s, x) \geq \sum_{y \in C} dist(c, y)$. We are thus able to give, for C , the average distance (*AD*) from c to all the remaining sequences of C :

$$AD = \frac{\sum_{x \in C} dist(x, c)}{|C|}$$

We report in figure 6 some of the worst average distances obtained during our experiments. For each sequence added in a cluster, we report the new value of *AD* for this cluster. For instance, when the last sequence is added to cluster 1, the value of *AD* for cluster 1 is 22%. We can observe that *AD* varies from 0 (when $|C| = 1$ the center is the only sequence)

to 50%. *AD* thus decreases rapidly to a value between 20% and 35% which is a good result considering that in figure 6 are reported the rare clusters that are not really homogeneous. The remaining clusters are very well formed and thus give a good framework for the alignment algorithm. Indeed, we report in figure 7 the “double average” value (*DA*) after processing each sequence of the batch. *DA* is given as follows: let N be the set of clusters, $DA = \frac{\sum_{i \in N} \sum_{x \in C_i} dist(x, c_i)}{|N|}$ with c_i the center of C_i (the i^{th} cluster). We can observe in figure 7 that for the second batch, *DA* rapidly grows up to almost 2% after sequence 220, then *DA* slightly increases to 3.7%. The final value of *DA* at the end of each batch is given in figure 8. We can observe that the value *DA* is always between 2% and 9%. At the end of the process, the average value of *DA* is 4.3% (an average clusters quality of 95.7%).

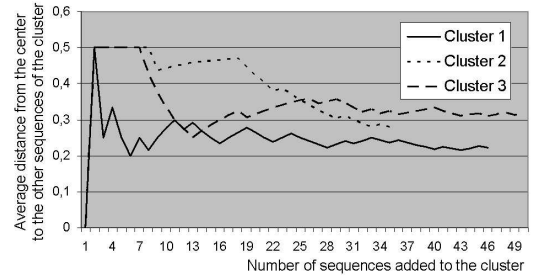


Fig. 6. Worst clusters step by step

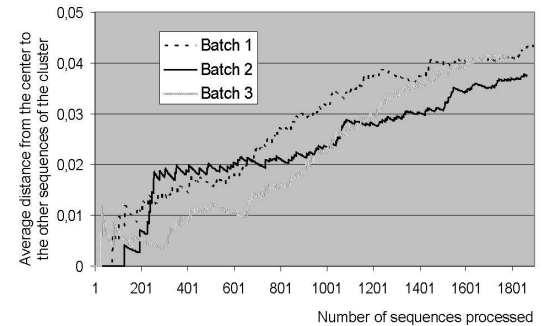


Fig. 7. Global distance step by step

V. CONCLUSION

In this paper, we proposed the SMDS algorithm for extracting sequential patterns in data streams. Our method has two major features. Our first study was intended to show the main limits and problems that have to be identified and solved prior to proposing a method for this subject. Then we proposed a principle designed for rapidly processing the sequences of a data stream and extracting the meaningful summary. Our algorithm relies on a clustering and alignment method associated to an efficient structure for managing the extracted sequences and their history. First, batches of transactions are summarized by means of a sequences alignment method. This alignment process relies on a greedy clustering algorithm that

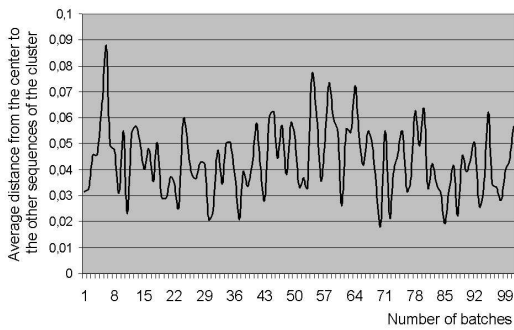


Fig. 8. Global distance batch by batch

considers the main characteristics of Web usage sequences in order to provide distinct clusters of sequences. Second, frequent sequences obtained by SMDS are stored in a prefix tree structure. Thanks to this mining scheme, SMDS is able to detect frequent behaviours shared by very little amounts of users (*e.g.* 13 users, or 0.5%) which is close to the difficult problem of mining sequential patterns with a very low support. Furthermore, our experiments have shown that SMDS performs fast enough to be integrated in a data stream environment at a negligible cost. We also reported the results of our experiments showing that SMDS provides clusters of quality and extracts the significant patterns of a Web usage analysis.

REFERENCES

- [1] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, Taiwan, March 1995.
- [2] Joong Hyuk Chang and Won Suk Lee. Finding recent frequent itemsets adaptively over online data streams. In *KDD '03: Proceedings of the ninth international conference on Knowledge discovery and data mining*, pages 487–492, 2003.
- [3] Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multidimensional regression analysis of time-series data streams, 2002.
- [4] Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.
- [5] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 635–644, 2002.
- [6] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 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*, 2002.
- [7] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. *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, 2003.
- [8] Birgit Hay, Geert Wets, and Koen Vanhoof. Web Usage Mining by Means of Multidimensional Sequence Alignment Method. In *WEBKDD*, pages 50–65, 2002.
- [9] H. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMAP: Approximate mining of consensus sequential patterns. In *Proceedings of SIAM Int. Conf. on Data Mining*, San Francisco, CA, 2003.
- [10] F. Masegla, F. Cathala, and P. Poncelet. The PSP Approach for Mining Sequential Patterns. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, Nantes, France, September 1998.
- [11] F. Masegla, P. Poncelet, and R. Cicchetti. An efficient algorithm for web usage mining. *Networking and Information Systems Journal (NIS)*, April 2000.
- [12] F. Masegla, Doru Tanasa, and Brigitte Trousse. Web usage mining: Sequential pattern extraction with a very low support. In *6th Asia-Pacific Web Conference, APWeb, Hangzhou, China*, 2004.
- [13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and MC. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In *17th International Conference on Data Engineering (ICDE)*, 2001.
- [14] Wei-Guang Teng, Ming-Syan Chen, and Philip S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *VLDB*, pages 93–104, 2003.
- [15] J. Wang and J. Han. BIDE: Efficient Mining of Frequent Closed Sequences. In *Proceedings of the International Conference on Data Engineering (ICDE'04)*, Boston, M.A., March 2004.