Mining Sequential Patterns from Data Streams: a Centroid Approach

Alice Marascu Florent Masseglia

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

Abstract

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 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 only a few methods 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 is 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.

1 Introduction

The problem of mining sequential patterns from a large static database has been widely addressed [2, 11, 14, 18, 10]. 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. These constraints are typical of a new kind of data: 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 in a linear time, 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 [6, 3, 5]. At first, the main

problem was 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 for mining data streams [7]. Then, recent methods [4, 8, 17] introduced different principles for managing the history of frequencies for the extracted patterns. The main idea is that people are often more interested in recent changes. [8] 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 [17] 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 only a few methods 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 of 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. In a web usage pattern, for instance, numerous repetitions of requests for pdf or php files are usual.

In this paper, we propose the SMDS (Sequence Mining in Data Streams) algorithm, which is based on sequences alignment (such as [10, 9] 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.

Our method, proposed in this paper, is able to perform several operations on the sequences of a batch, in only one scan. Those operations include a clustering of the sequences and an alignment of the sequences of each cluster.

The proposed algorithm is implemented and tested over a real dataset. Our data comes 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 behaviour of a Web site's users, also known as Web Usage Mining, is a research field, which consists of 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 behaviour 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 ET*[1] *offers, the ET missions and finally the past ET competitive selection*". We

[1]ET: Engineers, Technicians

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 2. Section 3 gives an overview of two recent methods for extracting frequent patterns in data streams. The framework proposed in this paper is presented in Section 4 and empirical studies are conducted in Section 5.

2 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.

2.1 Sequential Pattern Mining

The problem of mining sequential patterns from a static database DB is defined as follows [2]:

Definition 1 *Let $I = \{i_1, i_2, ..., i_k\}$, be a set of k 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 $< s_1 s_2 \dots s_n >$ where s_j is an itemset. The data-sequence of a customer c is the sequence in DB corresponding to customer c. A sequence $< a_1 a_2 \dots a_n >$ is a subsequence of another sequence $< b_1 b_2 \dots b_m >$ 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=< (c) (d e) (h) > 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.

2.2 From Web Usage Mining to Data Stream Mining

For classic Web usage mining methods, the general idea is similar to the principle proposed in [12]. 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*).

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

Table 1: File obtained after a pre-processing step

Definition 3 *Let Log be a set of server access log entries. An entry g, $g \in Log$, is a tuple $g =< ip_g, ([l_1^g.URL, l_1^g.time] \ ... \ [l_m^g.URL, l_m^g.time]) >$ such that for $1 \leq k \leq m$, $l_k^g.URL$ is the item requested 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 pattern algorithms. In order to extract frequent behaviours 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 1 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, <(a)(c)(b)(c)> (with the file illustrated in table 1 and a minimum support given by the user: 100%). Such a result, once mapped back into URLs, strengthens the discovery of a frequent behaviour, common to n users (with n the threshold given for the data mining process) and also gives the sequence of events composing that behaviour.
Nevertheless, most methods that 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 [13], 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 [13] 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.

3 Related Work

In recent years, many contributions have been proposed for mining patterns in data streams [6, 3, 5, 8, 17, 19, 20]. [1, 15] also consider the problem of mining sequences in streaming data. In this section, we give an overview of [8] and [17].

3.1 FP-Streaming: Frequent Itemset Mining

The authors of [8] describe an approach based on a batch environment and introduce the FP-stream structure for storing frequent patterns and the evolution of their frequency. The authors propose to consider batches of transactions (the update is done only when enough incoming transactions have arrived to form a new batch). For each batch, the frequent patterns are extracted by means of the FP-Growth algorithm applied on a FP-tree structure representing the sequences of the batch. Once the frequent patterns are extracted, the FP-stream structure stores the frequent patterns and their tilted time windows. The tilted time windows give a logarithmic overview on the frequency history of each frequent pattern.

3.2 FTP-DS: Temporal Pattern Mining

In [17] a regression based scheme is given for temporal pattern mining from data streams. The authors propose to record and monitor the frequent temporal patterns extracted. The frequent patterns are represented by a regression-based method. The FTP-DS method introduced in [17] processes the transactions time slot by time slot. When a new slot has been reached, FTP-DS scans the data of the new slot with the previous candidates, proposes a set of new candidates and will scan the data in the next slot with those new candidates. This process is repeated while the data stream is active. FTP-DS is designed for mining inter-transaction patterns. The patterns extracted in this framework are itemsets and this work do not address the extraction of sequences as we propose to do. The authors claim that any type of temporal pattern (causality rules, episodes, sequential patterns) can be handled with proper revisions. However, we discuss the limits of mining sequential patterns from data streams in Section 4.1.

3.3 SPEED: Mining Sequential Streaming Data

In [16] the authors propose to extract sequential patterns from a data stream thanks to an original and efficient tree structure. Their problem is the most similar to ours, since their goal is to extract sequences with a specific threshold and to manage the history of frequencies in a tilted time window manner. The proposed tree structure takes into account the inclusion of sequences embedded in each batch in order to optimize their storage. Actually, this tree offers a "region" technique in order to group sub-sequences of a sequence.

4 The SMDS Algorithm: Motivation and Principle

Our method relies on a batch environment (widely inspired from [8]) and the prefix tree structure of PSP [11] (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.

4.1 Sequential Pattern Mining in a Batch Environment

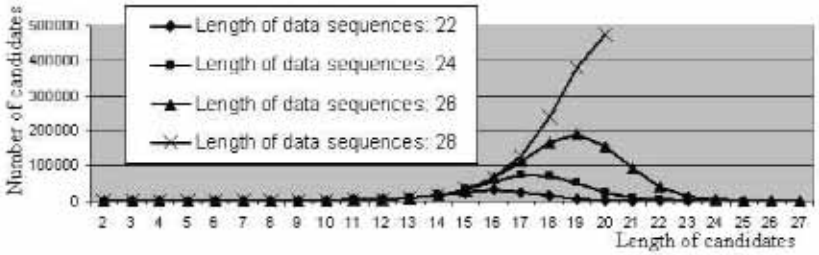


Figure 1: Limits of a batch environment involving PSP

Our method will process the data stream as batches of fixed size. Let $B_1, B_2, ...B_n$ 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 $[B_1..B_n]$ and to store the frequent approximate sequences in a prefix tree structure (inspired from [11]). 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 [11] algorithm. We have tested this algorithm on databases containing only two sequences (s_1 and s_2). Both sequences are equal and contain itemsets having length one. The first database contains 11 repetitions of the itemsets (1)(2) (*i.e.* $s_1 = < (1)(2)(1)(2)...(1)(2) >$, lentgh($s_1$)=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 PrefixSpan[2] [14] 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

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

method depending on the dataset), the integration of such a method in a data stream process is impossible because the worst case may appear in any batch[3].

4.2 Principle

The outline of our method is the following: for each batch of transactions, discovering clusters of users (grouped by behaviour) 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.

For each batch, our clustering algorithm is initialized with only one cluster, which contains the first navigation (first sequence of the batch). SMDS is then able to process a batch of sequences in only one scan. During this scan, the following operations are performed:

1. For each navigation n in the batch, n is compared to each existing cluster. Let c be the cluster such that its centroid ς_c is the most similar to n, then n is inserted into c. If no such cluster has been found then a new cluster is created and n is inserted in this new cluster. The comparison of n (the navigation sequence) with a cluster c is explained in section 4.4.

2. For each cluster c, SMDS computes the centroid ς_c of c incrementally. This step is detailed in section 4.3. This step is very important, since each sequence s to be inserted in a cluster will be compared to the centroid sequence of each cluster.

3. At the end of the batch, the centroid of each cluster c will stand for the extracted knowledge, since it can be considered as a summary of c.

4. Each centroid is inserted into a prefix tree designed to capture the history of the extracted sequential patterns.

4.3 Centroid of a Cluster

The centroid of a cluster is found thanks to the alignment technique of [10] applied to the cluster. This alignment technique uses the dynamic programming. When the first sequence is inserted in the cluster, the centroid is equal to this unique sequence.
The alignment of sequences leads to a weighted sequence represented as follows: $SA = < I_1 : n_1, I_2 : n_2, ..., I_r : n_r >: m$. 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 $(x_{i_1} : m_{i_1}, ...x_{i_t} : m_{i_t})$,where m_i is the number of

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

Step 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
Step 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
Step 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

Figure 2: Different steps of the alignment method with sequences from example 2

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:* $S_1 =< (a,c)\ (e)\ (m,n) >$, $S_2 =< (a,d)\ (e)\ (h)\ (m,n) >$, $S_3 =< (a,b)\ (e)\ (i,j)\ (m) >$, $S_4 =< (b)\ (e)\ (h,i)\ (m) >$. *The steps leading to the alignment of these 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 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. This aligned sequence, which will be the centroid of the cluster from the above example, may not be a true pattern. 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: <(a,b)(e)(h,i)(m,n)> (corresponding to items having a number of occurrences greater or equal to k).

In SMDS, the alignment is updated in a incremental way, for each sequence added to the cluster. For that purpose, we maintain a matrix, which contains the number of items for each sequence and a table of distances between each sequence and the other ones. This is illustrated in figure 3. The matrix (left) stores for each sequence the number of occurrences of each item in this sequence.

Seq	a	b	c
s_1	2	0	1
s_2	1	0	1
⋮			

Seq	$\sum_{i=1}^{n} similMatrix(s, s_i)$
s_1	16
s_2	14
s_n	13
s_3	11
⋮	
s_{n-1}	1

Figure 3: Distances between sequences

For instance, s_1 is a sequence containing the item a twice. The table of distances stores the sum of similarities ($similMatrix$) between each sequence and the other ones. Let s_{1_i} be the number of occurrences of item i in sequence s_1 and let m be the total number of items. $similMatrix$ is found thanks to the matrix in the following way :
$similMatrix(s_1, s_2) - \sum_{i=1}^{m} min(s_{1_i}, s_{2_i})$.

For instance, with two sequences s_1 and s_2 in the matrix given in figure 3 this sum is: $s_{1_a} + s_{2_b} + s_{2_c} - 1 + 0 + 1 - 2$.
Sometimes, the alignment has to be refreshed and cannot be updated incrementally. Let us consider a sequence s_n. First of all, s_n is inserted in the matrix and its distance to the other sequences is computed ($\sum_{i=1}^{n} similMatrix(s_n, s_i)$). s_n is then inserted in the table of distances, with respect to the decreasing order of distances values. For instance, in figure 3, s_n is inserted after s_2. Let r be the rank where s_n is inserted (in our current example, $r = 2$) in c. 0.5 is a parameter specified by the user. There are two possibilities after having inserted s_n:

1. $r > 0.5 \times |c|$. In this case, the alignment is updated incrementally and $\varsigma_c = alignment(\varsigma_c, s_n)$.
2. $r \leq 0.5 \times |c|$. In this case, the centroid has to be refreshed and the alignment is computed again for all the sequences in this cluster.

4.4 Comparing Sequences and Centroids

Let s be the current sequence and C the set of all clusters. SMDS scans C and for each cluster $c \in C$, performs a comparison between s and ς_c (the centroid of c, which is an aligned sequence). This comparison is based on the longest common sequence (LCS) between s and ς_c. The length of the sequence is also taken into account, since it has to be no more than 120% and no less than 80% of the original sequence (first sequence inserted in c).

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: $sim(s_1, s_2) = \frac{LCS(s_1, s_2)}{max(length(s_1), length(s_2))}$.

The distance between two sequences s_1 and s_2 can be defined as:
$dist(s_1, s_2) = 1 - sim(s_1, s_2)$.
Let t be the length of the first sequence inserted in c. The conditions that have to be respected by s are the following:

- $\forall d \subset C / d \neq c, dist(s, \varsigma_c) \leq dist(s, \varsigma_d)$
- $0.8 \times t \leq |s| \leq 1.2 \times t$
- $dist(s, \varsigma_c) < 0.3$

The first condition ensures that s will be affected to the cluster having the centroid that is the most similar to s. The second condition ensures that the clusters will contain sequences of similar length and that the average length of clusters will not vary too much. Finally, the third condition ensures that if no cluster similar to s (with a degree of 70%) is found, then a new cluster is created and s is inserted in this new cluster.

Frequent Sequences Storage and Management

The aligned sequences obtained from the previous step are stored in a prefix tree similar to that of [11]. 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 4 gives an example of a prefix tree where transaction cutting is captured by using labeled edges. Each path from the root to any node in the tree stands for an extracted sequence. The tree from figure 4 contains 6 sequences (<(a c)>, <(a d)>, <(b)>, <(c d)>, <(c)(e)>, <(d)(a)>). 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 4 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 =<(a)> from figure 4. 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 =<(d)(a)>. The support of the aligned sequence s_2 is 3 (meaning that the corresponding cluster contains this aligned sequence with a filter $k = 3$).*

The SMDS algorithm described in this paper is given below.

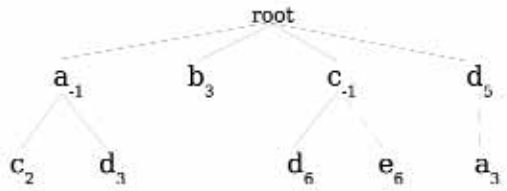


Figure 4: Example of a prefix tree

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 $(c \in C)$ **do**
// 3) Obtain the aligned sequence by applying the filter k on the centroid
$SA_c \leftarrow$ centroid(c, k);
// 4) Store frequent sequences
If (SA_c) **Then** PrefixTree ‹ PrefixTree | SA_c **Endif**
Done
// 5) Update Tilted Time Windows and delete obsolete sequences
TailPruning(PrefixTree);
Done (end Algorithm SMDS);

As we wrote in section 1, 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 [8, 4]. 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 $[f(n,n); f(n-1,n-1); f(n-2,n-3); f(n-4,n-7), ...]$ 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 $f(n \quad 1, n \quad 1)$ and so on. An intermediate windows system

allows to merge windows when needed in order to follow the logarithmic repartition of frequencies. 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.* only store frequencies for the last 100,000 batches, which will require $log_2(100,000) \approx 17$ units of time).

Complexity

In the worst case, the sequences in the batch have the same length: m. The LCS algorithm, involved in the similarity of definition 4 has a time complexity of $O(m^2)$. In the worst case, the clustering algorithm has a time complexity of $O(m^2.n)$ with n the number of clusters. It is well suited for Web navigation patterns and the results obtained (see Section 5) on real datasets (access logs of Inria Sophia-Antipolis) show its effectiveness. The complexity of the alignment algorithm for a batch is $O(p.m^2)$ with p the number of sequences in the batch.

5 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[4].

5.1 Feasibility and Scalability of SMDS

In order to show the efficiency of the SMDS algorithm, we report in figure 5 the time needed to extract the longest approximate sequential pattern on each batch corresponding to the Web usage data (up figure 5) and the synthetic data (down figure 5). 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 4.1). 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 1200 ms to 2000 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

[4]The synthetic data generator is available at http://www.almaden.ibm.com/cs/quest

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 5 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 1700 sequence and SMDS needs 1300 ms in order to extract the approximate sequential patterns.
For the synthetic data we generated batches of 10,000 transactions (corresponding to 470 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 5 (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 an average time of 1.5 seconds (*e.g.* batch number 2).

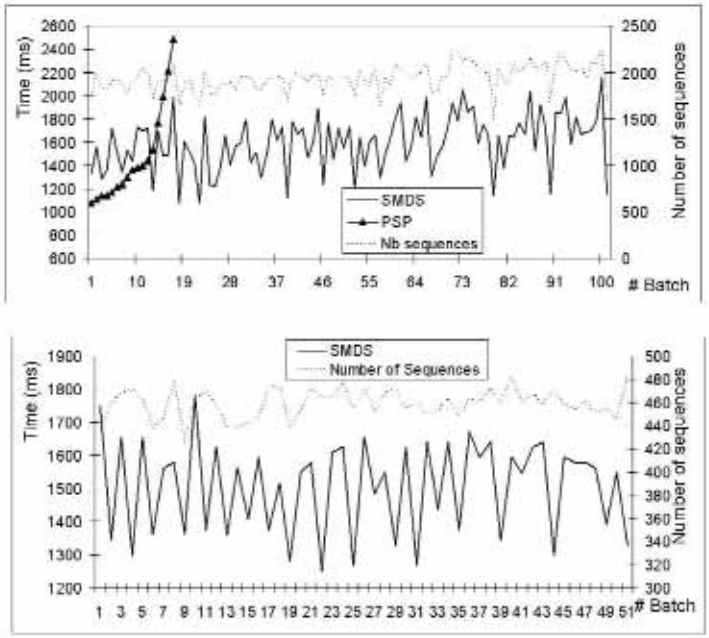


Figure 5: SMDS execution time

5.2 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 discovered patterns can be considered as "rare", but very "confident" (their support is low with respect to the global number of 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% of the batch's size, 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) $k = 30\%$ of the batch's size, cluster size = 100,
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.

5.3 Size of the Batches

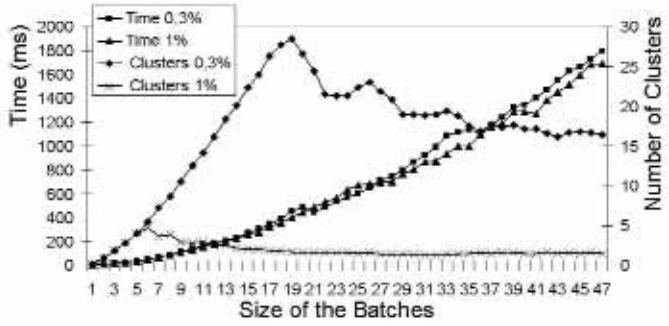


Figure 6: Size of the batches

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 6 the response time when S, the size of the batch, varies from 100 to 4700 sequences. *time 0.3%* stands for the response time. *clusters 0.3%* stands for the number of clusters

extracted by SMDS. 0.3% means that at the end of the batch we only keep cluster having size greater than 0.3% of the batch (*i.e.* $|c| > 0.003 \times S$). It corresponds to $minSize$ introduced in section 4.2. *clusters 1%* stands for the number of clusters c such that : $|c| > 0.01 \times S$. 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 at the beginning, but remains stable after a short number of iterations. The response time is obviously linked to the size of the batches, but it is neraly linear. It is reasonable to say that the final user can choose the size of the batches depending on the desired response time.

5.4 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, $dist(s_1, s_2) \subset [0..1]$ 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 7 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 18%. We can observe that AD varies from 0 (when $|C| = 1$ the center is the only sequence) to 33%. AD thus decreases rapidly to a value between 20% and 25% , which is a good result considering that in figure 7 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 8 the "double average" value (DA) after processing each sequence of the batch. DA is given as follows:

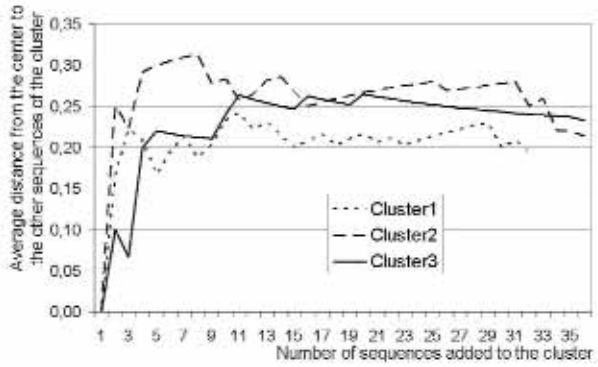


Figure 7: Worst clusters step by step

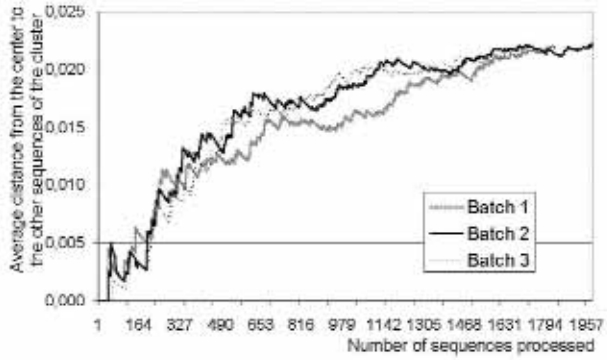


Figure 8: Global distance step by step

let N be the set of clusters, $DA = \frac{\sum_{i \in N} \frac{\sum_{x \in C_i} dist(x, c_i)}{|C|}}{|N|}$ with c_i the center of C_i (the i^{th} cluster). We can observe in figure 8 that for the second batch, DA rapidly grows up to almost 1% after sequence 280, then DA slightly increases to 2.2%. The final value of DA at the end of each batch is given in figure 9. We can observe that the final value of DA is always between 5% and 19%. At the end of the process, the average value of DA is 11% (an average clusters quality of 89%).

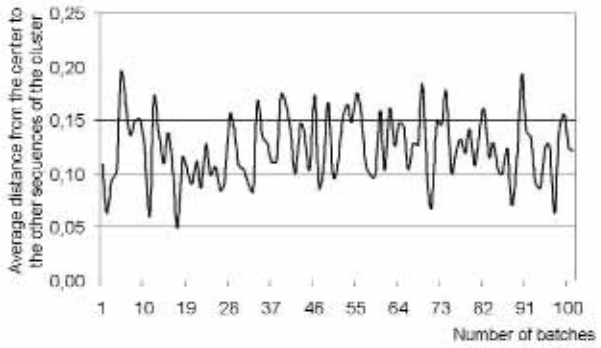


Figure 9: Global distance batch by batch

6 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 relies on a clustering scheme that compares sequences and centroids of clusters. 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] MAIDS project: http://maids.ncsa.uiuc.edu/index.html.

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

[3] 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.

[4] Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multidimensional regression analysis of time-series data streams, 2002.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Birgit Hay, Geert Wets, and Koen Vanhoof. Web Usage Mining by Means of Multidimensional Sequence Alignment Method. In *WEBKDD*, pages 50 65, 2002.

[10] 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.

[11] F. Masseglia, 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.

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

[13] F. Masseglia, 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.

[14] 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.

[15] K. Xu Q. Zheng and S. Ma. When to Update the Sequential Patterns of Stream Data? In *7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 545 550, 2003.

[16] Chedi Raissi, Pascal Poncelet, and Maguelonne Teisseire. Need for SPEED: Mining Sequential Pattens in Data Streams. In *Actes des 21iemes Journees Bases de Donnees Avancees (BDA 2005)*, October 2005.

[17] 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.

[18] 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.

[19] J.H. Chang and W.S. Lee. Efficient Mining Method for Retrieving Sequential Patterns over Online Data Streams. *Journal of Information Science*, Vol.31, No. 5, pages 420–432, 2005.

[20] G. Chen, X. Wu and X. Zhu. Mining Sequential Patterns across Data Streams. *University of Vermont Computer Science Technical Report*, CS-05-04.

Sequence Clustering in Data Streams

A.Marascu F. Masseglia

AxIS Project-Team
INRIA – Sophia Antipolis

Goals:

- Extract sequential patterns from data streams. Applied to: behaviour of a Web site's users.
- Identifying problems arising with this pattern extraction. Particularly the management of their history.

Framework: The SCDS method
(Sequence Clustering in Data Streams)

Data Streams: a few words…

- New elements are generated continuously.
- Data have to be considered as fast as possible.
- No blocking operator can be performed.
- Data can be examined only once.
- Memory usage is restricted.

Sequential Pattern Mining: some definitions.

- *Item*: bought by a customer
- *Transaction*: a customer + an item + a timestamp
- *Sequence*: ordered list of itemsets

- *Data sequence*: stands for the activities of a customer. Let T1, T2, …, T*n* be the transactions of C*j*, the data sequence of C*j* is:
  < itemset(T1) itemset(T2) … itemset(T*n*)>

- *Minimum support* : the minimum number of occurrences of a sequential pattern to be considered as *frequent*.

Illustration:

U1	Publications	Paper1	Paper2	Paper3
U2	Publications	Paper1	List	Paper2
U3	Research	Theme1	Theme3	Theme4
U4	Publications	List	Paper1	List
U5	Research	Theme1	Theme2	Theme3

Question : « Can we find a *behavior* that would be shared by (at least) 40% of the users recorded in the log file? »

behaviour : a series a requests performed during a navigation on the site.

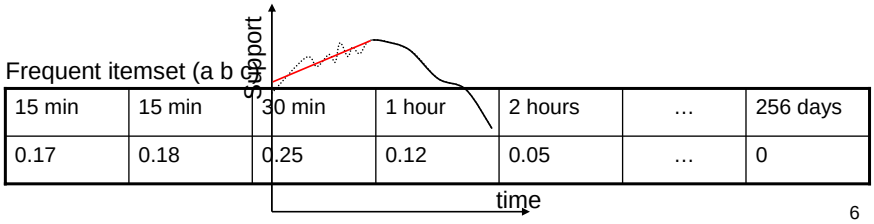
5

European Workshop on Data Stream Analysis
Caserta March 15, 2007

Extracting patterns from data streams

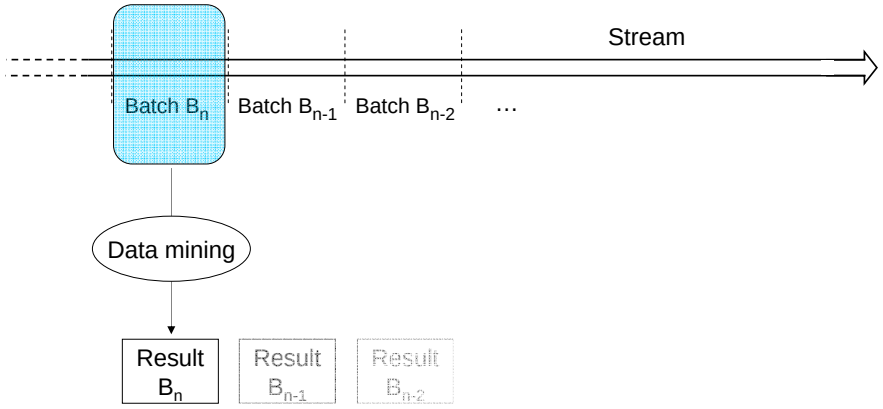
1) Satisfy the constraints of a data stream environment.
High speed algorithms.
Sampling with an estimation of the quality.
etc.

2) Managing the history of frequencies
Logarithmic Tilted Time Window (Han et al.)
Segment Tuning and Relaxation (Teng et al.)

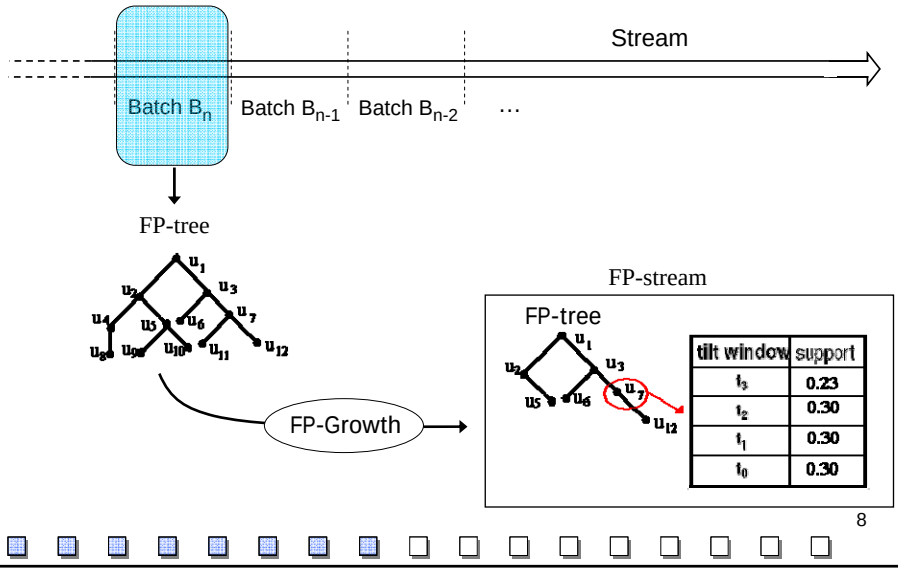


6

Overview



Example: FTPStream



Why is this such a deal to extract sequential patterns from a data stream?

A sequential pattern mining algorithm may be based on:

- Breadth-first search
- Depth-first search

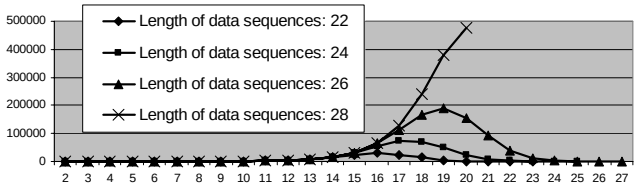
Size of the result!

- Without candidate generation
- Sampling

Size of the batch!

Why is this such a deal to extract sequential patterns from a data stream?

	T1	T2	T3	T4	…	T30
C1	1	2	1	2	…	1
C2	1	2	1	2	…	1



"We have to find the balance between the execution time and the quality of the extracted patterns."

Our proposal relies on two compromises:

1. A greedy sequence clustering algorithm.
2. A sequence alignment method applied to each cluster.

A Greedy Algorithm for clustering streaming sequences

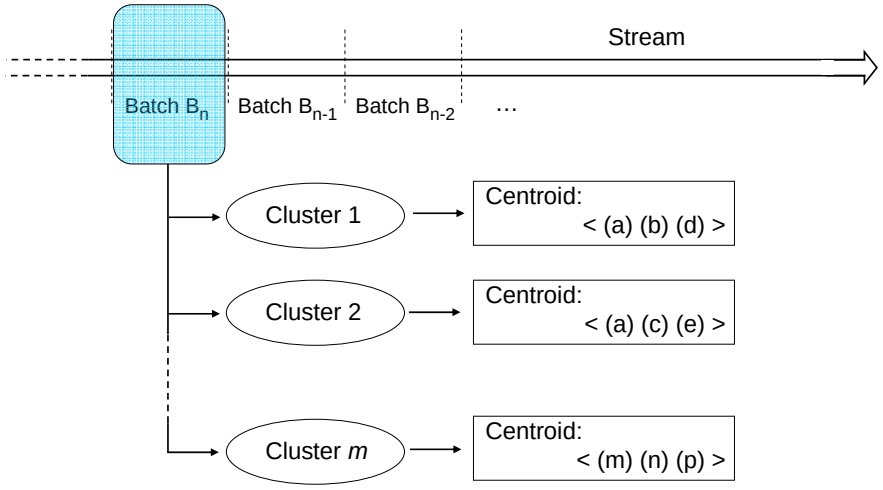
Builds the clusters on the fly.

Let ***S*** be the current sequence.
Scan ***C***, the set of clusters.
Let C_i be the cluster having the most similar centroid to ***S***,
insert (***S***, C_i);
update_centroid(C_i);

If no cluster has been found then
create_cluster (C_j, ***S***);

When "n" sequences have been processed: next batch.

Overview



Sequence alignment for each cluster

The centroid is the result of an alignment applied to the sequences of each cluster.

< (a) (b) (d) >
< (a) (c) (d) >

$\Rightarrow$

< (a) (b) (d) >
< (a) (c) (d) >

< (a:2) (b:1, c:1) (d:2) >

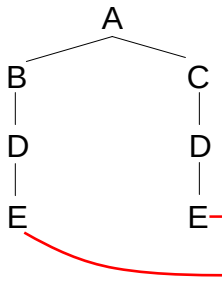
Filter *k* =1: < (a:2) (b:1, c:1) (d:2) >
Filter *k* =2: < (a:2) (d:2) >

Managing the history of the extracted patterns

- On static databases, the knowledge is stable
- On data streams, the knowledge is evolving with the stream

Ongoing work: an incremental clustering.

Motivation: The division of the stream into batches “blurs” the history of the frequent patterns.



Sequence	15 min	15 min	30 min	1h	…	256 jours
ACDE	5%	0%	4%	0%		0%
ABDE	0%	3%	0%	1%		0%
A(BC)DE	5%	3%	4%	1%		0%

We propose to perform an incremental clustering in order to maintain a coherent history.

- **Idea** : have the cluster evolving.
- **Objective** : be independent from slight variations when managing the history of extracted patterns.
- **Principle** : keep the centroid (aligned sequence) of the clusters from one batch to another.

15 min	15 min	30 min	1h	…
A(BC)D	A(BC)D	A(BC)DE	A(BC)E	…
3%	4%	2%	2%	…

17

A few questions motivating this work:

- Is data mining able to help summarizing a stream?

- What should this summary look like?

- Where does the approximation stop?

18