Sampling For Sequential Pattern Mining: From Static Databases to Data Streams

Chedy Raïssi, Pascal Poncelet

Sampling for Sequential Pattern Mining: From Static Databases to Data Streams

Chedy Raïssi
LIRMM, EMA-LGI2P/Site EERIE
161 rue Ada
34392 Montpellier Cedex 5, France
raissi@lirmm.fr

Pascal Poncelet
EMA-LGI2P/Site EERIE
Parc Scientifique Georges Besse
30035 Nîmes Cedex, France
Pascal.Poncelet@ema.fr

Abstract

Sequential pattern mining is an active field in the domain of knowledge discovery. Recently, with the constant progress in hardware technologies, real-world databases tend to grow larger and the hypothesis that a database can be loaded into main-memory for sequential pattern mining purpose is no longer valid. Furthermore, the new model of data as a continuous and potentially infinite flow, known as data stream model, call for a pre-processing step to ease the mining operations. Since the database size is the most influential factor for mining algorithms we examine the use of sampling over static databases to get approximate mining results with an upper bound on the error rate. Moreover, we extend these sampling analysis and present an algorithm based on reservoir sampling to cope with sequential pattern mining over data streams. We demonstrate with empirical results that our sampling methods are efficient and that sequence mining remains accurate over static databases and data streams.

1 Introduction

Sequential Pattern mining is one of the most active and challenging field in the domain of knowledge discovery. It allows the discovery of frequent sequences and helps identifying relations between itemsets in a transactional database. However, sequential pattern mining is a difficult task as the search space for this problem is huge [13]. To overcome this problem, researchers developed mining algorithms that use levelwise generate-and-prune strategies or a multiple database projections approach as heuristics.

Lately, thanks to evolutions in hardware technologies, companies and organisations are able to generate and store very large volumes of data from different sources: network monitoring with, for instance, TCP/IP traffic; financial transactions such as credit card customers operations, medical records and a wide variety of sensor logs. Therefore, the real-world databases can scale up to gigabytes and terabytes. Consequently, one of the main hypothesis used in pattern mining stating that the database can hold in main-memory is obviously challenged. Besides, if the database is very dense and the data is highly-correlated, the mining algorithm is likely to fail even with a high support value.

The same limits apply to the new model, called data stream, where the data appears in a continuous, high-speed and unbounded flow, where it is often impossible to mine patterns with classical algorithms requiring multiple scans over the database. As a consequence new approaches were proposed to mine itemsets based on the *landmark*, *sliding windows* or *time-fading* models [6, 4, 10]. However, few works focused on sequential patterns extraction over data streams.

The two major contributions of this work are: *(i)* A new pre-processing approach to reduce static database access by constructing a random sample before applying the mining algorithm. *(ii)* A new approach to mine sequential patterns based on the maintenance of a synopsis of the data stream.

The rest of this paper is organized as follows. In Section 2 we briefly introduce the sequential pattern mining agorithms and the different synopses approaches. Preliminary concepts and problem description are introduced in Section 3. Section 4 deals with the static database sampling and presents proveable error guarantees. Section 5 extends this sampling analysis to the data stream model. The experiments and their results are described and discussed in Section 6. In the last section we give some conclusions and perspectives for future work.

2 Related Work

Sequential Pattern mining problem was introduced by Agrawal and Srikant [9]. Different efficient algorithms like PrefixSpan [7], SPADE [13] and SPAM [2] were proposed later, each one of them using different mining heuristics. However, these algorithms implementations fail to mine multi-gigabytes databases.

The large and growing number of data in static databases or in data streams results in new challenging space and time constraints for mining algorithms. In these cases, it is acceptable to get approximate solutions. In other words, *one has to trade off accuracy against efficiency*. A number of synopsis structures have been developed in recent years like sketches, sampling, wavelets and histograms. These different synopses structures share the same properties of broad applicability, space efficiency, and robustness. Furthermore, all these synopsis methods respect a one-pass constraint which makes them perfectly suitable for the data stream model.

The method presented in this work belongs to the class of reservoir sampling and is very easy to understand as it generates a sample of the original data representation. Reservoir sampling was first introduced in [12]. However, as the data set length increases, the probability of the insertion of a data point in the reservoir reduces. This is a clear disadvantage for data stream mining tasks because users may consider that recent information provided by the stream is the most relevant. In [3], the authors adapted the reservoir sampling method to the sliding window model over a data stream. However, this solution does not allow to sample from different lengths of the stream history. One solution proposed recently in [1] was the use of an exponential bias function to regulate the choice of the stream sample.

3 Preliminary concepts and problem description

Let $\mathcal{D}$ be a database of customer transactions where each transaction T consists of: a customer identifier, denoted by C_{id}; a transaction time, denoted by $time$ and a set of items (called *itemset*) involved in the transaction, denoted by it.

Definition 1 (Sequence) *Let $\mathcal{I} = \{i_1, i_2 ... i_m\}$ be a finite set of literals called items. An itemset is a non-empty set of items. A sequence S is a set of itemsets ordered according to their timestamp. It is denoted by $< it_1\ it_2\ ...\ it_n >$, where it_j, $j \in 1...n$, is an itemset.*

A sequence $S' =< s'_1\ s'_2\ ...\ s'_n >$ is a subsequence of another sequence $S =< s_1\ s_2\ ...\ s_m >$, denoted $S' \preceq S$, if $\exists\, i_1 < i_2 < ...\, i_j\, ... < i_n$ such that $s'_1 \subseteq s_{i_1}$, $s'_2 \subseteq si_2$,... $s'_n \subseteq si_n$.

Definition 2 (Support) *Let C_{trans} be the ordered list of transactions for a single customer C. The support of a sequence S in a transaction database $\mathcal{D}$, denoted by $Sup(S, \mathcal{D})$, is defined as:*

$$Sup(S, \mathcal{D}) = \frac{|\{C \in \mathcal{D} | S \preceq C_{trans}\}|}{|\{C \in \mathcal{D}|}$$

Given a user-defined minimal support threshold, denoted σ, the problem of sequential pattern mining is the extraction of all the sequences S in $\mathcal{D}$ such that $Sup(S, \mathcal{D}) \geq \sigma$.

Example 1 *Consider the following database with $\mathcal{I} = \{a, b, c, d\}$. There are 3 different identifiers C_1, C_2 and C_3. Let the minimal support threshold be $\sigma = \frac{2}{3}$, the frequent sequences in $\mathcal{D}$ are: $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, $\langle (ab) \rangle$, $\langle (ad) \rangle$, $\langle (a)(c) \rangle$ and $\langle (d)(c) \rangle$*

$\mathcal{D} =$

C_1	T_1	a,b,c,d
	T_2	a,c
C_2	T_1	a,b
C_3	T_1	a,d
	T_2	c

Biased reservoir sampling was introduced in [1] and is based on the following fact: overtime, in many cases, the data stream distribution may evolve and the original reservoir sampling results may become irrelevant. A solution is to use a bias function to regulate the sampling in order to focus on recent or old behaviors in the stream following application specific constraints. The use of a bias function guarantees that recent points arriving over the stream have higher probabilities to be inserted in the reservoir. The author in [1] exploit some properties of a class of memory-less bias functions: the exponential bias function which is defined as follow: $f(r, t) = e^{-\lambda(t-r)}$ with parameter λ being the bias rate. The inclusion of such a bias function enables the use of a simple and efficient replacement algorithm.

4 Static data set sampling for sequence mining

In this section, we discuss the usefulness of the sampling method as a pre-processing step for sequence mining. We intuite that we can focus on a sample from the real data set in order to get a faster and easier mining task.

The first question that one has to answer when working on samples for mining tasks is: *how accurate my sample is compared to my original data set?* We answer to this question by theoretically analyzing the accuracy of our sample given a user defined error threshold. Notice that a similar approach was used for itemset mining in [11].

Definition 3 (Error rate) *Let $\mathcal{D}$ be a database of customer transactions and $\mathcal{S}_\mathcal{D}$ a random sample generated from $\mathcal{D}$. Let s be a sequence from $\mathcal{D}$. The absolute error rate in terms of support estimation, denoted $e(s, \mathcal{S}_\mathcal{D})$, is defined as: $e(s, \mathcal{S}_\mathcal{D}) = |Sup(s, \mathcal{S}_\mathcal{D}) - Sup(s, \mathcal{D})|$*

Let X_i be a Bernoulli random variable with $Pr[X_i = 1] = p_i$ if the i^{th} customer supports s and $Pr[X_i = 0] = 1 - p_i$ if not. Let $X(s, \mathcal{S}_\mathcal{D}) = \sum_i^{|\mathcal{S}_\mathcal{D}|} X_i$. Clearly, $X(s, \mathcal{S}_\mathcal{D})$ is the number of customers in the sample $\mathcal{S}_\mathcal{D}$ that support the sequence s. It can be written as $X(s, \mathcal{S}_\mathcal{D}) = Sup(s, \mathcal{S}_\mathcal{D}).|\mathcal{S}_\mathcal{D}|$. Then the expected number of customers that support the sequence s in the sample is $E[X(s, \mathcal{S}_\mathcal{D})] = Sup(s, \mathcal{D}).|\mathcal{S}_\mathcal{D}|$.

We would like to estimate the probability that our error rate gets higher than a user defined threshold ϵ, denoted $Pr[e(s, \mathcal{S}_\mathcal{D}) > \epsilon]$.

To answer this question we use Hoeffding concentration inequalities [5]. Basically, the concentration inequalities are meant to give an accurate prediction of the actual value of a random variable by bounding the error term (from the expected value) with an associated probability. These inequalities usually require the random variable to be decomposed as a sum of independent random variables.

This assumption holds in our problem and by using Hoeffding concentration inequalities, the following theorem gives us a lower bound for the size of the reservoir given ϵ and a maximum probability δ that the error rate exceeds ϵ (called also an (ϵ, δ)-approximation):

Theorem 1 *Given a sequence s then $Pr[e(s, \mathcal{S}_\mathcal{D}) > \epsilon] \leq \delta$ if the reservoir size $|\mathcal{S}_\mathcal{D}| \geq \ln(\frac{2}{\delta})\frac{1}{2\epsilon^2}$*

Proof 1

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

The Hoeffding concentration inequalites state that for n independant bounded random variables $X_1, X_2, \ldots, X_n$ lying in the range $[a, b]$, the sum S of these variables with positive values of t satisfies the following:

$$Pr[\ |S - E[S]\ | \geq nt] \leq 2exp\left(\frac{-2n^2t^2}{n(b-a)^2}\right)$$

Thus, $Pr[\ |X(s, \mathcal{S}_\mathcal{D}) - E[X(s, \mathcal{S}_\mathcal{D})]\ |\ > \epsilon.|\mathcal{S}_\mathcal{D}|\ ] \leq \delta$ with $\delta = 2e^{-2\epsilon^2.|\mathcal{S}_\mathcal{D}|}$ □

Table 1 gives some examples of sample sizes in term of customers for different values of ϵ and δ.

ϵ	δ	$\lvert\mathcal{S}_\mathcal{D}\rvert$
0.01	0.01	26492
0.01	0.001	38005
0.01	0.0001	49518
0.001	0.27	1000000

Table 1. Different sample sizes for ϵ and δ

5 Sampling over data streams for sequence mining

One of the main problem for stream mining is that the length of the stream is unknown. Therefore, we need to maintain a dynamic sample of the stream. In this section, we extend the previous results for sampling static data sets of customers sequences to the challenging case of data streams. Our goal is to approximate sequences support from the dynamic sample. However, and unlike in the static database sampling, customers transactions list size can *grow* over time in the data stream and thus, we need to find a way to keep the size of the sampled customers transactions list *under control*. Moreover, and from an application point-of-view, we would like to have a choice between biased and unbiased results.

In order to do so, our sampling algorithm should respect two conditions: *(i)* There must be a lower bound on the size of the sample. According to the previous results from section 4, we would like our approach to produce accurate mining results from a dynamic sample. This can also be achieved by using an (ϵ, δ)-approximation, like the one for static databases, combined with an exponential biased reservoir sampling method. Note also that the reservoir size is defined in term of customers number and not in term of transaction numbers. *(ii)* The insertion and replacement operations, essential for the reservoir updating, must be done at customers level *and* at transactions level. This is necessary to control the size of the itemsets for each customer in the reservoir.

In our data stream model, on every timestamp, a new data point arrives in the stream and a data point is defined as a couple of customer and its associated transaction.

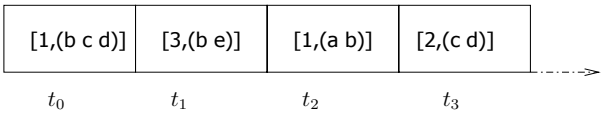

Figure 1. Stream model with 4 data points

Our approach is a simple replacement algorithm using an exponential bias function that regulates the sampling of customers and their transactions over a stream. We start with an empty reservoir of capacity $\frac{1}{\lambda}$ (where λ is the bias rate of our exponential bias function) and each data point arriving from the stream is deterministically added to the reservoir by flipping a coin, either as a simple insertion into the reservoir, or as a replacement of a customer and all its related transactions. Note that this is enough to sample customers, but we also need to bound the size of the list of transactions

belonging to each customer present in the reservoir. In order to do so, we use a sliding window mechanism. A sliding-window can be defined as a *sequence-based window of size* k consisting of the k most recent data elements that arrived on the stream or as a *timestamp-based window of duration* t containing all the data points whose arrival timestamp is within a time interval t of the current time.

In our approach we use a sequence-based window to retain the latest and most recent transactions for a given customer in the sample. This is useful to get accurate mining tasks over recent horizons of the stream. Besides, the exponential biased function that we use enables the user to choose the desired size of the reservoir (with some constraints on the (ϵ, δ)-approximation) and thus a relevant sample may be maintained in main memory, depending on the application needs.

The lower bound for the size of the reservoir can be deduced from the following corollary. This corollary highlights the direct link between the bias rate λ and $the(\epsilon, \delta)$-approximation from theorem 1.

Corollary 1 *Let λ be the bias rate, ϵ be an error threshold and δ the maximum probability that $e(s, \mathcal{S}_\mathcal{D}) > \epsilon$, then:*

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

Proof 2 *By [1], for a stream of length t, let $R(t)$ be the maximal reservoir requirement for a random sample from a stream which satisfies the exponential bias function $f(r,t) = e^{-\lambda(t-r)}$, then $R(t) \leq \frac{1}{\lambda}$. The whole reservoir is used as our sample for the mining task, thus $R(t) = |\mathcal{S}_\mathcal{D}|$. On substitution in theorem 1, the result follows.*

A problem arises in our algorithm for the specific case of customers replacement from the reservoir: we need to detect if a customer was already in the sample, because reinserting it without any sanity-check will introduce inconsistency at the mining step. The customer sequence-based window will be incomplete: there will be missing transactions that were discarded when the customer was previously replaced. To avoid this inconsistency the data point needs to be ignored. On the other hand, ignoring data points over the whole stream is no good either, because this will bias the sample towards customers that replaced others in the reservoir. To overcome this problem, we introduce the idea of a customers black list that gets *refreshed* at each replacement in the reservoir. The black list is updated at each sliding of the window in order to allow the ignored customers to be able to get back into the reservoir. Example 2 presents a case of sequence-based window consistency checking using the black list. The full algorithm is described in Algorithm 1.

Example 2 *Let $\mathcal{R}$ be the reservoir with maximal size $= 2$ and $W = 4$ the sequence-based window size for the transactions in the reservoir, at an instant t:*

$Customers$	*Sequence-based window*			
C_1	(ab)	(cd)	(ab)	(d)
C_2	$(abcde)$	(a)	(b)	

Suppose that the next point from the stream at time $t+1$, which is the tuple formed by the the customer C_3 and its transaction (de), replaces customer C_2 in the reservoir after a successful coin flip:

$Customers$	Sequence-based window			
C_1	(ab)	(cd)	(ab)	(d)
C_3	(de)			

$BlackList$	
C_2	3

The black list is updated with a new entry for customer C_3 for 3 window slides. It is obvious that for consistency reasons our algorithm cannot reintroduce in the reservoir any data point with customer C_2 because we already discarded the sequence of 3 itemsets $(abcde)(a)(b)$.

Data points with customer C_2 can be reintroduced in the sample only when its counter in the black list is equal to 0. All customers in the black list have their counters decremented by one at each window sliding.

One observation about this algorithm is that it can be implemented very easily. However, in the section 5 we assumed that Algorithm 1 does indeed achieve exponential bias with parameter λ. We need now to prove this assumption. Like in [1], we show that the replacement policy in our algorithm results in a biased sample of size $|\mathcal{S}_\mathcal{D}| = n$ which respects the exponential bias behaviour with $\lambda = 1/n$. The proof appears in the full technical report [8].

Theorem 2 *The probability of a data point r to be still present in the reservoir of maximum size n at the time t is approximately equal to $f(r,t) = e^{-\lambda(t-r)}$*

6 Experimental Evaluation

In this section, we present the experiments we conducted in order to evaluate the feasability of our sampling techniques as a pre-processing step before sequential pattern mining (our complete set of results is in the full paper [8]).

The experiments were performed on a Core-Duo 2.16 Ghz MacBook Pro with 1GB of main memory, running Mac OS X 10.4.6. Sequential pattern mining is performed with the public PrefixSpan [7] implementation[1]. We performed

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

Data Set	Items	Avg. size of transactions	Avg. # trans per customer	# of clients	size in GB
CL1MTR2.5SL50IT10K	10000	2.5	50	1000000	1.05
CL6MTR2.5SL10IT20K	20000	2.5	10	6000000	0.686

Data Set Name	Number of frequent sequential patterns	Support	Processing time	Required memory
CL6MTR2.5SL10IT20K	5503	0.3%	523.969 s.	685.821 MB

Table 2. Different data sets used for the experiments and mining results on the static data set without sampling.

Algorithm 1: RESERVOIR SAMPLING FOR SEQUENTIAL PATTERNS algorithm

Data: Reservoir $\mathcal{S}_{\mathcal{D}}$;Bias rate λ;sequence-based window size $|W|$;Data point T;Black List $\mathcal{BL}$
; **Result**: Updated Reservoir $\mathcal{S}_{\mathcal{D}}$ after processing the data point T

begin
  if $T.C_i \notin \mathcal{BL}$ **then**
    if $T.C_i \notin \mathcal{S}_{\mathcal{D}}$ **then**
      // Deterministic insertion
      // of the data point T
      $Coin \leftarrow Random(0,1)$;
      // $F(t) \in [0,1]$ is the fraction
      // of the reservoir filled
      // at instant t
      if $Coin \leq F(t)$ **then**
        // Success case:
        // replacement
        $pos \leftarrow Random(0,q)$;
        $Replace(C_{pos}, T.C_i)$;
        //Add $T.C_{pos}$ to $\mathcal{BL}$.
        $\mathcal{BL}.add(T.C_{pos})$;
      else
        //Failure case:
        //Add directly T
        $Add(T, \mathcal{S}_{\mathcal{D}})$;
    else
      //C_i already in $\mathcal{S}_{\mathcal{D}}$
      //Insert T in C_i's
      //sequence-based window
      Insert(C_i,T);
      // Check if there is a slide
      if $C_i.window.size > |W|$ **then**
        $slideAllWindows()$;
        $Update(\mathcal{BL})$;
end

several tests with different synthetic data sets that were generated with QUEST [2] software (see table 2). In order to sample static databases we used and implemented the reservoir sampling algorithm from [12]. All the implementations are done in C++.

We tested the correctness of our theoretical results presented in section 4. We used in these experiments the data set CL6MTR2.5SL10IT20K to compare the accuracy of our sampling based on the (ϵ, δ)-approximation from Theorem 1. The results of the different experimentations are listed in Tables 2 and 3. We considered samples of different sizes varying from 25000 to 500000 customers. These samples are built using reservoir sampling. One interesting observation is that these small size samples are easily handled in main-memory by the PrefixSpan implementation. We repeated these experiments 5 times for each sample and each value presented in the results is the average value computed over the 5 runs. The error column shows the number of frequent sequential patterns that had their absolute error rate such that $e(s, \mathcal{S}_{\mathcal{D}}) > \epsilon$. As estimated by our (ϵ, δ)-approximations (cf. Table 1), the results remain very accurate and precise even for small samples, the mining speed is of course faster over the samples and we can push the extraction for low support values that could not be mined from the original data set. In order to test Algorithm 1, we used the data set CL1MTR2.5SL50IT10K with different parameters: the bias rate value ($\lambda = 0.00001$ and 0.000004) and the sequence-based window size ($|W| = 10$ and 2). In these experimentations, we focus on the size of the reservoir and on the size of the black list. Figure 3 shows that even if the number of black listed customers vary drastically over time, it remains bounded (less than 3000 and 100 customers) because of the updatings that are made at each window sliding. Notice that if the sequence-based window size is small, the black list tends to be very small also, this is due to the frequent slidings implying a rapid update of the black list. Besides, the reservoir size is stable and bounded in term of required memory (less than 27 and 16MB) in the as it can

[2] http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/

$\lvert\mathcal{S}_\mathcal{D}\rvert$	# of seq. patterns	Error	Proc. time
25000	5830	342	1.6 s.
38005	5239	271	1.608 s.
50000	5321	184	1.903 s.
100000	5432	75	2.725 s.
500000	5531	31	9.854 s.
$\lvert\mathcal{S}_\mathcal{D}\rvert$	Required Memory	Sample size (Mb)	
25000	3.070 MB	3	
38005	4.503 MB	4.4	
50000	5.997 MB	5.9	
100000	11.831 MB	11.8	
500000	58.947 MB	58.8	

Table 3. Mining results for different sample sizes for data set CL6MTR2.5SL10IT20K

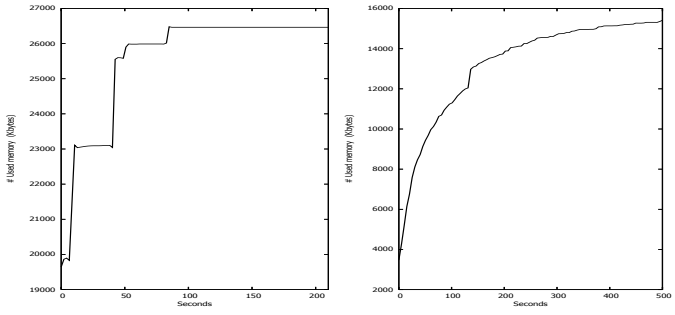

Figure 2. Memory requirement in order to sample data set CL1MTR2.5SL50IT10K

be seen in Figure 2.

7 Conclusion

We have described a novel approach to ease the sequential pattern mining problem. We have detailed the theoretical aspects to sample static data sets and we extended this work to take into account the data streams model by using a biased reservoir sampling approach in order to build a dynamic sample. We showed that these pre-processing sampling techniques give good results on the data sets we considered. We provided compelling evidence that it is possible to obtain accurate and fast results for sequential pattern mining using small samples. Furthermore, we showed that our stream sampling algorithm is efficient, making it suitable for use on real-world data streams like network traffic. Finally, our results shows the potential of further work on sampling for sequential pattern mining and specially in the new challenging data streams model.

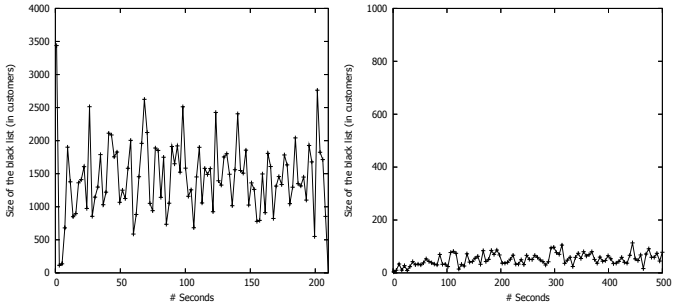

Figure 3. Black list size in term of customers for data set CL1MTR2.5SL50IT10K

References

[1] C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, pages 607–618. ACM, 2006.

[2] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *KDD*, pages 429–435. ACM, 2002.

[3] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data, 2002.

[4] Y. Chi, H. Wang, P. Yu, and R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *ICDM 04*, pages 59–66, 2004.

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

[6] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB 02*, pages 346–357, 2002.

[7] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. Knowl. Data Eng.*, 16(11):1424–1440, 2004.

[8] C. Raïssi and P. Poncelet. Sampling for sequential pattern mining : From static databases to data streams. Technical report, LIRMM Laboratory - Univ. of Montpellier, 2007.

[9] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE 95*, pages 3–14, 1995.

[10] W.-G. Teng, M.-S. Chen, and P. Yu. A regression-based temporal patterns mining schema for data streams. In *VLDB 03*, pages 93–104, 2003.

[11] H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145. Morgan Kaufmann, 1996.

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

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