Data Min Knowl Disc (2006) 13:193–217
DOI 10.1007/s10618-006-0042-x

Mining top-*K* frequent itemsets from data streams

Raymond Chi-Wing Wong · Ada Wai-Chee Fu

Received: 29 April 2005 / Accepted: 1 February 2006 / Published online: 31 May 2006

Abstract Frequent pattern mining on data streams is of interest recently. However, it is not easy for users to determine a proper frequency threshold. It is more reasonable to ask users to set a bound on the result size. We study the problem of mining top *K* frequent itemsets in data streams. We introduce a method based on the Chernoff bound with a guarantee of the output quality and also a bound on the memory usage. We also propose an algorithm based on the Lossy Counting Algorithm. In most of the experiments of the two proposed algorithms, we obtain perfect solutions and the memory space occupied by our algorithms is very small. Besides, we also propose the adapted approach of these two algorithms in order to handle the case when we are interested in mining the data in a sliding window. The experiments show that the results are accurate.

Keywords Data mining algorithm · Data stream · Top *K* frequent itemset mining · Sliding window · Chernoff bound · Probabilistic algorithm

1. Introduction

A data stream is an unbounded sequence of data arriving at high speed. Data streams are found in a diversity of applications including network monitoring, financial monitoring such as stock tickers, sensor networks, web logging and large retail store transactions. Data stream mining is deemed useful in all of these applications. For example, looking for hot lists can help to detect attacks in a network. There are two main challenges in handling data streams: (1) since the data is unbounded and arriving at high speed, the volume of data will quickly exceed normal storage capacity and one cannot hope to record every single data in the data

R. C.-W. Wong (✉) · A. W.-C. Fu
Department of Computer Science and Engineering, The Chinese University of Hong Kong,
Ho Sin-Hang Engineering Building, Shatin N.T. Hong Kong
e-mail: cwwong@cse.cuhk.edu.hk

A. W.-C. Fu
e-mail: adafu@cse.cuhk.edu.hk

stream; (2) the mining results are usually expected to be returned in real time. Traditional data mining processes are typically not real-time, and the data is assumed to be kept in total in some storage. Therefore, new methods are designed to handle data streams.

Mining frequent patterns (or itemsets) has been studied extensively in the literature of data mining. Frequent patterns can be very meaningful in data streams. In network monitoring, frequent patterns can correspond to excessive traffic which could be an indicator for network attack. In sales transactions, frequent patterns relate to the top selling products in a market, and possibly their relationships. If we consider that the data stream consists of transactions, each being a set of items, then the previous problem definition of mining frequent patterns is the following:

Problem A: Given a set of transactions, find all patterns with frequency above a threshold s.

However, this definition suffers from a serious problem—the requirement of a suitable support threshold value s from users. If s is set too small, we generate a lot of frequent patterns. If s is set too high, no frequent patterns may be generated. The choice of the threshold s is a difficult task. In Cheung and Fu (2002), it is found that in different data sets or even in different subsets of the same data set, the proper values of s that give a reasonable amount of results can differ by an order of magnitude. In most previous works on mining frequent patterns from data streams, much effort is spent on providing some guarantee so that the resulting patterns have frequencies with error at most ϵ, where ϵ is an error parameter. However, if a user cannot determine a proper s in the first place, such a guarantee would not have much value. From the above observation, it is better to remove the assumption of s. Instead we can ask user to determine the number of patterns to be returned. Thus we have the following problem:

Problem B: Let an l-itemset be a set of l items. Given a set of transactions and two positive integers K and L. For each l, where $1 \leq l \leq L$, find all l-itemsets with frequencies greater than or equal to f_l, where f_l is the frequency of the K-th frequent l-itemset[1] in the data streams. We call these itemsets the top K-frequent itemsets.[2]

If we know the frequency f_l of the K-th most frequent l-itemset, we can set f_l as the frequency threshold and apply the ideas of some method for Problem A to find the top K l-itemsets. In data stream mining, one typically cannot get the exact frequencies, but has to make an estimation. For Problem A, one need only estimate the itemset frequencies. For Problem B, one not only has to estimate the frequencies of itemsets but also has to guess the threshold frequency f_l in order to cut off the infrequent patterns.

Our contribution can be summarized as follows.

(1) We switch from Problem A (Manku and Motwani, 2002; Teng et al., 2003; Giannella et al., 2003; Yu et al., 2004; Lee and Chen, 2001), to the more realistic Problem B of top K frequent itemsets mining, where a user does not need to determine any frequency threshold. We believe that this problem is more practical and it is more difficult because, without a support threshold, there is an extra dimension of guessing in the algorithm and hence a source of error.

[1] To obtain the K-th frequent itemset, we assume all itemsets are sorted in descending order by their frequencies. For itemsets with the same frequencies, the order is arbitrary. The K-th itemset in such a sorted list is a K-th frequent itemset.

[2] Note that there may be more than K itemsets in the output when there are more than one itemset with frequency f_l.

(2) We propose two algorithms to tackle Problem B and derive some analytical bounds on the error and memory consumption.
(3) The two proposed methods obtain near 100% accuracy in the overwhelming majority cases of our experiments. The memory requirement is reduced many folds compared to a naive approach.
(4) We modify the problem of mining the data over an entire data stream to one with a sliding window. We also modify the two proposed methods in order to handle Problem B. The experiments show that the proposed algorithms achieve high accuracy and fast execution time.

2. Chernoff-based algorithm versus top-*K* lossy counting algorithm

We introduce two algorithms in this paper, namely the Chernoff-based Algorithm and the Top-*K* Lossy Counting Algorithm. Let us compare the two methods here.

The *Chernoff-based Algorithm* is controlled by two parameters ϵ and δ for error bounds and reliability. ϵ is determined automatically when data arrives. It will be close to zero when more and more data is read. The other parameter δ determines the confidence that the results returned are the true top K frequent itemsets.

Although the Chernoff-based Algorithm assumes data independence, there are techniques to handle dependencies in data (see Section 3.3) so that the algorithm can still be applied.

The *Top-K Lossy Counting Algorithm* does not make any assumption about data independence. It is controlled by an error parameter ϵ set by the users. From analysis in Manku and Motwani (2002), if ϵ is set to a value smaller than the support of the K-th frequent itemset, the result will be highly accurate. Otherwise, many correct results may be missed. Setting ϵ too high yields inaccurate results while setting it too low leads to excessive memory consumption. This is similar in nature to the setting of the support threshold of mining frequent itemset.

The two algorithms have their own strengths and weaknesses. It is expected that the Chernoff-based Algorithm returns accurate results when the data is independent. However, even with data dependence, the algorithm can return good results by applying the methods introduced in Section 3.3. The Lossy Counting Algorithm makes no assumption on the data distribution. However, we assume an input parameter ϵ. If we can set a reasonable value for ϵ, accurate result, fast computation and low memory utilization can be achieved, which is shown in Section 6.

3. Chernoff-based algorithm

In this section, we develop an algorithm based on the Chernoff bound for mining top K frequent itemsets.

The Chernoff bound is a bound on the probability that a random variable deviates by a certain amount from its expectation. Consider a series of observations, $o_1, o_2, \ldots, o_n, o_{n+1}, \ldots$ where each o_i is a Bernoulli trial. Observation o_i can be interpreted as whether the outcome is X or not. It is convenient to relate this to coin flips, where observation o_i is interpreted as whether the result is a head, so that $o_i = 1$ for head, and $o_i = 0$ for tail. Let $\hat{x}$ be the number of heads in n observations $o_1, \ldots o_n$, and $\tilde{x} = \hat{x}/n$. Let x be the probability of a head in a coin flip. Formally, we write $Pr[o_i = 1] = x$. x is also the expected fraction

of heads in n observations. For $\gamma > 0$, the Chernoff bound states that

$$Pr\{|\tilde{x} - x| \geq x\gamma\} \leq 2e^{\frac{-nx\gamma^2}{2}}$$

Replacing $x\gamma$ with ϵ, $P\gamma\{|\tilde{x} - x| \geq \epsilon\} \leq 2e^{\frac{-n\epsilon^2}{2x}}$. Let us set the right hand side of this equation to δ, where δ is the *reliability parameter*. We see that, with probability $\leq\delta$, the *running average* $\widetilde{x}$ is beyond $\pm\epsilon$ of x, where ϵ is the error parameter and

$$\epsilon = \sqrt{\frac{2x\ \ln(2/\delta)}{n}} \quad (1)$$

From Eq. (1), if δ is fixed, then ϵ will become smaller when n increases. The basic strategy is therefore to fix δ and force ϵ to zero when n (the number of data read) increases. This strategy is similar to Yu et al. (2004). If we regard an observation o_i as a transaction, where the outcome of X is the existence of an itemset X in the transaction, then x is the expected *support* of X (fraction of transactions that contain X), and $\tilde{x}$ is the observed support of X in n transactions. δ relates to the probability that we may overestimate or underestimate the support, which may produce false positives or false negatives. ϵ is how much the observed support deviates from the expected support. For the data stream consideration, the expected support of X is taken as the observed support in the entire data stream, and we denote it by x. We also refer to the expected support as the true support.

We shall consider l-itemsets for $1 \leq l \leq L$. The treatment of l-itemsets is the same for different l's. Hence, in the following, when we consider itemsets, we refer to l-itemsets. Let s_T be the maximum transaction size. Let s_K be the support of the K-th frequent itemset in a set of n observed transactions. Let $C_l^{s_T}$ be the number of combinations of l objects chosen from a set of s_T objects.

In Section 3.2, we show that for mining l-itemsets, the number of itemsets we would store is at most

$$n_{0,l} = \frac{2\left[C_l^{s_T} + 4\ln(2/\delta)\right]}{s_K} \quad (2)$$

Some of the symbols we use are shown as follows.

$s(X)$	the true support of itemset X
$\tilde{s}(X, R)$	the observed support of itemset X in a given set of R transactions
s	the true support of the K-th frequent l-itemset in the entire data stream

In our algorithm, we divide the itemsets into two groups—*potential K-frequent itemsets* in terms of n and *unpromising itemsets* in terms of n. The definition is shown as follows.

Definition 1. Let s_K be the observed support of the K-th frequent itemset in n transactions. Let $\epsilon_{s_K} = \sqrt{\frac{2s_K\ \ln(2/\delta)}{n}}$. An itemset X is *potential K-frequent* in terms of n if $\tilde{s}(X) \geq s_K - 2\epsilon_{s_K}$. An itemset X is *an unpromising itemset* in terms of n if it is not potential K-frequent in terms of n.

In our algorithm, unpromising itemsets will be pruned regularly to keep the memory usage low. Similar definitions are given in Yu et al. (2004). However, in that work, the unpromising itemsets involves the term ϵ_n instead of $2\epsilon_{s_K}$. In our work, the values of ϵ_{s_K} and the term $s_K - 2\epsilon_{s_K}$ are selected to give a guarantee on the accuracy and they are derived mathematically from the proofs of the lemmas and theorems in our analysis in Sections 3.1 and 3.2. However, intuitively, we can explain the factor of 2 as follows: Yu et al. (2004) assumes a support threshold. The estimated frequency of the itemset is the source of error. In the problem of top K itemsets, we have two sources of error. On top of the error in itemset frequency estimation, we need to estimate also a support threshold (i.e. s_K), which would also generate some error. From our mathematical analysis, it turns out that the factor of 2 above provides the guarantees that we want.

Algorithm 1 shows the pseudo code of our method. For efficiency, we process the data stream in batches. For every batch B of R transactions, we keep in the pool P_l the potential K-frequent l-itemsets in terms of R in batch B with respect to s_K found in B. Let n be the number of transactions examined so far in the data stream. Then, we combine the pool P_l with a global pool F_l. We update the support count of each entry in the global pool F_l. If the number of entries in the global pool F_l is greater than the storage capacity given by $n_{0,l}$, then we prune the unpromising itemsets in terms of n in F_l according to the s_K value found in the n transactions read so far. We repeat the process for the remaining batches.

For the first batch, after finding the potential K-frequent itemsets, we evaluate the expected storage $n_{0,1}$ in our algorithm by Eq. (2).

For each batch B, we update the support of each entry in the global pool F_l after we have inserted the entries of P_l into it. Let $F_l{}^{(b)}$ be the pool F_l before the insertion of entries of P_l. Suppose the entry e occurs c times in P_l. If e exists in $F_l{}^{(b)}$, then the support of e in F_l will be updated according to the value of c.[3] Otherwise, the support of e in F_l will be set to c. We also update the support of any e in the global pool F_l which is not in P_l. We obtain the incremental supports of such e by examining the transactions in batch B. We output on demand, returning the top K frequent itemsets in the current global pool F_l.

The implementation details are given in Section 5.

3.1. Analysis

This section will derive some guarantees for the Chernoff-based Algorithm. For the remaining discussion, we find that often we have the criteria $\mathcal{C}$ that $x > \frac{\ln(2/\delta)}{2R}$, where x is roughly the support of the K-th frequent itemset.

This holds in almost all cases because, if $\delta = 10^{-4}$ and $R = 1000$, then $\frac{\ln(2/\delta)}{2R} < 0.01$ and, for any data stream, we can always set the batch size R to be 1000 or above. The value of $\frac{\ln(2/\delta)}{2R}$ decreases with increase in δ or R. This inequality holds in all our experiment.

Our algorithm uses two main estimations: the value of s_{K} in a batch to estimate s, and the estimation of the frequency of itemsets. The inaccuracy in each of these estimates can lead to overall error. The analysis here first sets a bound for the possible error of s_{K} (Lemma 1). Next, in Theorems 1 and 2, we merge this with the guarantee on the estimation of $\tilde{s}(X, R)$ to give the overall guarantee.

In particular, our algorithm finds s_{K} dynamically in each batch and stores the itemsets Z where $\overline{s}(Z, R) \geq s_K - 2\epsilon_{s_K}$. If $s_K \leq s$ occurs in all batches, then the term $s_K - \epsilon_{s_{\mathrm{K}}}$ will

[3] In our implementation, we store the frequency/count of all itemsets, instead of the support (in fraction). So, in this step, we just need to increment the count of e in F_l by c. Similar arguments can be made for other updates on the frequency of the itemsets.

Algorithm 1 Chernoff-Based Algorithm for Mining Top K-frequent itemsets in a Data Stream

```
1: n ← 0, P_l ← ∅, F_l ← ∅
2: for every R transactions do
3:    n ← n + R;
4:    for all l such that 1 ≤ l ≤ L do
5:       find potential K-frequent itemsets in terms of R in the current batch and store
         them in P_l
6:       F_l ← P_l ⋃ F_l
7:       update the support of each entry stored in F_l
8:       if the current batch is the first batch then
9:          calculate n_{0,l} in terms of n
10:      end if
11:      prune unpromising itemsets in terms of n from F_l further if |F_l| > n_{0,l}
12:      P_l ← ∅
13:   end for
14: end for
15: output on demand: the itemsets in F_l with frequency count f greater than or equal to
    f_K, where f_K is the K-th greatest frequency stored in F_l
```

prune less itemsets compared to $s - \epsilon_s$, and the accuracy of the method will not be affected compared to the case when s is known. However, problem would arise if $s_K > s$, and we may prune itemsets that would not be pruned if s is used. The following lemma gives a guarantee that s_K is close to s with high probability.

Lemma 1. *Let* $\epsilon_{s_K} = \sqrt{\frac{2s_K \ln(2/\delta)}{R}}$. *The probability that* $s_K \geq s + \epsilon_{s_K}$ *is at most* δ.

Proof: From the definition of s_K, we are considering a batch B of R transactions, and s_K is the frequency of the K-th frequent itemset in B. We are interested in cases where s_K is greater than s (i.e. $s < s_K$). Let itemset Y be the K-th frequent in B. Hence, $\tilde{s}(Y, R) = s_K$. Let $\epsilon_Y = \sqrt{\frac{2s(Y) \ln(2/\delta)}{R}}$. By the Chernoff bound, with probability $\leq\delta$, $\tilde{s}(Y, R) \geq s(Y) + \epsilon_Y$. Thus, with probability $p_Y \leq \delta$,

$$s_K \geq s(Y) + \epsilon_Y \quad (3)$$

There are three possible cases for itemset Y:

Case 1: Y is the actual K-th frequent itemset in the entire data stream. Hence, $s = s(Y)$. From Eq. (3), with probability $p' \leq \delta$, $s_K > s + \epsilon_Y$

As $s(Y) = s$ and $s < s_K$, $s(Y) < s_K$. Thus, $\epsilon_Y < \epsilon_{s_K}$.

So, with probability p, where $p \leq p' \leq \delta$, $s_K \geq s + \epsilon_{s_K}$.

Case 2: Consider the case when Y is one of the top K frequent itemsets, such that $s(Y) > s$. As Y is the K-th frequent itemset found in the batch, there exists a set A of K itemsets Z such that $\tilde{s}(Z, R) \geq s_K$. Let $\epsilon_Z = \sqrt{\frac{2s(Z) \ln(2/\delta)}{R}}$. By the Chernoff bound, with probability $p_Z \leq \delta$,

$$\tilde{s}(Z, R) \geq s(Z) + \epsilon_Z \quad (4)$$

We need to consider two subcases:

Case 2a: If at least one itemset Z in set A is none of the actual top K itemsets, then $s(Z) < s$.

As $s(Z) < s$ and $\tilde{s}(Z, R) \geq s_K$, from Eq. (4), we conclude that, with probability p', where $0 \leq p' \leq p_Z \leq \delta$, $s_K \geq s + \epsilon_Z$.

As $s(Z) < s$ and $s < s_K$, $s(Z) < s_K$. Thus, $\epsilon_Z \leq \epsilon_{s_K}$.

So, with probability p, where $p \leq p' \leq \delta$, $s_K \geq s + \epsilon_{s_K}$

Case 2b: If all itemsets in set A are the actual top K itemsets in the entire data streams, then there exists an itemset Z in A which is the K-th frequent itemset in the entire data streams. We have $s = s(Z)$.

As $s = s(Z)$ and $\tilde{s}(Z, R) \geq s_K$, from Eq. (4), we conclude that, with probability p', where $0 \leq p' \leq p_Z$, $s_K \geq s + \epsilon_Z$.

As $s = s(Z)$ and $s < s_K$, $s(Z) < s_K$. Thus, $\epsilon_Z < \epsilon_{s_K}$.

So, with probability p, where $p \leq p' \leq \delta$, $s_K \geq s + \epsilon_{s_K}$

Case 3: If Y is not one of the true top K frequent itemsets, then $s(Y) < s$.

From Eq. (3), we can derive that, with probability p', where $p' \leq p_Y$, $s_K \geq s + \epsilon_Y$.

As $s(Y) < s$ and $s < s_K$, $s(Y) < s_K$. Thus, $\epsilon_Y < \epsilon_{s_K}$. So, with probability p, where $p \leq p' \leq \delta$, $s_K \geq s + \epsilon_{s_K}$. □

Lemma 2. *Let* $f(x) = x - \sqrt{\frac{2x \ln(2/\delta)}{R}}$, *where* $0 \leq x \leq 1$. $f(x)$ *is an increasing function for* $x > \frac{\ln(2/\delta)}{2R}$.

The lemma follows since $f'(x) = 1 - \frac{1}{2}x^{-\frac{1}{2}}\sqrt{\frac{2 \ln(2/\delta)}{R}}$.

The following lemma takes care of the scenarios where $s \geq s_K$ while the case of $s < s_K$ will be considered in the proof of Theorem 1.

Lemma 3. *If* X *is among the real top* K *frequent itemsets in the entire data streams,* $s \geq s_K$ *and* $s_K > \frac{\ln(2/\delta)}{2R}$, *for all* s_K, *our algorithm stores itemset* X *with probability at least* $1 - \delta$.

Proof: As X is among the top K frequent itemsets in the entire data stream, X is an itemset with support $s(X) \geq s$ in the entire data stream. Let $\epsilon_X = \sqrt{\frac{2s(X) \ln(2/\delta)}{R}}$. By the Chernoff bound, with probability at most δ, $\tilde{s}(X, R) < s(X) - \epsilon_X$. In other words, with probability at least $1 - \delta$, $\tilde{s}(X, R) \leq s(X) - \epsilon_X$.

As $s_K \leq s$ and $s \leq s(X)$, $s_K \leq s(X)$. By Lemma 2, $s_K - \epsilon_{s_K} \leq s(X) - \epsilon_X$ as $\frac{\ln(2/\delta)}{2R} < s_K \leq s(X)$. Thus, $s_K - 2\epsilon_{s_K} \leq s(X) - \epsilon_X$. Recall that our algorithm stores a set of itemsets Z where $\tilde{s}(Z, R) \geq s_K - 2\epsilon_{s_K}$. Hence, we keep itemsets where $\tilde{s}(Z, R) \geq s(Z) - \epsilon_Z$ if $s(Z) - \epsilon_Z \geq s_K - \epsilon_{s_K}$. This latter is satisfied by the top K frequent itemsets Z. In other words, with probability of at least $(1 - \delta)$, a top K frequent itemset is stored. □

Lemma 4. *Let* X *be one of the top* K *itemsets in the entire data streams. In a batch of* R *transactions, the probability that* $\tilde{s}(X, R) \leq s - \epsilon_s$ *is at most* δ, *where* $\epsilon_s = \sqrt{\frac{2s \ln(2/\delta)}{R}}$.

Proof: If X is K-th frequent, the inequality follows from the Chernoff Bound. For other frequent itemset X' with $s(X') > s$, the expected value of $\tilde{s}(X', R)$ will also be greater than that of $\tilde{s}(X, R)$. Hence the lemma follows. □

Theorem 1. *If* X *is among the top* K *frequent* l*-itemsets in the entire data stream and* $s_K > \frac{\ln(2/\delta)}{2R}$ *for all* s_K, *then, as* $n \to \infty$, *it is returned by Algorithm 1 with a probability of at least* $(1 - \delta)^2$.

Proof: As $n \to \infty$, the corresponding $\epsilon \to 0$. Hence, if X has not been pruned, the recorded support of X approaches $s(X)$. Hence, if a top K frequent itemset is never pruned, it will be returned. There are two cases where itemsets are pruned. The first case is when we prune the unpromising items in terms of R in a transaction batch in line 5 of Algorithm 1 while the second case occurs in line 11 of the algorithm. The two cases are similar except for the number of transactions considered. In the following we arbitrarily take the case of R transactions.

From Lemma 3, we only need to consider the case where $s_K > s$. From Lemma 4, the probability that $\tilde{s}(X, R) \leq s - \epsilon_s$ is at most δ, where $\epsilon_s = \sqrt{\frac{2s\ln(2/\delta)}{R}}$. From Lemma 1, if $\epsilon_{s_K} = \sqrt{\frac{2s_K \ln(2/\delta)}{R}}$, the probability that $s_K \geq s + \epsilon_{s_K}$ is at most δ. Hence, the probability that $\tilde{s}(X, R) > s - \epsilon_s$ is at least $(1 - \delta)$, and the probability that $s_K < s + \epsilon_{sK}$ is at least $(1 - \delta)$. As $s < s_K$, we have $\epsilon_s < \epsilon_{s_K}$. The probability that $\tilde{s}(X, R) > s - \epsilon_{s_K}$ is at least $(1 - \delta)$. When $\tilde{s}(X, R) > s - \epsilon_{s_K}$ and $s_K < s + \epsilon_{s_K}$, we have $\tilde{s}(X, R) > s_K - 2\epsilon_{s_K}$, and X will thus be kept. This probability is at least $(1 - \delta)^2$.

Theorem 2. *If X is not among the top K frequent l-itemsets in the entire data streams and $s_K > \frac{ln(2/\delta)}{2R}$ for all s_K, it is returned by Algorithm 1 with a probability of at most $2\delta - \delta^2$, as $n \to \infty$.*

Proof: Let s be the support of the K-th frequent l-itemset. The value of ϵ_n approaches 0 as $n \to \infty$. When itemsets are returned, we can have their true support. Since we return the K best solutions, the probability of returning X is the same as the probability that a top K frequent itemset is pruned, which is at most $(1 - (1 - \delta)^2) = 2\delta - \delta^2$. □

The following table shows the values of ϵ_s of some values of δ, s_K and n. We observed that, if δ is doubled, ϵ_s is decreased slightly. When n increases, ϵ_s decreases. With the setting $\delta = 0.05$, $s_K = 0.01$ and $n = 100{,}000$, ϵ_s is equal to 0.00086, which is quite small. In synthetic data set, on average, $\epsilon_s = 0.00000$. In real data set, on average, $\epsilon_s = 0.00021$. (The difference between these two data sets is due to a small value of s_K in the synthetic data set and a large value of s_K in the real data set. The larger the s_K is, the greater the error is.)

δ	s_K	n	$(1-\delta)^2$	$2\delta - \delta^2$	ϵ_s
0.05	0.01	50,000	0.91	0.098	0.0012
0.1	0.01	50,000	0.81	0.19	0.0011
0.05	0.01	50,000	0.91	0.098	0.0012
0.05	0.01	100,000	0.91	0.098	0.00086
0.05	0.01	1,000,000	0.91	0.098	0.00027
0.05	0.02	1,000,000	0.91	0.098	0.00038

3.2. Memory consumption

In this section, we analyze the memory utilization of the Chernoff-based Algorithm. Let $\tilde{s}(X, R)$ be the recorded support of itemset X in R transactions in a batch. Let s_T be the maximum transaction size.

Theorem 3. *According to the Chernoff bound, the memory space required by Algorithm 1 for itemsets of size l is* $O(\frac{2[C_l^{s_T}+4\ln(2/\delta)]}{s_K})$.

Proof: In Algorithm 1, there are two pools for itemsets, P_l and F_l. Here we adopt the symbol P for either pool. In the algorithm we deal with either a batch of R transactions or a batch of n transactions in similar ways. Without loss of generality, consider a batch B of R transactions. We store all potential K-frequent items (or itemsets) X with $\tilde{s}(X, R) \geq (s_K - 2\epsilon_{s_K})$. We only need to consider the cases when $s_K > 2\epsilon_{s_K}$.

As each transaction contains at most s_T items, there are at most $C_l^{s_T}$ possible itemsets of size l in each transaction. For R transactions, there are at most $C_l^{s_T}$ R occurrences of l-itemsets. Suppose all l-itemsets that are stored in P has exactly the support of $(s_K - 2\epsilon_{s_K})$, which is the case when the greatest possible number of entries are stored. The number of itemsets stored is therefore bounded by $\frac{C_l^{s_T} R}{(s_K - 2\epsilon_{s_K})R}$. Hence, we have $|P| \leq (\frac{1}{s_K - 2\epsilon_{s_K}}) \times C_l^{s_T}$.

Let R_0 be an upper bound of the storage capacity of our data stream algorithm for itemsets of size l. We can set

$$R_0 = \frac{C_l^{s_T}}{s_K - 2\epsilon_{s_K}} = \frac{C_l^{s_T}}{s_K - 2\sqrt{\frac{2s_K \ln(2/\delta)}{R_0}}}$$

$$R_0 = \frac{C_l^{s_T} + 4\ln(2/\delta) \pm \sqrt{\left[\left(C_l^{s_T} + 4\ln(2/\delta)\right)\right]^2 - \left(C_l^{s_T}\right)^2}}{s_K}$$

(pick + from $\pm$ to choose a larger value.)

$$R_0 \leq \frac{2\left[C_l^{s_T} + 4\ln(2/\delta)\right]}{s_K}$$

Hence, $n_{0,l} = \frac{2[C_l^{s_T}+4\ln(2/\delta)]}{s_K}$ is an upper bound on R_0. Algorithm 1 stores at most $n_{0,1}$ entries, or the memory space is bounded by $O(n_{0,1})$. □

In the above we have a bound for the memory space for arbitrary l-itemset which is independent of n.[4] In the synthetic data with 10,000 items and 1×10^6 transactions of size s_T at most 20, with the setting $\delta = 0.05$ and $l = 8$, from our experiment, the algorithm only stores at most 773 itemsets of size l and 609.4 itmesets on average. (*Note*: s_K in this data set is equal to 0.001158). Compare this with the bound 2,519,640 from the above theorem with the setting $\delta = 0.05$, $s_T = 20$, $l = 8$ and $s_K = 0.1$. The empirical value is typically smaller than the theoretical value many folds. In the real data set BMS-PODS (Kohavi et al., 2000) with 1,657 items and 515,597 transactions, with the same setting, from our experiment, the algorithm only stores at most 90 itemsets of size l and 54.5 itmesets on average. (*Note*: s_K in this data set is found to be 0.012).

[4] Note that in Yu et al. (2004) where Problem A is tackled, there is a bound on the memory space for the single item mining but not for the case of itemsets with multiple items.

3.3. Discussion

Algorithm 1 is built on the Chernoff bound which assumes data independence. This assumption may not hold true in many real data sets. However, as pointed out in Yu et al. (2004), there are some known approaches to handle data dependency that allows us to use our algorithm. One approach is random sampling with a reservoir (Vitter, 1985), which allows us to select n samples randomly without replacement from a pool of transactions. The second one suggested is the probabilistic-inplace algorithm (Demaine et al., 2002) which handles arbitrary data distribution.

In our algorthm, we process data in batches of significant size, e.g. 100,000 transactions, within each batch, ordering of transactions are irrelevant, and hence to some extent our method is randomizing the orders of transaction. To further enhance our algorithm, we suggest a heuristical approach, the main idea of which is to store more potential itemsets. From our experiments, we find that our data stream algorithm stores extremely few entries and the deletion process is seldom required. Hence, we can store more entries to counter the effect of data dependency. We propose two ways to achieve this goal. The first way is to make the batch size smaller. Since the number of times to store the entries will be greater and the probability of storing additional entries is higher. So, the frequency count of each entry stored should be more accurate. The second way is to keep potential $(K \times D)$-frequent itemsets in each batch, instead of potential K-frequent itemsets, where D is an input parameter. If D is set to a large value, that means we expect the data to have a high tendency to change pattern frequently. As we store more top K entries in this way, the chance that we miss the itemsets which are not frequent at the beginning but finally will become top K itemsets is smaller. If we store those itemsets, the frequency counts of those itemsets will be kept and the accuracy of our data stream algorithm can be ensured.

4. Top-K lossy counting algorithm

In this section, we propose an algorithm for mining top K frequent itemsets over the data streams based on the Lossy Counting Algorithm (Manku and Motwani, 2002) which was designed for Problem A. The frequency counts are estimated roughly according to an input error parameter ϵ. Some statistics of the frequencies according to ϵ are kept. In Manku and Motwani (2002), the output are the frequent itemsets according to an input support threshold. We modify the output step for mining top K frequent itemsets. Let us call our proposed method the *Top-K Lossy Counting Algorithm*.

The data are divided into a number of *batches* of R transactions each for efficiency purpose. In each batch, we further divide the transactions into a number of *buckets* of size w, where $w = \lceil \frac{1}{\epsilon} \rceil$. Let β be the number of buckets in a batch. The algorithm stores the itemsets *set* in the pool F_l in the form of entries represented by $(set, f, \triangle)$, where f is the frequency count (not support in fraction) of *set* since this entry was inserted into F_l and $\triangle$ is the maximum possible error in f.

Definition 2. Let the entry of an itemset *set* stored in F_l be $(set, f, \triangle)$. Let n be the number of transactions read so far. Let w be the number of transactions in a bucket. An entry is called *unpromising entries* if $f + \triangle \leq \lceil \frac{n}{w} \rceil$.

The pseudo-code of our proposed method is shown in Algorithm 2. For efficiency, we process the data in batches. For every batch containing R transac-

tions, we find all itemsets with frequency greater than or equal to β in the current batch and store them in the pool P_l. Then, we merge the the entries in pool P_l to the pool F_l. We update the support of each entry stored in F_l. We repeat the above process for the remaining batches.

For each batch, we have to update the support of each entry. The way we update the support is similar to that in the Chernoff-based Algorithm. The implementation details in the Top-K Lossy Counting Algorithm are based on Manku and Motwani (2002). It is noted that, in Manku and Motwani (2002), a new entry will be created if an itemset I occurs frequently in the processing batch. This entry $(set, f, \triangle)$ is created with $set = I, f =$ frequency of I in the processing batch and $\triangle = \lceil \frac{n}{w} \rceil - \beta$.

Algorithm 2 Top-K Lossy Counting Algorithm for Mining Top K-frequent itemsets in a Data Stream

1: $n \leftarrow 0, P_l \leftarrow \emptyset\ F_l \leftarrow \emptyset$
2: **for** every R transactions (a batch) **do**
3: $n \leftarrow n + R$;
4: **for all** l such that $1 \leq l \leq L$ **do**
5: find all itemsets of size l with frequency count greater than or equal to β in the current batch and store them in P_l
6: $F_l \leftarrow P_l \bigcup F_l$
7: update the support of each entry stored in F_l
8: remove all unpromising entries just updated in F_l
9: $P_l \leftarrow \emptyset$
10: **end for**
11: **end for**
12: output on demand: the itemsets in F_l with frequency count f greater than or equal to f_K, where f_K is the K-th greatest frequency stored in F_l

The algorithm requires an input error parameter ϵ, which upper-bounds the error of the estimated frequency of an itemset (Manku and Motwani, 2002). For Problem A, researchers usually set ϵ to be one-tenth of the input support threshold. In our problem of mining top K frequent itemsets, there is no input support threshold. The analogy of the input support threshold in Problem A is the support of the K-th frequent itemset in our problem, s_K. However, before reading the data, we have no idea of s_K and thus we cannot set ϵ according to s_K. Thus, if ϵ happens to be larger than s_K, then the result can be very inaccurate. Therefore, it is suggested to set a very small ϵ value, though a side-effect of this is the increase of memory usage. In Section 6, we show empirically that, with a reasonable input error parameter ϵ, the Top-K Lossy Counting Algorithm not only does not consume much memory but also gives good results.[5]

Our memory utilization is similar to Manku and Motwani (2002), where it is shown that, for mining frequent single items, the memory utilization is $O(\frac{1}{\epsilon}\log(\epsilon n))$, where n denotes the current length of the stream. However, no bound is known on the space requirement for mining of frequent itemsets with more than one item. Our experiments show that the memory consumption is quite small.

[5] Note that in Line 12 of the algorithm, we could use $f + \triangle$ instead of f since $\triangle$ bounds the error in f and the greatest possible frequency is $f + \triangle$. This will ensure no false dismissal but will generate more false positives. We therefore use f instead.

5. Implementation

Chernoff-based Algorithm: In the Chernoff-based Algorithm, we need to store the entries in the global pool F_l and the local pool P_l. The entries in the pools are kept in a hash structure. In the step of mining the potential K-frequent itemsets, we mine the top K' itemsets for every batch. In our implementation, we make use of BOMO algorithm (Cheung and Fu, 2004) which mines top K' itemsets by an FP-tree like structure (Han et al., 2000) (a compressed form of transactions), increasing the support threshold of the FP-tree dynamically.

We process the data in batches as follows. We make use of BOMO algorithm (Cheung and Fu, 2004) to mine the top K itemsets, from which we can obtain the K-th frequent itemset and thus can determine s_K. Next we can use a support threshold of $s_K - 2\,\epsilon_{sK}$ to mine all the itemsets with support above the threshold, and insert them into the pool P_l. Keep in mind that the itemsets stored in P_l are associated with their individual frequencies (count in the batch).

Next the global pool F_l and the local pool P_l are merged. We first build an FP-tree for a batch R by the BOMO algorithm when we look for the potential K-frequent itemset in R. Frequencies of itemsets are kept with the corresponding nodes in the FP-tree.

In order to find the frequency of itemset e in the batch, we do the following. Let q be the frequency of itemset e in the batch. Set $q = 0$ initially. We first find the item I with the smallest individual frequency from the set of items in the itemsets e. Then, we traverse the nodes of item I one by one horizontally. For each node traversed, we parse the link of the node upwards to its parent and continue all the way up to the root, checking whether those nodes above I contains the items in the itemset e other than I. If yes, q is incremented by the frequency of the node. We repeat the above steps with all nodes in the horizontal link and sum up all the counts. The sum is equal to the frequency of itemset e in the batch.

Top-K Lossy Counting Algorithm: We adopted the same implementation as Manku and Motwani (2002) (including the three modules *Buffer*, *Trie* and *SetGen*), except the output step of the algorithm. Let F be the pool storing the entries of the itemsets. In the output step, we just scan the pool once to find the top K itemsets with frequency count (not support in fraction) greater than or equal to f_K, where f_K is the K-th greatest frequency stored in F.

6. Empirical study

We have conducted our experiments on the Pentium IV 2.2 GHz PC with 1 GB memory, on a Linux platform. We compare our algorithms with three approaches. The first one is a naive approach which computes the top K frequent itemsets by the BOMO algorithm (Cheung and Fu, 2004) treating all the data in the data stream as one batch. The second one is the algorithm based on the Zipfian Distribution (Wong and Fu, 2005b), which mines the top K frequent itemsets with a sliding window over data streams. The third one is the algorithm making use of the minimal space for mining top K frequent items (Metwally et al., 2005), which is called Space-Saving algorithm.[6] We denote our algorithms by *Chernoff* and *Lossy*, the (Wong and Fu, 2005b) algorithm by *Zipfian*, the (Metwally et al., 2005) algorithm by *Space* and the naive approach by *BOMO*.

[6] As Metwally et al. (2005) only considers mining frequent items, we adapt the algorithm in Metwally et al. (2005) for mining frequent itemsets. The modification is just a straightforward approach which is similar to Manku and Motwani (2002).

For the Zipfian algorithm (Wong and Fu, 2005b), the sliding window is divided into a number of partitions, called buckets or batches. Each bucket corresponds to a set of transactions and the algorithm maintains the statistics for the transactions in each bucket separately. The window is slided forward one bucket at a time. When the window is advanced, the oldest bucket is discarded and a newly generated bucket is appended to the sliding window. At the same time, the candidate top K interesting itemsets are adjusted. The algorithm has some guarantees that there are no false negatives for any data distribution. Given a Zipfian data distribution with Zipfian parameter θ and an error parameter $\overline{\epsilon} > 0$, it outputs no more than $K[(1+\overline{\epsilon})^{1/\theta} - 1]$ false positives.[7] As the Zipfian algorithm is an approach of finding the top K itemsets with a sliding window over data streams, for comparison, we regard the entire data streams as a sliding window when the Zipfian algorithm is adopted.

Our data stream algorithms (Chernoff and Lossy) and the tailoring of the BOMO algorithm are implemented in C/C++. The code of the BOMO algorithm is provided by the authors in Cheung and Fu (2004). The code of the data stream algorithm based on the Zipfian Distribution is available by same authors in Wong and Fu (2005b). We make use of the BOMO algorithm to obtain the top K' itemsets in a batch. We have tested on both two synthetic data sets and several real data sets.

To measure the quality of the results, we use two metrics—the **recall** and the **precision**. Given a set T of true frequent itemsets and a set O of frequent itemsets obtained in the output by the algorithm, the recall, denoted by **R**, is $\frac{|T \bigcap O|}{|T|}$ and the precision, denoted by **P**, is $\frac{|T \bigcap O|}{|O|}$. If the recall equals 1, the results returned by the algorithms contains all the true results. That means no false negatives are returned. If the precision equals 1, all the results returned by the algorithms are some or all of the true results. That means no false positives are returned. An algorithm is said to be *accurate* if *both* the recall and the precision are near to 1.00.

Synthetic Data Set 1: We experimented on the IBM synthetic data set (Agrawal,). We have generated the data with the following parameters: 10,000 items, 1×10^6 transactions, 10 items per transaction on average, 4 items per frequent itemset on average and support of the top K-th frequent itemsets is 0.1% of the average support. The parameters used are the same as Manku and Motwani (2002) and Yu et al. (2004) except for the average support of frequent itemsets.

Synthetic Data Set 2: This is similar to Synthetic Data Set 1, except that support of the top K-th frequent itemsets is 0.2 of the average support.

Real Data Sets: We have adopted three real data sets. (1) BMS-POS (Kohavi et al., 2000): This data set comes from a large electronics retailer with many different products. There are 515,597 transactions and 1,657 different items. The average transaction size is 6.5. We found that 5% of the top K frequent itemsets changes every batch containing 50 K transactions, where $K = 20$. There is a slight change of frequent patterns over time in this data set. (2) tiny.dat (Minnesota, 1998): This is a small database of the 1990 United States census data with 77 different items and 5,577 tuples. (3) small.dat (Minnesota, 1998): This is a large database of the census data with 77 different items and 57,972 tuples. Real Data Sets (2) and (3) are also the data set used by previous work in Cheung and Fu (2004).

[7] *Note*: The meaning of $\overline{\epsilon}$ in the algorithm is different from the error parameter ϵ in Lossy Counting Algorithm.

We adopt the default setting $\delta = 0.05$ for the Chernoff-based Algorithm[8] and the default setting $\epsilon = 0.001$ for the Lossy Counting Algorithm.[9] For clarity, we do not apply any methods introduced in Section 3.3 for all the experiments except the last empirical study for "Impact of Improvement on Chernoff algorithm" in this section. We also follow (Wong and Fu, 2005b) to adopt the default setting $\overline{\epsilon} = 1.0$ and $\theta = 1$ for the Zipfian Algorithm. The default settings for Synthetic Data Set 1 and Synthetic Data Set 2 are: $K = 20, R = 50K$ and $L = 6$. The default settings for the real data sets are $K = 20, R = 50K$ and $L = 4$.

The memory of the data stream algorithms (i.e. Chernoff, Lossy and Zipfian) involves the number of entries stored and the temporary storage of the most recent batch. As all these algorithms need to store the transactions in the most recent batch, for comparison, we report the memory occupied by the entries stored in the algorithms in the memory graphs. For comparison, the memory usage of Space Algorithm is set to the memory usage of Chernoff Algorithm.

It is easy to see that the memory usage of the naive algorithm (BOMO) is by far larger than that of the other algorithms, because the naive algorithm stores all transactions read so far and the other algorithms store a summary of the data. For instance, if we have the setting of $K = 20, R = 50K, L = 6, \delta = 0.05$ and $\epsilon = 0.001$ in Synthetic Data Sets 1 and 2, the memory usage of the naive algorithm is at least 11 times time greater than that of all data stream algorithms. For the sake of comparison of data stream algorithms, we do not include the naive approach in the graphs for memory usage.

We study the effect of the following factors on the the execution time, the memory consumption and the recall/precision of the algorithms in the Synthetic Data Set 1, the Synthetic Data Set 2 and the Real Data Sets. For interest of space, we only show the results of the real data sets. The results of the synthetic data sets can be found in Wong and Fu (2005a).

Effect of number of frequent itemset to be mined K: Figures 1–3 show the results by varying K. In all data sets, the execution time and the memory usage of all algorithms increases with K because of the higher complexity and higher storage capacity in mining K itemsets with larger K. The accuracy of the algorithms remains nearly the same when K increases.

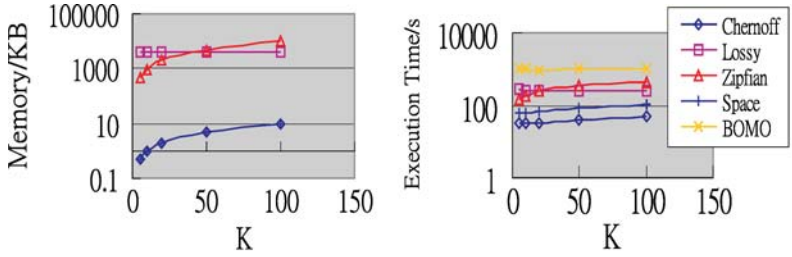
In most cases, the Zipfian algorithm consumes the largest amount of memory among all the data stream algorithms. There are the following two reasons. Firstly, intuitively, it tries to store as many itemsets as possible in order to achieve a guarantee that the output should contain no false negatives. Secondly, as the Zipfian algorithm is designed for the mining over a sliding window (Wong and Fu, 2005b), it needs to "remove" the effect of out-dated data. For that, it stores the entries in the local pool associated with each batch. Usually, the Space algorithm does not have 100% recall because the algorithm removes the real frequent itemsets at the beginning and stores them back afterwards, which introduces errors.

For the Real Data Set 1, as data independence may not hold, the Chernoff algorithm does not give a perfect solution. Nevertheless, the recall and the precision of the results are nearly 1.00. Lossy algorithm maintains high accuracy, probably because ϵ happens to be set above

[8] Yu et al. (2004) adopted the default setting of $\delta - 0.1$, which means the probability that the real frequent itemsets are missed is at most 0.1. In this paper, we have to make a smaller default value $\delta - 0.05$ because Theorem 2 suggests that the probability that the top K frequent itemsets are missed is at most $2 * 0.05 - 0.05^2 = 0.0975 \approx 0.1$.

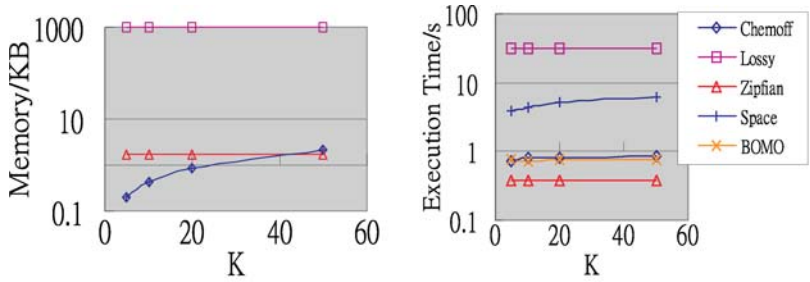
[9] In Manku and Motwani (2002), ϵ is set to be $0.1 \times s$, where s is the user support threshold in Problem A. In this paper, as the problem of mining top K frequent itemsets has no information of the support threshold, we adopt $\epsilon = 0.001$ which was also used in Manku and Motwani (2002) when the support threshold was set to be 0.01.

the support required threshold. However, we observe that the Lossy algorithm consumes more than 200 times memory in terms of entries compared with the Chernoff algorithm when $K = 100$. The high accuracy of the Lossy algorithm has benefited from the abundance of entries stored. For the Real Data Sets 2 and 3, the recall and the precision of all algorithms are equal to 1.00. This is because this data set is quite skewed. The execution time of the Lossy algorithm is the longest and the memory usage of the Lossy algorithm is the greatest.



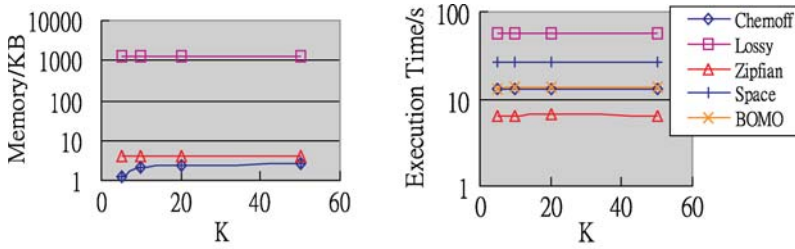
	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	0.97	0.97	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
10	0.95	0.95	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00
20	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
50	0.99	0.99	1.00	1.00	1.00	1.00	0.92	0.98	1.00	1.00
100	0.98	0.98	1.00	1.00	1.00	1.00	0.95	1.00	1.00	1.00

Fig. 1 Real Data Set 1: Varying K



	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Fig. 2 Real Data Set 2: Varying K



	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	0.80	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	0.83	1.00	1.00	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Fig. 3 Real Data Set 3: Varying K

As ϵ is much smaller than the support of the K-th frequent itemsets, Lossy algorithm does a lot of redundant operations and stores a lot of useless entries/itemsets.

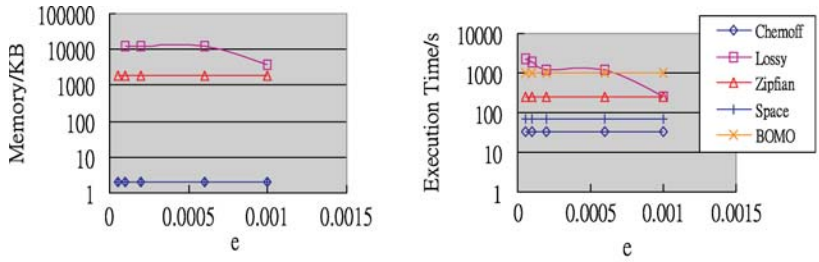
Effect of L, δ and R: We have also carried out an extensive set of experiments to study the effects of the maximal size of itemsets to be mined L, the reliability parameter δ and the batch size R. In all cases, we obtain results as expected. The execution time and memory usage increase with L, the increase is sharper with the Lossy algorithm. The increase in δ affects the accuracy and it is more significant for the Space algorithm, with recall dropping to 0.89 or lower in all cases. With increase in R from 20 to 150 K, the execution time increases slightly but the memory consumption decreases slightly, because there are fewer batches and the chance that we store new entries not in the global pool is lower.

Effect of error parameter ϵ: Figure 4 show the results with the variation of ϵ. Recall that ϵ is an input parameter for the Lossy algorithm only. When ϵ decreases, the execution time and the memory usage of the Lossy algorithm increases steeply. This is because with a smaller ϵ, the Lossy algorithm needs to explore and store more frequent itemsets.

Impact of data arrival order: Similar to Yu et al. (2004), we test several data arrival orders using the Synthetic Data Set 2: OO (original Order), r0 (reverse order), RO (random order), SO (segment-based random order),[10], FF (frequent first), FM (frequent middle) and FL (frequent last). Table 1 shows the results with different data arrival orders. The Chernoff algorithm and the Lossy algorithm is insensitive to the data arrival order. They achieves 100% recall and 100% precision. However, the Zipfian algorithm is sensitive to the data arrival order.

Impact of Improvement on Chernoff algorithm: As the real data set is not independent, the Chernoff Bound may not hold and the Chernoff algorithm may not give good results (shown in the previous sections). Thus, in this subsection, we study how the improved version of the Chernoff algorithm can handle the dependent data.

[10] We randomly re-order data in a unit of segment (1,000 items).



	Chernoff		Lossy		Zipfian		Space		BOMO	
ϵ	R	P	R	P	R	P	R	P	R	P
0.00100	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00060	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00020	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00010	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00005	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00

Fig. 4 Real Data Set 1: Varying ϵ

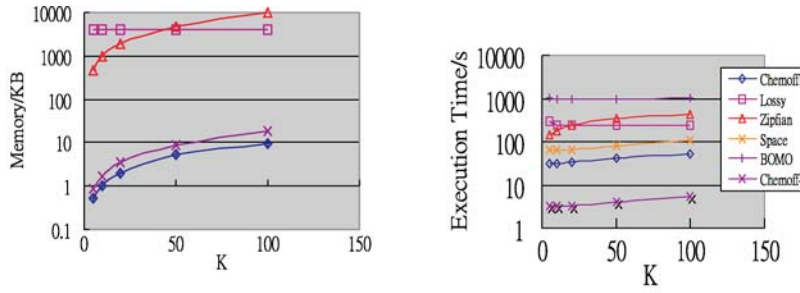
Table 1 Impact of data arrival order: Accuracy

	Chernoff		Lossy		Zipfian		Space		BOMO	
Data	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**
OO	1.00	1.00	1.00	1.00	1.00	0.82	0.89	1.00	1.00	1.00
rO	1.00	1.00	1.00	1.00	1.00	0.82	0.89	1.00	1.00	1.00
RO	1.00	1.00	1.00	1.00	1.00	0.89	0.85	1.00	1.00	1.00
SO	1.00	1.00	1.00	1.00	1.00	0.78	0.85	1.00	1.00	1.00
FF	1.00	1.00	1.00	1.00	1.00	0.90	1.00	1.00	1.00	1.00
FM	1.00	1.00	1.00	1.00	1.00	0.90	0.82	1.00	1.00	1.00
FL	1.00	1.00	1.00	1.00	1.00	0.90	0.75	1.00	1.00	1.00

We adopt the heuristical approach described in Section 3.3. First, we reduce the batch size from 50 to 25 K. Figure 5 shows the results, where Chenoff1 is for 50 K and Chernoff-s is for size 25 K. The accuracy of Chernoff-s (i.e. recall and precision) is just a little better. We have also conducted the experiments with smaller batch sizes (e.g. 12.5 K and 5 K). However, there is no significant increase in accuracy.

The second approach aggressively store more potential top K frequent itemsets. Details can be found in Section 3.3. In this experiment, we set $D = 2$. Figure 6 shows the effects where Chernoff1 is the original Chernoff algorithm with $D = 1$ and Chernoff2 is for $D = 2$. The recall and the precision of Chernoff2 ($D = 2$) are all 1.00. Compared with the algorithm with $D = 1$ (i.e. the original algorithm), on average, the memory usage and the execution time with $D = 2$ is 2.13 times and 2.24 times higher, respectively.

Finally, we have implemented the random sampling with a reservoir (Vitter, 1985) in the Chernoff Algorithm. The recall/precision become 0.99, which is a slight improvement compared with the original Chernoff algorithm with 0.98 recall/precision. We adopted the sample size equal to the half of the batch size. On average, the execution time of this



	Chernoff1		Chernoff-s		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P	R	P
5	0.97	0.97	0.97	0.97	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
10	0.95	0.95	0.95	0.95	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00
20	0.98	0.98	0.99	0.99	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
50	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00
100	0.98	0.98	1.00	1.00	1.00	1.00	1.00	1.00	0.95	1.00	1.00	1.00

Fig. 5 Real data set 1: Impact of cherno. improvement (small batch size): Varying *K*

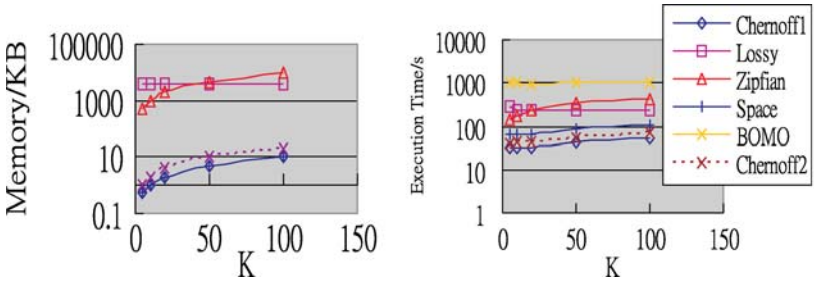


Fig. 6 Real data set 1: Impact of chernoff improvement (More storage): Varying *K*

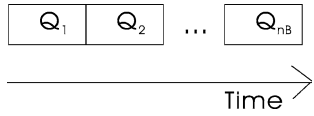


Fig. 7 Storage in a sliding window

improvement is 2.72 times faster than the execution time of the original approach, because the improved approach only processes the data of smaller size. On average, the memory requirement is nearly the same. We have also tested the improvement with different sample sizes (e.g. 25 and 75% of the batch size). The results are also similar.

7. Mining top *K* itemsets in a sliding window

In many applications, dated data is no longer relevant. Instead, only the most recent data is useful. The sliding window model is hence very useful for discounting obsolete data in some applications. Therefore, it is also an important problem to be able to mine a data stream based

on the sliding window model. Here we modify our two data stream algorithms proposed previously for the problem of finding top K itemsets over the entire data streams in order to tackle the same problem for a sliding window model. With a sliding window, we need to find top K itemsets over a period of time. For instance, we want to know the top K itemsets for the past three months, instead of over all records stored in the database.

Formally, we define the problem as follows. Let the size of the sliding window be m. The problem is to mine top K itemsets in the most recent m transactions. There are two strategies to deal with the problem in the sliding window—an eager re-evaluation strategy and a lazy evaluation strategy (Golab and Ozsu, 2003). An algorithm with an eager re-evaluation strategy (Babcock et al., 2003) updates the results whenever data arrives, which is often infeasible when data arrive at a high rate. On the other hand, a lazy evaluation strategy is more practical as an algorithm with this strategy (Lee and Chen, 2001; Xu et al., 2004; Yu et al., 2004) only generate results periodically. Thus, it can handle data with a high arrival rate.

In this paper, we adopt the lazy evaluation strategy. That means our data stream algorithms update the results periodically. We divide the data into a number of batches, each batch containing R transactions. We assume the sliding window size is equal to n_{B} batches, which equals $n_B R$ transactions. Every time we have received R transactions, a batch is formed, and

Algorithm 3 Chernoff-Based Algorithm for the Sliding Window

$n \leftarrow 0, P_l \leftarrow \emptyset, F_l \leftarrow \emptyset$
for every R transactions **do**
  $n \leftarrow n + R$;
  for all l such that $1 \leq l \leq L$ **do**
    switch the batch storages (i.e. $Q_{l,0} \leftarrow Q_{l,1}$, $Q_{l,1} \leftarrow Q_{l,2}$, ..., $Q_{l,n_B-1} \leftarrow Q_{l,n_B}$)
    find potential K-frequent itemsets in terms of R in the current batch and store them in P_l
    $F_l \leftarrow P_l \bigcup F_l$
    update the frequency of each entry stored in F_l
    store all entries updated in the previous step in Q_{l,n_B}
    $P_l \leftarrow \emptyset$
    if the current batch is the first batch **then**
      calculate $n_{0,l}$ in terms of n
    end if
    if there is a batch leaving the sliding window **then**
      decrement the frequency of each entry in F_l which also exists in $Q_{l,0}$ by the count of the entry stored in $Q_{l,0}$
      remove the entries with count 0 in F_l
      $Q_{l,0} \leftarrow \emptyset$
    end if
    prune unpromising itemsets in terms of m from F_l further if $|F_l| > n_{0,l}$, where $m = n_B R$
  end for
end for
output F_l on demand;

we process the data. The results of this batch are stored both in a local pool P for the batch and are updated to the global pool F. The process repeats for the remaining data points.

When there is a batch leaving the sliding window, we update the global pool according to the the local pool for that batch. Note that this algorithm can also work in the eager evaluation strategy if the batch size R is set to 1.

The Chernoff-based Algorithm and the Top-K Lossy Counting Algorithm for the sliding window shown in Algorithms 3 and 4 are similar to those for the continuous data shown in Algorithms 1 and 2, respectively. One of the differences is that we need to store not only the global pool F_l but also the local pool for each batch $Q_{l,i}$, where l is the size of the itemset

Algorithm 4 Top-K Lossy Counting Algorithm for the Sliding Window

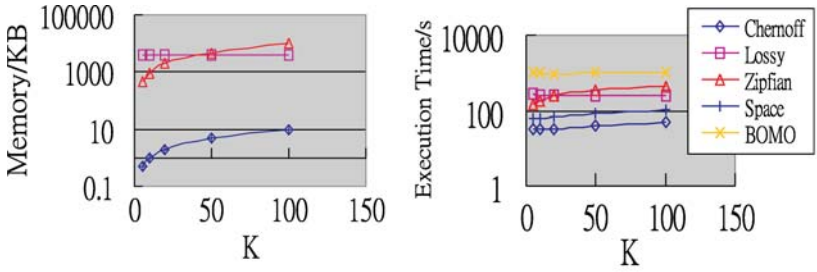
```
1: n ← 0, P_l ← ∅ F_l ← ∅
2: for every R transactions do
3:    n ← n + R;
4:    for all l such that 1 ≤ l ≤ L do
5:       switch the batch storages (i.e. Q_{l,0} ← Q_{l,1}, Q_{l,1} ← Q_{l,2}, ..., Q_{l,n_B−1} ← Q_{l,n_B})
6:       find all itemsets of size l with frequency count greater than or equal to β in the
         current batch and store them in P_l
7:       F_l ← P_l ⋃ F_l
8:       update the frequency of each entry stored in F_l
9:       store all entries updated in the previous step in Q_{l,n_B}
10:      P_l ← ∅
11:      if there is a batch leaving the sliding window then
12:         decrement the frequency of each entry in F_l which also exists in Q_{l,0} by the
            count of the entry stored in Q_{l,0}
13:         remove the entries with count 0 in F_l
14:         Q_{l,0} ← ∅
15:      end if
16:      remove all unpromising entries just updated in F_l
17:   end for
18: end for
19: output on demand: the itemsets in F_l with frequency count f greater than or equal to
    f_K, where f_K is the K-th greatest frequency stored in F_l
```

size and i is the batch number in the sliding window. Another difference is that we need to update the counts of some entries in the global pool F_l when there is a batch leaving the sliding window in Algorithms 3 and 4. If a batch with the local storage $Q_{l,0}$ leaves the sliding window, we decrement the frequency of each entry in F_l which also exists in $Q_{l,\mathrm{o}}$ by the count of the entry stored in $Q_{l,\mathrm{o}}$. Note that we still prune unpromising itemsets in terms of m, where $m = n_B\, R$. However, since the threshold of $n_{l,\mathrm{o}}$ is seldom reached, this effect has not been very significant. We expect data to change over time. For the Chernoff-based Algorithm, the same analysis as in Section 3.1 based on the Chernoff bound would not apply. Instead we have empirical study on the performance.

For the Top-K Lossy Counting Algorithm, as we are now interested in the data in the sliding window, instead of the entire data streams, the method of updating $\triangle$ stored in the entry $(set, f, \triangle)$ should be adjusted accordingly, where $\triangle$ is the maximum possible error in f. When we insert a new entry into the global pool, $\triangle$ should be initialized to be $(n_B - 1) \times \beta$, where $(n_B - 1) \times \beta$ estimates the maximum possible frequency of itemset *set*. Every time there is a batch leaving the sliding window, $\triangle$ of all entries stored in the global storage should be decremented by β. This is because the estimated possible frequency of an itemset should become β smaller as there is a leaving batch containing β buckets. Besides, whenever we decrement the frequency of each entry in F_l which also exists in $Q_{l,0}$, we also set the $\triangle$ of those entries to be 0, because the maximum possible frequency of an itemset should be zero after we remove that leaving batch.

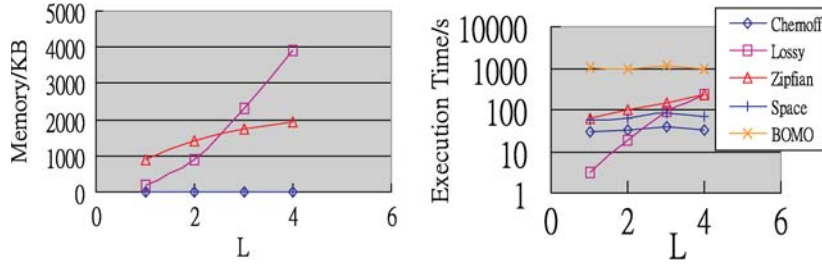
7.1. Empirical study

We have conducted some experiments with Algorithms 3 and 4. The synthetic data sets used here are *Synthetic Data Sets 1* and *2*, which are the same as that in the previous experiments. In the empirical study about mining over an entire data stream, we found that the Chernoff-based algorithm obtain reasonably good results when we set $D = 2$. Results of the memory usage, the recall/precisionand the execution time for Synthetic Data Sets 1 and 2 can be



	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	1.00	1.00	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00	1.00	0.95	1.00	1.00	1.00

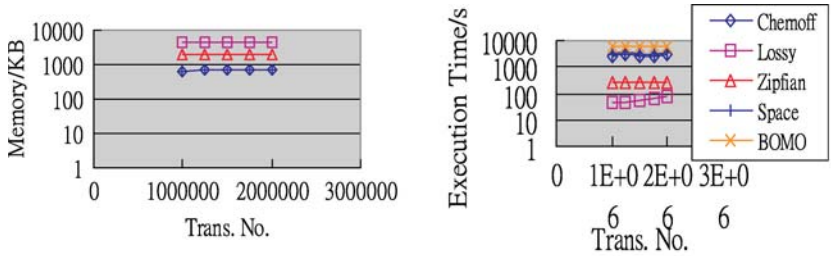
Fig. 8 Real data set 1: Varying *K*



	Chernoff		Lossy		Zipfian		Space		BOMO	
L	R	P	R	P	R	P	R	P	R	P
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00

Fig. 9 Real data set 1: Varying *L*

found in Wong and Fu (2005a). Results for the real data set are shown in Fig. 8 and 9. All the results for the sliding window are similar as those without the sliding window. In conclusion, the Chernoff-based Algorithm is the best among all approaches because it gives 1.0 precions/recall in nearly all of the cases in our experiments.



	Chernoff		Lossy		Zipfian		Space		BOMO	
Trans No.	R	P	R	P	R	P	R	P	R	P
1000K	1.00	1.00	1.00	1.00	1.00	0.95	0.89	1.00	1.00	1.00
1250K	1.00	1.00	1.00	1.00	1.00	0.92	0.91	1.00	1.00	1.00
1500K	0.96	0.96	1.00	1.00	1.00	0.85	0.85	1.00	1.00	1.00
1750K	0.95	0.95	1.00	1.00	1.00	0.87	0.85	1.00	1.00	1.00
2000K	1.00	1.00	1.00	1.00	1.00	0.92	0.89	1.00	1.00	1.00

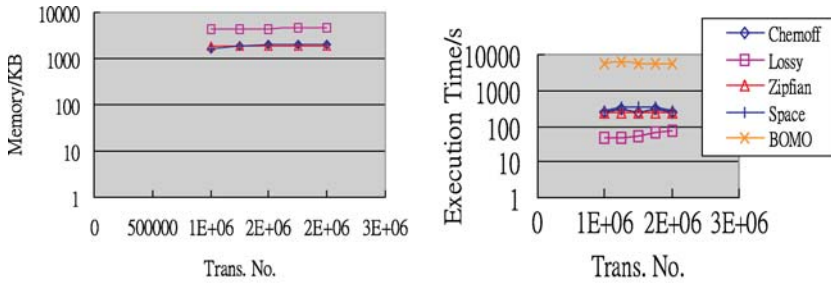
Fig. 10 Synthetic data set 3: Varying Trans. No.

In addition to the previous synthetic data sets, we generated two extra synthetic data sets, namely *Synthetic Data Sets 3* and *4*, in order to evaluate the algorithm for the sliding window model under frequent data pattern changes. Synthetic Data set 3 is generated as follows. There are 2×10^6 transactions. The first 1×10^6 transactions are from the transactions in the previous data set. The remaining 1×10^6 transactions are the same as the transactions in the previous data set but all items are scrambled (different from the previous ones). This synthetic data set is for the cases with a sudden change of frequent patterns.

We generated Synthetic Data Set 4 as follows. Same as Synthetic Data Set 3, Synthetic Data Set 4 has 2×10^6 transactions. The first half of the transactions are from the transactions in the previous data set. The second half are generated like this. There are 1×10^6 transactions in the second half, which are generated based on the first half. We divide this set into a number of batches of size $R = 50K$. For each batch, we find the top $K = 20$ frequent items. We randomly find 5 infrequent items, and call this set of items S. We randomly select one of the top K frequent items, says I, and one item I' other than item I in the set S. Then, we swap these two items, meaning I' becomes frequent and I becomes not. We do the above steps for all batches in the second part. By doing this, we can achieve a gradual change of frequent pattern such that for every batch, we have the probability of $\frac{5}{24} = 0.21$ that a frequent itemset becomes a non-top K frequent itemset. The results are shown in Fig. 10 for Synthetic Data Set 3. The results are shown in Fig. 11 for Synthetic Data Set 4.

Let us examine the results for these two synthetic data sets. The memory usage ratio decreases when the transaction number is increased. This is because in both data sets, there is a change of frequent patterns in the second half of the data. Our algorithm stores more local frequent itemsets for the batches and finally stores them in the global pool, which yields an increase in the memory usage. As the memory usage of the naive approach remains unchanged. So, the memory ratio decreases.

Figures 10 and 11 show that the precision and the recall are nearly equal to 1 in most cases. However, in Synthetic Data Set 3, some values of the precision and the recall of the



	Chernoff		Lossy		Zipfian		Space		BOMO	
Trans No.	R	P	R	P	R	P	R	P	R	P
1000K	1.00	1.00	1.00	1.00	1.00	0.95	0.89	1.00	1.00	1.00
1250K	1.00	0.98	1.00	1.00	1.00	0.90	0.85	1.00	1.00	1.00
1500K	1.00	1.00	1.00	1.00	1.00	0.92	0.87	1.00	1.00	1.00
1750K	1.00	1.00	1.00	1.00	1.00	0.95	0.89	1.00	1.00	1.00
2000K	1.00	1.00	1.00	1.00	1.00	0.90	0.89	1.00	1.00	1.00

Fig. 11 Synthetic Data Set 4: Varying Trans. No.

Chernoff-based algorithm are not equal to 1.0 when the transaction number is equal to 1500 or 1750 K. The reason is that the first half of the data is totally different from the second half of the data in term of the frequent patterns. In other words, there is no gradual change of frequent patterns, which greatly violates the data independence assumption of Chernoff bound. However, if the data set exhibits a gradual change of frequent patterns, there is less probability of false positives/negatives. In fact, no false positives or negatives are found in the results in Synthetic Data Set 4.

In Figs. 10 and 11, the execution time of our proposed algorithm remains nearly constant, because the execution time of the algorithm depends mainly on the time of finding the top K itemsets in the current processing batch, which is not much affected by the number of transactions. In summary, the algorithm works well in spite of the data changes.

8. Related work

Frequent itemset mining on data streams has been studied by many researchers (Gibbons and Matias, 1998; Manku and Motwani, 2002; Teng et al., 2003; Giannella et al., 2003; Yu et al., 2004; Lee and Chen, 2001; Charikar et al., 2002; Metwally et al., 2005). Some work like ours here considered the mining of frequent itemsets from the beginning of data. The earliest such work seems to be Gibbons and Matias (1998) which mines the hot lists, or most frequent single items, and proposes a sampling technique. Manku and Motwani (2002) proposes the sticky sampling and lossy counting methods. The algorithm in Babcock and Olston (2003) reports the K largest values obtained from distributed data streams consisting of values. Metwally et al. (2005) makes use the minimal space for mining frequent items and top K items. These methods do not handle itemsets of multiple items.

A variation of the problem for mining frequent temporal patterns is studied in Teng et al. (2003). Giannella et al. (2003) proposes to mine time-sensitive frequent itemsets with approximate support guarantee, with fine granularity for more recent data and coarse granularity with less recent data. It is shown in Yu et al. (2004) that mining that may generate false negative can have advantages over that of false positive. These previous works handle Problem A but not Problem B.

The problem of mining data stream over a sliding window has been considered in Datar et al. (2002), Babcock et al. (2003), Giannella et al. (2003) and Wong and Fu (2005b). With a sliding window, we can fade out out-dated data which have become irrelevant. The problem of maintaining aggregates and statistics over a data stream with respect to the last N data elements seen so far is considered in Datar et al. (2002). In Babcock et al. (2003), the problem of finding variance and k-medians over a sliding window consisting of a number of buckets was studied. Both (Chang and Lee, 2003) and (Giannella et al., 2003) gave higher weights for the recent data and lower weights for the more dated data. Wong and Fu (2005b) studied the mining of top K frequent itemsets with a sliding window over data streams, which is probably closest to our work here. The algorithm has a different assumption which is based on the Zipfian Distribution.

Mining top K itemset in more conventional databases has been studied in Fu et al. (2000), Cheung and Fu (2002, 2004), Han et al. (2002) and Charikar et al. (2002). Hidber (1999) noted the weakness of the requirement of a fixed support threshold, and allows the user to change the support threshold at any time.

9. Conclusion

In this paper, we study the problem of mining the K most frequent itemsets (or top-K itemsets) from data streams. The first proposed method is based on the Chernoff bound while the other one is developed from the Lossy Counting Algorithm. For efficiency purpose, we process the data streams in batches. We keep the potential top-K candidates (or frequent itemsets) and their counts. At anytime on demand, our algorithms can return the current results. For the Chernoff-based Algorithm, we show that the memory usage of our algorithm is theoretically bounded. We have a guarantee of no false positive/negatives with high probability. We also extend our problem over an entire data stream to one with a sliding window. Both algorithms are modified to adapt to the new problem. Our experiments shows perfect solutions in almost all cases.

Acknowledgments We thank Y.L. Cheung for providing us the coding of BOMO. This research was supported by the RGC Earmarked Research Grant of HKSAR CUHK 4179/01E, and the Innovation and Technology Fund (ITF) in the HKSAR [ITS/069/03].

References

Agrawal, R. IBM Synthetic Data Generator, http://www.almaden.ibm.com/cs/quest/syndata.html.

Babcock, B., Datar, M., Motwani, R., and OCallaghan, L. 2003. Maintaining variance and k-medians over data stream windows. In *SIGMOD*.

Babcock, B., and Olston, C. 2003. Distributed top-K monitoring. In *SIGMOD*.

Lee, C.-H., C.-R.L., and Chen, M.-S. 2001. Sliding-window Filtering: An Efficient algorithm for incremental mining. In *Intl. Conf. on Information and Knowledge Management*.

Chang, J.H. and Lee, W.S. 2003. Finding recent frequent itemsets adaptively over online data streams. In *SIGKDD*.

Charikar, M., Chen, K., and Farach-Colton, M. 2002. Finding frequent items in data streams. In *29th Intl. Colloquium on Automata, Language and Programming*.

Cheung, Y.-L., and Fu, A.W.-C. 2002. An FP-tree approach for mining n-most interesting itemsets. In *SPIE Conference on Data Mining*.

Cheung, Y.-L., and Fu, A.W.-C. 2004. Mining frequent itemsets without support threshold: with and without item constraints. In *IEEE Trans. on Knowledge and Data Engineering*.

Datar, M., Gionis, A., Indyk, P., and Motwani, R. 2002. Maintaining stream statistics over sliding windows. In *SIAM Journal on Computing*.

Demaine, E., Lopez-Ortiz, A., and Munro, J. 2002. Frequency estimation of internet packet streams with limited space. In *Proc. of 10th Annual European Symposium on Algorithms*.

Fu, A.W.-C., Kwong, F.W.-W., and Tang, J. 2000. Mining N-most interesting itemesets. In *ISMIS*.

Giannella, C., Han, J., Pei, J., Yan, X., and Yu, P. 2003. Mining frequent patterns in data streams at multiple time granularities. In *Next Generation Data Mining*.

Gibbons, P.B. and Matias, Y. 1998. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*.

Golab, L. and Ozsu, M.T. 2003. Processing sliding window multi-joins in continuous queries over data streams. In *VLDB*.

Han, J., Pei, J., and Yin, Y. 2000. Mining frequent patterns without candidate generation. In *SIGMOD*.

Han, J., Wang, J., Lu, Y., and Tzvetkov, P. 2002. Mining Top-K frequent closed patterns without minimum support. In *ICDM*.

Hidber, C. 1999. Online association rule mining. In *SIGMOD*.

Kohavi, R., Brodley, C., Frasca, B., Mason, L., and Zheng, Z. 2000. KDD-Cup 2000 Organizers Report: Peeling the Onion". In *SIGKDD Exploration* 2(2).

Manku, G.S., and Motwani, R. 2002. Approximate frequency counts over data streams. In *VLDB*.

Metwally, A., Agrawal, D., and Abbadi: A.E. 2005. Efficient computation of frequent and top- *k* elements in data streams. In *ICDT*.

Minnesota: 98. http://www.ipums.umn.edu/usa/samples.html. In *Minnesota Population Center in Univ. of Minnesota IPUMS-98*.

Teng, W.-G., Chen, M.-S., and Yu, P.S. 2003. A regression-based temporal pattern mining scheme for data streams. In *VLDB*.

Vitter, J.S. 1985. Random sampling with a reservoir. In *ACM Transactions on Mathematical Software (TOMS),* 11(1).

Wong, R.C.-W. and Fu, A.W.-C. 2005a. Mining top K-frequent patterns from data streams: A study. In *Technical report, Computer Science and Engineering Department, Chinese University of Hong Kong*.

Wong, R.C.-W. and Fu, A.W.-C. 2005b. Mining top-K itemsets over a sliding window based on zipfian Distribution. In *SIAM International Conference on Data Mining*.

Xu, J., Lin, X., and Zhou, X. 2004. Space efficient quantile summary for constrained sliding windows on a data stream. In *The 5th Interntaional Conference on Web-Age Information Management*.

Yu, J., Chong, Z., Lu, H., and Zhou, A. 2004. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *VLDB*.