Computing and Informatics, Vol. 27, 2008, 315–339

MINING RECENT FREQUENT ITEMSETS IN SLIDING WINDOWS OVER DATA STREAMS

Congying Han

Department of Mathematics
Shanghai Jiaotong University
Shanghai 200240, China
&
School of Information Science and Engineering
Shandong University of Science and Technology
Qingdao, Shandong Province 266510, China
e-mail: congyh@sdust.edu.cn

Lijun Xu

China Foreign Exchange Trade System
Shanghai 200240, China

Guoping He

School of Information Science and Engineering
Shandong University of Science and Technology
Qingdao, Shandong Province 266510, China
e-mail: hegp@263.net

Manuscript received 21 November 2006; revised 28 September 2007
Communicated by Hélène Kirchner

Abstract. This paper considers the problem of mining recent frequent itemsets over data streams. As the data grows without limit at a rapid rate, it is hard to track the new changes of frequent itemsets over data streams. We propose an efficient one-pass algorithm in sliding windows over data streams with an error bound guarantee. This algorithm does not need to refer to obsolete transactions when they are removed from the sliding window. It exploits a compact data structure to maintain potentially frequent itemsets so that it can output recent frequent itemsets at any time. Flexible queries for continuous transactions in the sliding window can be answered with an error bound guarantee.

Keywords: Data mining, frequent itemset, significant itemset, sliding window, data stream, prefix tree

Mathematics Subject Classification 2000: 68P20, 68U35, 68P30, 68P10

1 INTRODUCTION

Frequent itemset mining [2] has been studied extensively in the last decade. It has become one of the important subjects of data mining. Algorithms for frequent itemset mining form the basis for algorithms for a number of other mining problems, including association rule mining, sequential pattern mining, structured pattern mining, iceberg cube computation, associative classification and so on. Algorithms for frequent itemset mining have typically been developed for datasets stored in persistent storage and involve multiple passes over the dataset. Recently, there has been much interest in data arriving in the form of continuous and infinite data streams [3], which arise in several application domains like high-speed networking, financial services, e-commerce and sensor networks. Data streams possess distinct computational characteristics, such as unknown or unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived items (only one sequential pass over the data is permitted), and a lack of system control over the order in which the data arrive. As data streams are of unbounded length, it is intractable to store the entire data into main memory. However, for many applications, it is important to retain the ability to execute queries that refer to past data. To support such queries is a major challenge of data stream processing. On the other hand, in most applications very old data is considered less useful than more recent data. For the sake of analysis of data streams, we need to identify the recent changes. And the sliding window model [4, 6–10] is one commonly used approach, in which only the last N elements to arrive in the stream are considered useful for answering queries, where N is the size of the window.

In this paper we consider mining recent frequent itemsets in sliding windows over data streams and estimate their true frequencies, while making only one pass over the data. In our design, we actively maintain potentially frequent itemsets in a compact data structure. Compared with existing algorithms, our algorithm has two contributions as follows:

1. It is a real one-pass algorithm. The obsolete transactions are not required when they are removed from the sliding window.

2. Flexible queries based on continuous transactions in the sliding window can be answered with an error bound guarantee.

2 DEFINITIONS

The problem of mining frequent itemsets in sliding windows over data streams is stated as follows:

Definition 1. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct literals, called items. A subset of items is denoted as an itemset. An itemset with k items is called a k-itemset. A transaction $t = (TID, e)$ is a tuple where TID is a transaction-id and e is an itemset. A data stream S is a sequence of transactions, $S = \{t_1, t_2, \ldots\}$. We denote $|S|$ as the number of transactions in the current data of S.

Definition 2. The frequency of an itemset e is the number of transactions in the current data of S that contain e. The support value of an itemset e is the frequency of e divided by the total number of transactions in the current data of S, i.e., $s_e = |\{t|t \in S \bigwedge t \supseteq e\}|/|S|$. An itemset is frequent if it satisfies the support threshold (θ). We denote FI as the set of frequent itemsets, i.e., $FI = \{e|s_e \geq \theta\}$.

Definition 3. A sliding window over a data stream is a bag of last N elements of the stream. There are two variants of sliding windows based on whether N is fixed (fixed-sized sliding windows) or variable (variable-sized sliding windows). Fixed-sized windows are constrained to perform the insertions and deletions in pairs, except in the beginning when exactly N elements are inserted without a deletion. Variable-sized windows have no constraint.

Fixed-sized and variable-sized windows model several variants of sliding windows. For example, tuple-based windows correspond to fixed-sized windows, and time-based windows correspond to variable-sized windows.

3 RELATED WORK

Chang et al. [5] utilized an information decay model to differentiate the information of recent transactions from the information of old transactions. Essentially, this approach reflects the new change of data streams by giving large weights to recent transactions. Manku et al. [11] presented an algorithm, Lossy Counting, to discover approximate frequent itemsets over whole data streams through one pass of the data. This algorithm is derived from an algorithm for finding frequent items. In this algorithm newly generated transactions are stored in one buffer of main memory and are batch-processed later. Based on the work by Karp et al.[15] on computing frequent items, Jin et al. [14] proposed another one-pass algorithm, STREAM, to compute approximate frequent itemsets over entire data streams. Unlike Lossy Counting, STREAM processes incoming transactions one by one. For [5, 11, 14], the support count is computed from the entire data set between the landmark and the current time instead of recent data.

Chang et al. [6] proposed a sliding window method of finding recent frequent itemsets over a data stream based on the estimation mechanism of the Lossy Counting algorithm [11]. They [7] proposed another data stream mining algorithm for recent frequent itemsets by sliding window method. The algorithm estimates the frequency of an itemset that is not currently monitored by the frequencies of its subsets that are currently monitored. Chi et al. [8] monitored potentially frequent closed itemsets as well as the itemsets that form the boundary between the potentially frequent closed itemsets and the rest of the itemsets. The set of frequent closed itemsets is a concise representation of the set of frequent itemsets. Each frequent itemset and its support value can be determined from the set of frequent closed itemsets without accessing the data. Cheng et al. [12] proposed a progressively increasing minimum support function, which increases the error parameter at the expense of only slightly degraded accuracy, but significantly improves the mining efficiency and saves memory usage. [6–8, 12] have one common shortcoming, i.e., they still refer to obsolete transactions when these transactions are removed from the sliding window. Thus the scalability is heavily limited as they have to maintain all the data in the sliding window.

Giannella et al. [10] maintained FP-stream on a tilted-time window, a variant of FP-tree, to mine time-sensitive frequent itemsets at multiple time granularities. This algorithm computes recent change at a fine granularity and long term changes at a coarse granularity. It uses a time-based sliding window and the content of the result is limited by the definition of the window. Lin et al. [13] introduced an efficient algorithm for mining frequent itemsets over data streams under the time-sensitive sliding-window model. They designed the data structures for mining and discounting the support counts of the frequent itemsets when the window slides. The algorithm does not store obsolete transactions. However, the errors in the support counts which are essential to generate interesting association rules can not be precisely estimated.

4 ALGORITHM DESCRIPTION

In this section, we propose one novel stream mining algorithm, FW_SM, which can discover frequent itemsets over a fixed-sized window. In Section 4.1, we discuss the basic estimation mechanism. In Section 4.2, the design of the sliding window is discussed in detail. The sliding window is divided into disjoint blocks and potentially frequent itemsets of each block are maintained. Through these maintained itemsets, we can generate frequent itemsets over the whole sliding window with an error bound guarantee. In Section 4.3, we introduce how to discover potentially frequent itemsets of each block. In Section 4.4, we present an efficient data structure, CP-tree, to store potentially frequent itemsets. In Section 4.5, we outline the general description of our algorithm. In Section 4.6, we discuss how to deal with the case of variable-sized windows.

4.1 Estimation Mechanism

Obviously, a recent infrequent itemset may be frequent in the future. So the frequency of each itemset should be recorded so as to compute frequent itemsets accurately. However, the method is impractical due to the prohibitive number of itemsets occurring in data streams. In order to reduce the number of monitored itemsets, some itemsets whose support values are far less than the support threshold are not necessarily recorded as they cannot be frequent in the near future. In order to tell these itemsets from potentially frequent itemsets, we introduce the definition of significant itemsets.

Definition 4. Given the error parameter $\varepsilon(\varepsilon \leq \theta)$, the set of ε-significant itemsets, SI, is a set of itemsets such that (for an itemset e, let v_e denote its estimated support value):

1. If $s_e \geq \varepsilon$, then $e \in SI$.
2. $\forall e \in SI$, $s_e - \varepsilon \leq v_e \leq s_e$.

Obviously, SI may contain some itemsets whose support values are less than ε. So more than one set may satisfies the above two properties. The set of all of the itemsets occurring in the stream is a special case of a set of 1-significant itemsets if we assume their estimated support values are zero.

SI can be used to solve the frequent itemsets problem. Let $AFI = e \mid e \in SI \wedge v_e \geq \theta - \varepsilon$. AFI has the following property:

Property 1. If $s_e \geq \theta$, then $e \in AFI$.

Proof. $s_e \geq \theta > \varepsilon$, then $e \in SI$. According to Definition 4, $v_e \geq s_e - \varepsilon \geq \theta - \varepsilon$ is satisfied, hence $e \in AFI$. □

Property 2. $\forall e \in AFI$, then $s_e - \varepsilon \leq v_e \leq s_e$.

Proof. As $AFI \subseteq SI$, the property is easily concluded. □

Property 3. $\forall e \in AFI$, then $s_e \geq \theta - \varepsilon$.

Proof. $\forall e \in AFI$, then $e \in SI$ and $v_e \geq \theta - \varepsilon$ are satisfied. According to Definition 4, we can obtain that $s_e \geq \theta - \varepsilon$. □

Obviously, AFI is a superset of frequent itemsets. AFI contains all frequent itemsets and some infrequent itemsets with support values between $\theta - \varepsilon$ and θ.

4.2 Sliding Window

An efficient data stream mining algorithm must be one pass algorithm due to the characteristics of data streams. When some transactions are moved out of the sliding window, the algorithm cannot access them again. It is essential to control the accuracy of the algorithm when the window moves forward. In our design, the sliding window is divided into a collection of disjoint equal-sized blocks. Figure 1 shows such a sliding window.

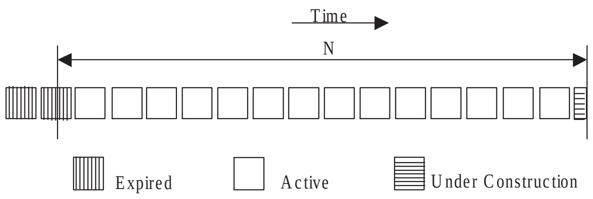


Fig. 1. A Fixed-sized sliding window

The status of any block is evolved according to the following order: under construction (some transactions belonging to the current window and other transactions are not generated), active (all of transactions belonging to the current window), expired (at least one transaction is older than the last N transactions). Obviously, only the blocks at both ends are affected when the window moves forward. In this way, the change is limited to a part of the window instead of the whole window.

The design of the window is as follows: The size of the block is k/ε_B, where k is an integer and $\varepsilon_B = \varepsilon - k/N$. We can adjust k to change the size of the block. We assume there are β/ε_B transactions being processed in main memory. β may change according to the available memory as long as $k \geq \beta$. Each block is uniquely identified by an integer starting from 1. Large blocks with large labels contain recent transactions. $FS(N)$ is defined as a collection of synopses over the window, each of which maintains ε_B-significant itemsets of one block. When the recent frequent itemsets are required, the synopses are merged to compute the answer. When the block is under construction, the significant itemsets are updated as the coming of new transactions. How to discover these significant itemsets will be discussed in this section. If the block becomes active, the significant itemsets with estimated support values are stored and not updated again. When the block finally becomes expired, the corresponding synopsis is useless and discarded.

The correctness of the design of the window is given as follows. Lemma 1 gives the basic approach to generate globally significant itemsets by using locally significant itemsets. Lemma 2 shows how to get significant itemsets with a larger error parameter from significant itemsets with a smaller error parameter. Theorem 3 shows how to merge significant itemsets of each block to compute frequent itemsets over the whole sliding window.

Lemma 1. Let $SI_1, SI_2, \ldots, SI_m$ be sets of significant itemsets over disjoint datasets $D_1, D_2, \ldots D_m$ with error parameters $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m$, respectively. We can compute a set of ε-significant itemsets SI over the dataset D with $SI = SI_1 \cup SI_2 \cup \ldots \cup SI_m$, $D = D_1 \cup D_2 \cup \ldots \cup D_m$ and $\varepsilon = (|D_1|\varepsilon_1 + |D_2|\varepsilon_2 + \ldots + |D_m|\varepsilon_m)/|D|$.

Proof. For an itemset e, let $(s_e)_i (1 \le i \le m)$ be the real support value in D_i, $(t_e)_i$ the estimated support value in D_i, s_e be the real support value in D and t_e be the estimated support value in D.

1. If $s_e \ge \varepsilon$, there must exist at least one dataset $D_i (1 \le i \le m)$, $(s_e)_i \ge \varepsilon_i$. Thus $e \in SI_i$ and $e \in SI$.
2. There are two possible cases:
   (a) $e \in SI_i$
   $$(s_e)_i - \varepsilon_i \le (t_e)_i \le (s_e)_i.$$
   (b) $e \notin SI_i$. We assume $(t_e)_i$ is 0. Clearly, $(s_e)_i < \varepsilon_i$, otherwise, $e \in SI_i$. Thus
   $$(s_e)_i - \varepsilon_i \le 0 = (t_e)_i \le (s_e)_i.$$

   So the following can be got:
   $$(s_e)_i \frac{|D_i|}{|D|} - \varepsilon_i \frac{|D_i|}{|D|} \le (t_e)_i \frac{|D_i|}{|D|} \le (s_e)_i \frac{|D_i|}{|D|}.$$
   Then for D,
   $$\frac{(s_e)_1|D_1| + (s_e)_2|D_2| + \ldots + (s_e)_m|D_m|}{|D|} - \frac{\varepsilon_1|D_1| + \varepsilon_2|D_2| + \ldots + \varepsilon_m|D_m|}{|D|}$$
   $$\le \frac{(t_e)_1|D_1| + (t_e)_2|D_2| + \ldots + (t_e)_m|D_m|}{|D|}$$
   $$\le \frac{(s_e)_1|D_1| + (s_e)_2|D_2| + \ldots + (s_e)_m|D_m|}{|D|}.$$
   Let
   $$t_e = \frac{(t_e)_1|D_1| + (t_e)_2|D_2| + \ldots + (t_e)_m|D_m|}{|D|}.$$
   As
   $$s_e = \frac{(s_e)_1|D_1| + (s_e)_2|D_2| + \ldots + (s_e)_m|D_m|}{|D|},$$
   the following can be concluded:
   $$s_e - \varepsilon \le t_e \le s_e.$$

   So SI is a set of ε-significant itemsets of D. □

It seems that we can easily generate significant itemsets over the whole window based on the above Lemma 1. However, in most cases only a part of the expired block is included into the window. Lemma 1 cannot apply for such cases.

Lemma 2. Let SI_1 be a set of ε_1-significant itemsets. If $\varepsilon_1 < \varepsilon_2$, then SI_1 is also a set of ε_2-significant itemsets.

Proof. The key of controlling the accuracy lies in the algorithm how to deal with the obsolete transactions.

1. If $s_e \geq \varepsilon_2$, then $s_e \geq \varepsilon_2 > \varepsilon_1$, i.e. $e \in SI_1$.
2. $\forall e \in SI_1$, $s_e - \varepsilon_1 \leq t_e \leq s_e$. As $\varepsilon_1 < \varepsilon_2$, $s_e - \varepsilon_2 \leq t_e \leq s_e$.

So SI_1 is a set of ε_2-significant itemsets. □

Theorem 1. $FS(N)$ generates ε-significant itemsets.

Proof. The key of controlling the accuracy lies in the algorithm how to deal with the obsolete transactions. The transactions in the window can be divided into two segments: D_1 and D_2. D_1 consists of the transactions belonging to the expired block. D_2 consists of the transactions belonging to the active blocks or the block under construction. D_2 contains at most N transactions. These transactions are included in one block under construction and several active blocks. We can generate a set of ε_B-significant itemsets of D_2 from FS(N) according to Lemma 1.

If D_1 is empty, $FS(N)$ generates ε_B-significant itemsets according to its definition and Lemma 1. As $\varepsilon_B < \varepsilon$, $FS(N)$ generates ε-significant itemsets according to Lemma 2.

If D_1 is not empty, the transactions of D_1 are included in the latest expired block. We record this block as B_1, the set of ε_B-significant itemsets of B_1 as $SI(B_1)$ and the number of transaction in D_1 as x, where $1 \leq x \leq k/\varepsilon_B - 1$. Note that as some obsolete transactions in B_1 have been moved out of the window, we cannot guarantee that $SI(B_1)$ is still a set of ε_B-significant itemsets of D_1.

There are two cases as follows:

If $x > k$, we assume that we can generate a set of ε_1-significant itemsets of D_1 from $SI(B_1)$. ($SI(D_1)$ denotes the set of ε_1-significant. For an itemset e, let $s_e(D_1), s_e(B_1), t_e(D_1), t_e(B_1)$ denote its real support value in D_1, its real support value in B_1, respectively.)

1. $SI(D_1) = SI(B_1)$.
2. $\forall e \in SI(D_1)$.

$$t_e(D_1) = \begin{cases} \frac{kt_e(B_1)}{\varepsilon_B x} & \text{if } \frac{kt_e(B_1)}{\varepsilon_B x} < 1 \\ 1 & \text{if } \frac{kt_e(B_1)}{\varepsilon_B x} \geq 1 \end{cases}$$

Obviously, we can maximize the value of ε_1 if we assume that no deleted transaction contains any significant itemset. In this case,

$$s_e(D_1) = \begin{cases} \frac{ks_e(B_1)}{\varepsilon_B x} & \text{if } \frac{ks_e(B_1)}{\varepsilon_B x} < 1 \\ 1 & \text{if } \frac{ks_e(B_1)}{\varepsilon_B x} \geq 1. \end{cases}$$

Then we prove that $SI(D_1)$ is a set of k/x-significant itemsets of D_1 when $x > k$.

1. For an itemset e with $s_e(D_1) \geq k/x$, there are two possible cases:
   (a) $\frac{ks_e(B_1)}{\varepsilon_B x} < 1$. Then $s_e(D_1) = \frac{ks_e(B_1)}{\varepsilon_B x}$, i.e. $s_e(B_1) = s_e(D_1)\frac{\varepsilon_B x}{k} \geq \frac{k}{x} \times \frac{\varepsilon_B x}{k} \geq \varepsilon_B$. So $e \in SI(B_1)$ and $e \in SI(D_1)$.
   (b) $\frac{ks_e(B_1)}{\varepsilon_B x} \geq 1$. Then $s_e(B_1) \geq \frac{\varepsilon_B x}{k} \geq \varepsilon_B$. So $e \in SI(B_1)$ and $e \in SI(D_1)$.

2. $\forall e \in SI(D_1)$, $s_e(B_1) - \varepsilon_B \leq t_e(B_1) \leq s_e(B_1)$ as $SI(D_1) = SI(B_1)$. So

$$\frac{ks_e(B_1)}{\varepsilon_B x} - \frac{k}{x} \leq \frac{kt_e(B_1)}{\varepsilon_B x} \leq \frac{ks_e(B_1)}{\varepsilon_B x}.$$

There are three possible cases:

1. $\frac{ks_e(B_1)}{\varepsilon_B x} < 1$. Then $s_e(D_1) = \frac{ks_e(B_1)}{\varepsilon_B x}$ and $t_e(D_1) = \frac{kt_e(B_1)}{\varepsilon_B x}$. So $s_e(D_1) - \frac{k}{x} \leq t_e(D_1) \leq s_e(D_1)$.
2. $\frac{ks_e(B_1)}{\varepsilon_B x} \geq 1$ and $\frac{kt_e(B_1)}{\varepsilon_B x} < 1$. Then $s_e(D_1) = 1$ and $t_e(D_1) = \frac{kt_e(B_1)}{\varepsilon_B x}$. So $s_e(D_1) - \frac{k}{x} = 1 - \frac{k}{x} \leq \frac{ks_e(B_1)}{\varepsilon_B x} - \frac{k}{x} \leq t_e(D_1) \leq 1 = s_e(D_1)$.
3. $\frac{kt_e(B_1)}{\varepsilon_B x} \geq 1$. Then $s_e(D_1) = 1$ and $t_e(D_1) = 1.s_e(D_1) - \frac{k}{x} = 1 - \frac{k}{x} \leq 1 = t_e(D_1) \leq 1 = s_e(D_1)$.

So $SI(D_1)$ is a set of k/x-significant itemsets of D_1 when $x > k$.

If $x \leq k$, we assume that the significant itemsets of D_1 is a set of all possible itemsets whose estimated support values are zero. It is easy to prove that this set is a set of 1-significant itemsets.

According to Lemma 1, the error parameter for the window ε_w is given by:

$$\varepsilon_w \leq \begin{cases} \frac{\frac{k}{x} \times x + (N-x) \times \varepsilon_B}{N} & if\ x > k \\ \frac{k \times 1 + (N-x) \times \varepsilon_B}{N} & if\ x \leq k \end{cases} < \frac{k + N \times \varepsilon_B}{N} = \frac{k + N \times (\varepsilon - k/N)}{N} = \varepsilon.$$

According to Lemma 2, $FS(N)$ generates ε-significant itemsets. □

Note we do not perform any operation for D_1 when merging the result when $x \leq k$. This is because their support values are zero and cannot influence the final result. In this case, only the synopses of the active blocks or the block under construction are used to generate recent frequent itemsets. Otherwise, the synopsis of the expired block B_1 is also adopted besides the above mentioned synopses.

4.3 Local mining over each block

From Theorem 3, if we can maintain ε_B-significant itemsets of each block, we can generate ε-significant itemsets and compute frequent itemsets over the whole sliding window. Any stream mining algorithm that generates significant itemsets can apply to our algorithm. We use the estimation mechanism of Lossy Counting algorithm [11] to process the data of each block. Lossy Counting takes two user-defined input parameters, namely: the support threshold θ and the error parameter ε. It computes ε-significant itemsets through one pass of the data. The significant itemsets are maintained in a data structure TRI consisting of a set of triples $\langle e, fre, err\rangle$, where e is an itemset, fre is the frequency of e and err is the maximum possible error of fre. The incoming transactions are first stored together and divided into equal-sized sections. Each section includes $1/\varepsilon$ transaction and is uniquely numbered by an integer, starting from 1. The current section identifier is labelled by α, i.e., $\alpha = |S|\varepsilon$. There are β sections being processed in main memory, where $\beta \geq 1$. Notice β may change according to the available memory. TRI is updated as follows:

1. For each triple (e, fre, err) in TRI, fre is first updated according to the incoming transactions. Then the triples that satisfy $fre + err < \alpha$ are removed.
2. Let f be the frequency of an itemset e in the incoming transactions. If $f \geq \beta$ and e does not occur in TRI, a new triple $(e, f, \alpha - \beta)$ is inserted into TRI.

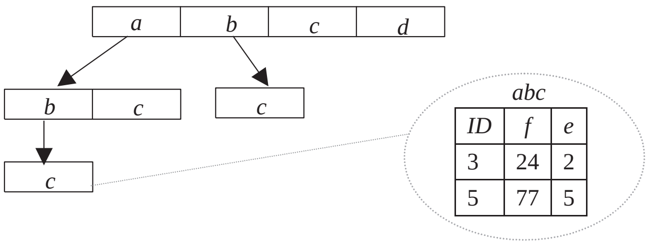
When the recent frequent itemsets are required, the algorithm outputs the itemsets in the TRI with $fre \geq (\theta - \varepsilon)|S|$. The result contains all frequent itemsets and some infrequent itemsets with support values between $\theta - \varepsilon$ and θ.

Lossy Counting regards a local significant itemset in the incoming transactions as a global significant itemset in the current data of the window, which may introduce some insignificant itemsets. There are two ways to relieve this problem:

1. To increase the number of transactions being processed each time. This increases the possibility that a local significant itemset becomes a global significant itemset; but this requires trade-offs among space requirement, accuracy and sensitivity of its mining result.
2. To record the exact frequency of each item. A local significant itemset including a global insignificant item will not be inserted into TRI.

4.4 CP-Tree for Synopses

A compact data structure is required to store the set of significant itemsets in main memory. Most frequent itemsets mining algorithms often use a prefix tree for this purpose. In the prefix tree each node records an item. It is easy to identify an itemset via the path from the root node in the prefix tree. Although the prefix tree has distinct advantages, it is a bottleneck as far as space requirement is concerned because it always uses a great number of pointers. Thus we design a novel implementation, CP-tree (Concise Prefix tree). All sibling nodes are placed in an array in a CP-tree. We use one CP-tree to maintain all synopses in order to save space when some synopses contain identical itemsets or similar itemsets. Besides the item, each node records a list of pairs $\langle bid, fre \rangle$, where bid is the identifier of a block, fre is the estimated frequency of the corresponding itemset in the block. Figure 2 gives a sample of the CP-tree.



ID	f	e
3	24	2
5	77	5

Fig. 2. CP-tree

4.5 Incremental Update

In this section, we present the algorithm in more details. The incoming transaction can not be processed immediately until its number is accumulated to β/ε_B. The size of each block is $k/\varepsilon_B(\beta \leq k)$. Let γ be the label of the block under construction and α be the current section label for the block γ, which means the algorithm has processed α/ε_B transactions in the block γ.

The pseudo-code of the algorithm is given in Algorithm 1. Note that step 3 is only performed when the recent frequent itemsets are required.

Algorithm 1. FW_SW algorithm Input:

1. the support threshold θ and the error parameter ε;
2. the size of the sliding window N and the size of the block k/ε_B;
3. the incoming transactions;
4. the CP-tree.

Output:
the updated CP-tree.

```
1. Mine the incoming transactions by means of modifying Apriori algorithm [2].
   For each mined itemset $e$ with frequency $fre_e$,
   (a) If $e \in$ the CP-tree,
```

- i If there exists an entry $\langle \gamma, fre, err \rangle$ in the list of e, increase the frequency by fre_e. If $fre+err < \alpha$, delete this entry and stop mining the supersets of e.
- ii If γ is not in the list
  - A If $fre_e \geq \beta$, add the entry $\langle \gamma, fre_e, \alpha - \beta \rangle$ into the list.
  - B If $fre_e < \beta$, stop mining the supersets of e.

(b) If $e \notin$ the CP-tree,

- i If $fre_e \geq \beta$, insert e into the CP-tree and add the entry $\langle \gamma, fre_e, \alpha - \beta \rangle$ into the list of e.
- ii If $fre_e < \beta$, stop mining the supersets of e.

2. Traverse the CP-tree (depth-first search). For each itemset e encountered,

   (a) If e is not updated in step (1) and an entry $\langle \gamma, fre, err \rangle$ satisfying $fre+err < \alpha$ exists in its list, then remove the entry.
   (b) Remove the entry of the expired block if the number of its transactions in the current window is not more than k.
   (c) If the list is empty, remove e and its subtree in the CP-tree.

3. If the recent frequent itemsets in the current window is required, merge the related synopses and output itemsets whose support values are not less than $\theta - \varepsilon$.

4.6 Discussion

We use fixed-sized sliding windows so far and shall discuss how to deal with the case of variable-sized windows in brief. Without loss of generality, let the minimal size of the window be N_1. We divide the window into a collection of disjoint blocks. Let VS(N) be a collection of synopses over the window, each of which maintains ε_B-significant itemsets of one block, where $\varepsilon_B = \varepsilon - k/N_1$. Unlike fixed-sized windows, the sizes of blocks are not equal. The sizes of blocks are set according to the following rules:

1. The size of the block with the largest label is k/ε_B, where k is an integer. The number of blocks of size k/ε_B cannot be more than$\frac{2N_1}{k/\varepsilon_B} + 2$. If there are more than $\frac{2N_1}{k/\varepsilon_B} + 2$ blocks of size k/ε_B, merge the synopses of the oldest two into one synopsis of one block of size $\frac{2k}{\varepsilon_B}$ according to Lemma 1.
2. The number of blocks of size $\frac{2^l k}{\varepsilon_B}(l \geq 1)$ cannot be more than $\frac{N_1}{k/\varepsilon_B} + 2$. If there are more than $\frac{N_1}{k/\varepsilon_B} + 2$ blocks of size $\frac{2^l k}{\varepsilon_B}$, merge the synopses of the oldest two into one synopsis of one block of size $\frac{2^{l+1} k}{\varepsilon_B}$.

According to the above design, we can conclude:

$$N \geq \frac{2^{l-1}k}{\varepsilon_B} \times \frac{N_1}{k/\varepsilon_B} + \frac{2^{l-2}k}{\varepsilon_B} \times \frac{N_1}{k/\varepsilon_B} + \cdots + \frac{k}{\varepsilon_B} \times \frac{2N_1}{k/\varepsilon_B}$$

$$= 2^{l-1}N_1 + 2^{l-2}N_1 + \cdots + 2N_1 = 2^l N_1.$$

Theorem 2. VS(N) generates ε-significant itemsets.

Proof. The transactions in the window can be divided into two divisions: D_1 and D_2. D_1 consists of the transactions belonging to one expired block. D_2 consists of the transactions belonging to the active blocks or the block under construction. We can generate a set of ε_B-significant itemsets of D_2 from VS(N) according to Lemma 1.

If D_1 is empty, the conclusion is obvious. Otherwise, we record the corresponding expired block of D_1 as B_1, the size of the block is $2^l k/\varepsilon_B (l \geq 0)$ and the number of transaction in D_1 as x, where $1 \leq x \leq 2^l k/\varepsilon_B - 1$.

Using similar method in Theorem 1, we can generate a set of $(2^l k/x)$-significant itemsets of D_1 from the synopsis of the block B_1 when $x > 2^l k$.

According to Lemma 1, the error parameter for the window ε_w is given by:

$$\varepsilon_W \leq \begin{cases} \frac{2^l k \times x + (N-x) \times \varepsilon_B}{N} & \text{if } x > 2^l k \\ \frac{2^l k \times 1 + (N-x) \times \varepsilon_B}{N} & \text{if } x \leq 2^l k \end{cases} < \frac{2^l k + N \times \varepsilon_B}{N} = \frac{2^l k + N \times (\varepsilon - k/N_1)}{N}$$

$$= \varepsilon + k\frac{2^l - N/N_1}{N} \leq \varepsilon.$$

So VS(N) generates ε-significant itemsets. □

5 EXPERIMENT RESULTS

We performed extensive experiments to evaluate the performance of our algorithm and we present results in this section. We first compared our algorithm (FW_SM) with the Lossy Counting algorithm [11] and estWin algorithm [7], respectively. Then we tested the adaptability of FW_SM by changing the data distribution of the dataset.

The experiments were performed on a Pentium 1.2 G processor with 1 G memory, running Windows 2000 (SP4). Our algorithm is implemented in C++ and compiled by using Microsoft Visual C++ 6.0. In the implementation of three algorithms, we used the same data structures and subroutines in order to minimize the performance differences caused by minor differences.

Two data sets, T20 and Kosarak, are used in the experiments. The T20 data set is generated by the IBM data generator [1]. For T20, the average size of transactions, the average size of the maximal potential frequent itemsets and the number of items are 20, 4, and 1 000, respectively. The Kosarak data set is a real-world data set (`http://fimi.cs.helsinki.fi/data/`), which is derived from the click-stream data of a Hungarian on-line news portal. The data sets were broken into batches of 10 K size transactions and provided to our program through standard input.

For FW_SM, the whole procedure can be divided into two different phases: (1) Window initialization phase, which is activated when the number of transactions generated so far is not more than the size of the sliding window. (2) Window sliding phase, which is activated when the window is full of generated transactions.

1. Comparison between Lossy Counting and FW_SM based on different block size

To observe the influence of different block sizes on algorithm, we take 2×10^4, 5×10^4 and 10×10^4 as an example. Meanwhile, the size of the window is fixed 5×10^5. In this case, this window contains 25, 10 and 5 blocks, respectively. As mentioned above, three versions of our algorithm(FW_SM) are used. Since $L = \frac{k}{\varepsilon_B}$ (we set L to be the size of a block), $\varepsilon_B = \varepsilon - \frac{k}{N}$ and $\varepsilon_B = \frac{N}{N+L}\varepsilon$, then the values of ε_B can be shown in Table 1. (For T20, $\theta = \varepsilon = 0.0025$. For Kosarak, $\theta = \varepsilon = 0.002$.)

Block size	20 000	50 000	100 000
T20	0.0024	0.00227	0.00208
Kosarak	0.00192	0.00182	0.00167

Table 1. ε_B for different block size

The experimental results of the execution time and the space requirement are plotted in Figure 3 and 4, respectively. We collected the total number of seconds and the size of storage space required in KB per 10×5 transactions. These results basically keep stable as the window moves forward. The above experimental results provide evidence that two algorithm can handle long data streams both. As FW_SM may keep multiple triples for one significant itemset, it needs more memory and time than Lossy Counting does.

In order to evaluate the effects of different block size on the result accuracy, two measures are introduced.

1. Precision. It is defined as the ratio of the number of the real frequent itemsets computed by Aprior algorithm to the number of the frequent itemsets computed by our algorithm.
2. ASE(Average Support Error). It is defined as follows:

$$\sum_{e \in FI} \frac{s_e - t_e}{|S|}.$$

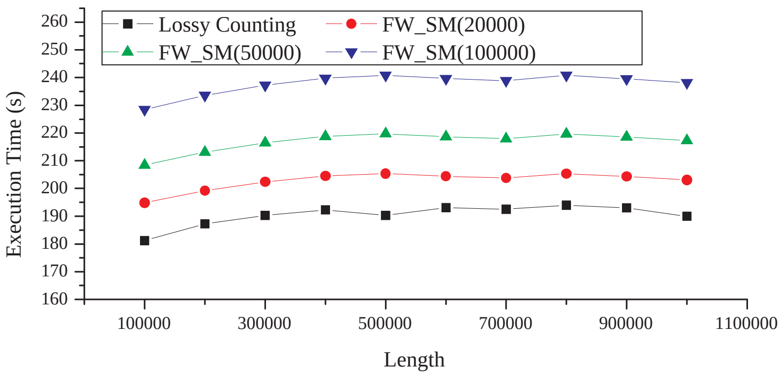
The two above statistics are collected at the end of per 80K transactions and shown in Figures 4 and 5. It is observed that the Lossy Counting algorithm can only guarantee the result accuracy in the initialization phase, but not in the window sliding phase. The accuracy reduces rapidly as the window moves in the window sliding phase. The reason for this is that Lossy Counting computes results based on the whole data instead of the recent data. So the Lossy Counting algorithm can not be used to mine the recent frequent itemsets.

In addition, on the one hand, the result accuracy of FW_SM degrades with the increase of the number of blocks in the window initialization phase. As each block has its own local significant itemsets and the result is computed based on all local significant itemsets, more infrequent itemsets with support values between θ and $\theta - \varepsilon$ may be inserted into the required result as the increase of the number of blocks. On the other hand, FW_SM with large number of blocks basically has better result accuracy than the algorithm with small number in the window sliding phase. If the window just includes some transactions in the oldest block, in FW_SM, we either use the significant itemsets of this block to simulate the significant itemsets of these transactions or neglect these transactions (see the proof of Theorem 3). So the algorithm has bigger error with larger size of blocks.

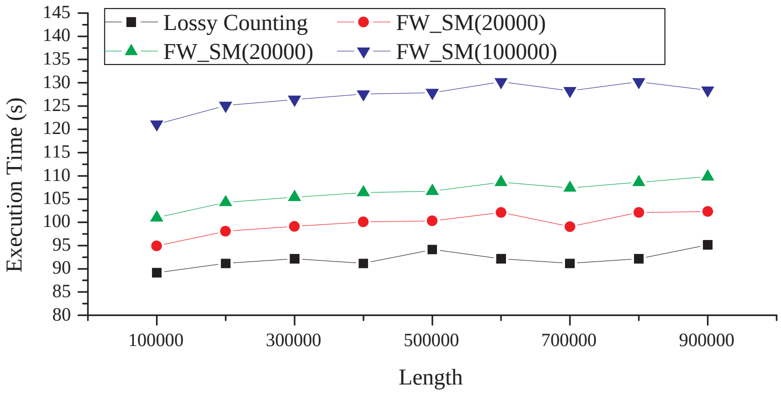
2. Comparison between FW_SM and estWin algorithm
We now compare FW_SM with estWin algorithm as follows. We use the same data structure in these two algorithms. In FW_SW, block size is set to be 5×10^4 and window size is fixed 5×10^5. Meanwhile, the number of transactions in buffer is not greater than 10^4. The results on data set T20 are shown in Figures 6 through 9, where $\theta = \varepsilon = 0.0025$.

From the experimental results, FW_SM execution time is far less than estWin, especially in window sliding phase. Although estWin's space requirement and accuracy are superior to those of FW_SM, its precision is inferior to that of FW_SM. Compared with FW_SM, estWin is not purely memory resident program due to its extra I/O operation. If estWin is used to process high speed data streams, not all data can be dealt with. In other words, estWin algorithm's improvement in space and accuracy is at the cost of execution time.

3. Adapting to change
In reality, seasonal variations may cause the underlying data distribution to change in time. A simple-minded way to simulate some of this shifting effect is to periodically, randomly permute some item names of T20. To do this, we use an item mapping table. The table initially maps all item names to themselves. However, for every 10^6 transactions, 30 % entries of the table are shifted through random permutations. In the experiments, the block size is set to be 5×10^4 and the window size is fixed to 5×10^5. Meanwhile, the number of transactions in buffer is not greater than 10^4. The results are shown in Figures 10 through 14, where $\theta = \varepsilon = 0.0025$. These results fluctuate in a relatively large every 10^6 transactions. Stability is regained soon. In general, the results of the algorithm tend to stabilize despite the random permutations.

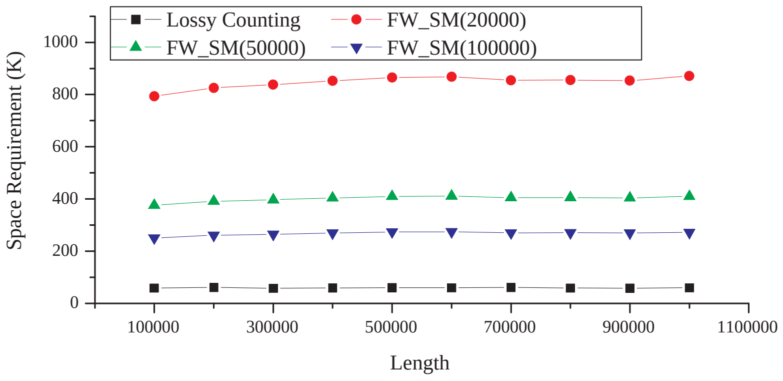


T20

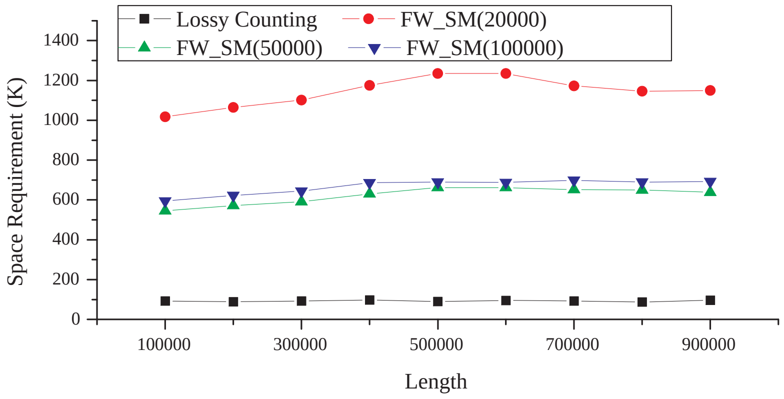


Kosarak

Fig. 3. Execution time

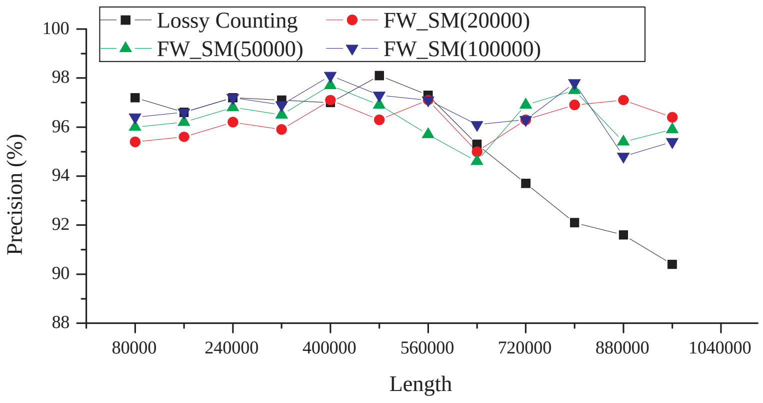


T20

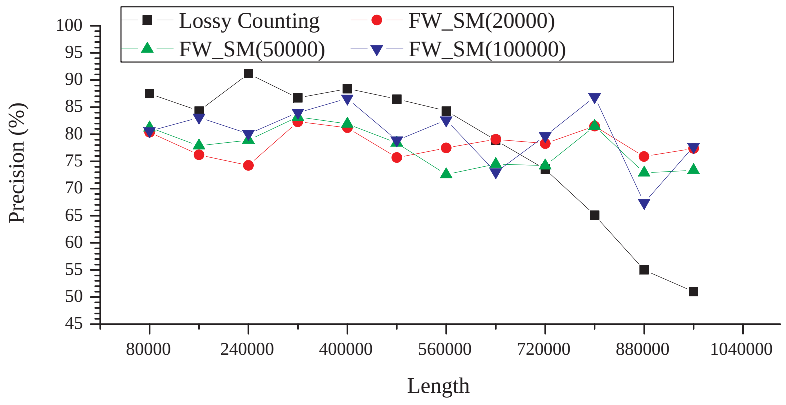


Kosarak

Fig. 4. Space requirement

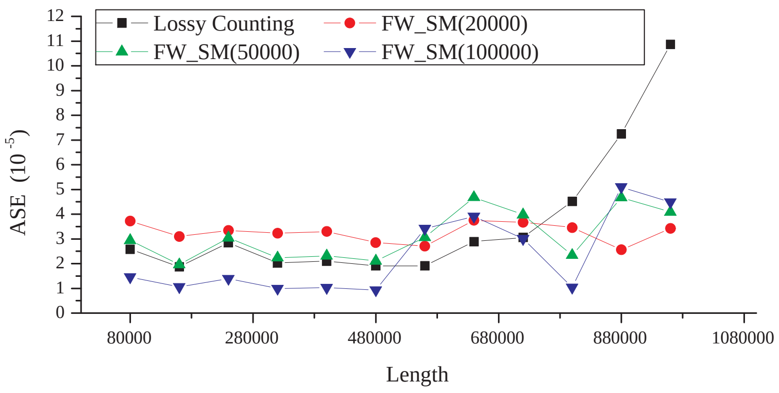


T20

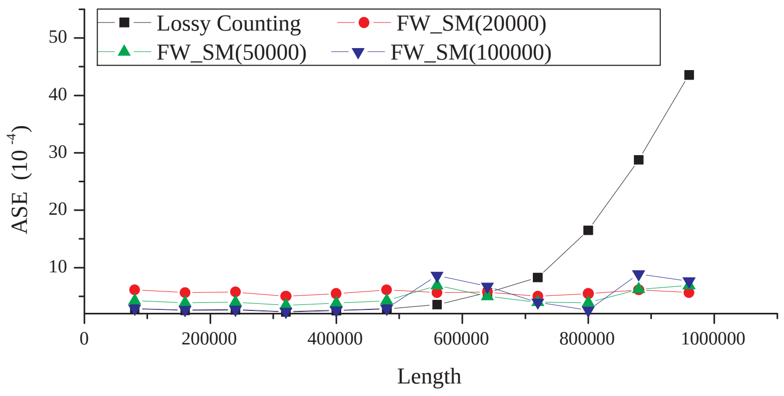


Kosarak

Fig. 5. Precision



T20



Kosarak

Fig. 6. Average support error

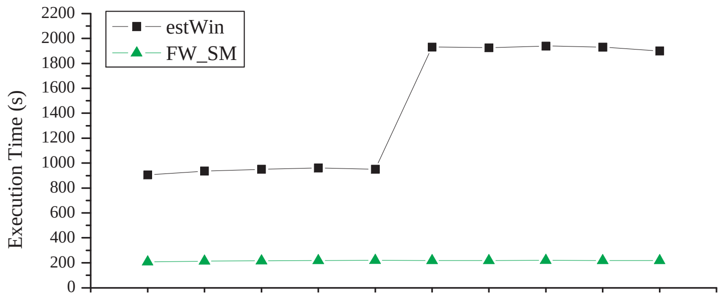


Fig. 7. Execution time

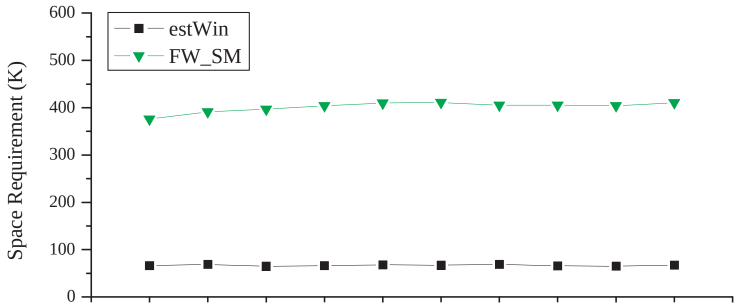


Fig. 8. Space requirement

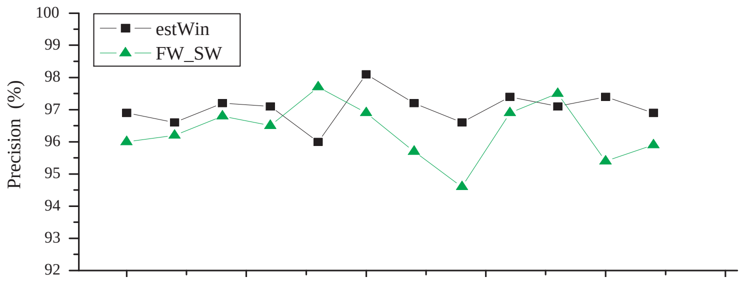


Fig. 9. Precision

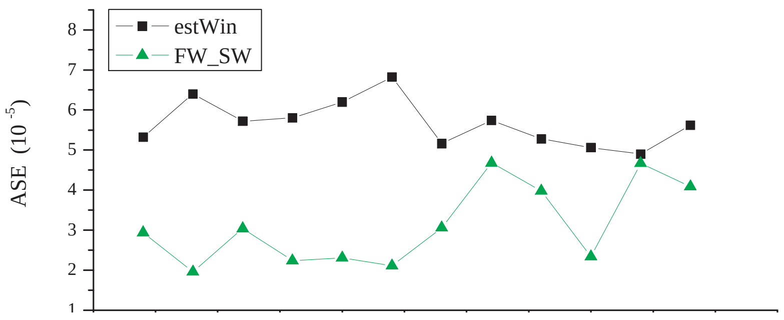


Fig. 10. Average support error

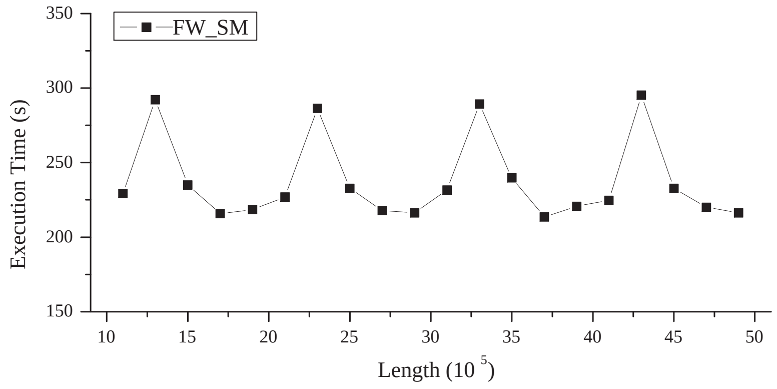


Fig. 11. Execution Time

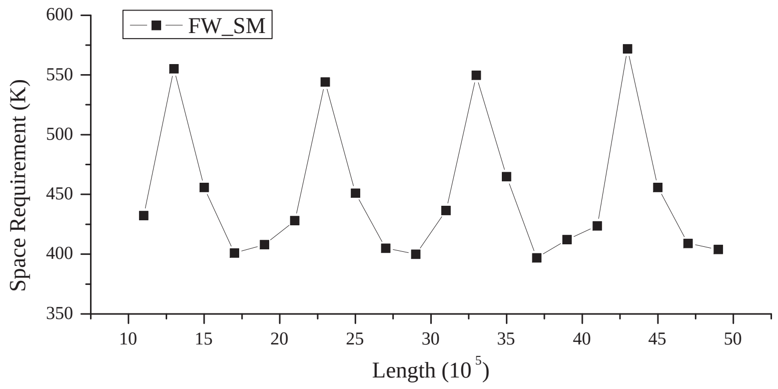


Fig. 12. Space Requirement

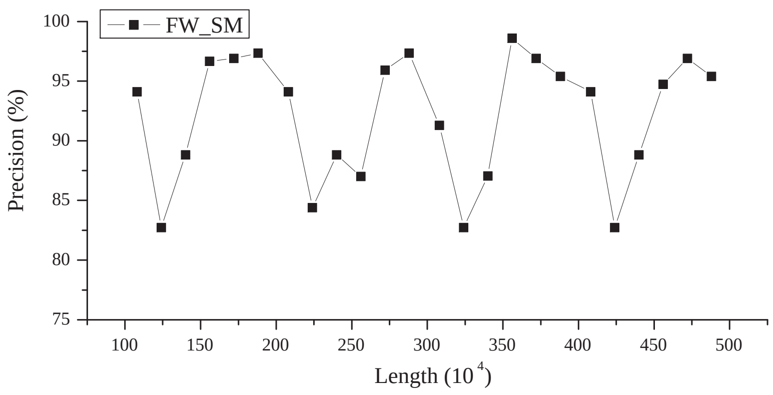


Fig. 13. Precision

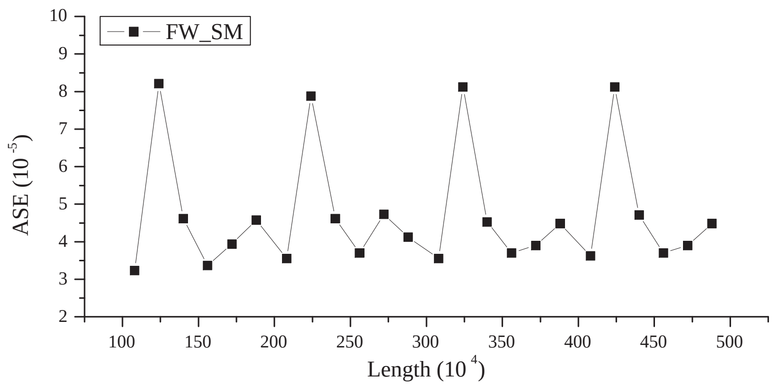


Fig. 14. Average support error

6 CONCLUSION

We mainly discuss how to discover recent frequent itemsets in sliding windows over data streams. An efficient algorithm is presented in detail. Significant itemsets are maintained in a CP-tree, which is incrementally updated with incoming transactions. Compared with previous algorithms, this algorithm doesn't keep the data in the window, which considerably increase the scalability of the algorithm. Moreover, it can figure out answers with an error bound guarantee for continuous transactions in the window. The extensive experiment results demonstrate the effectiveness and efficiency of our approach.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (10571109) and the Educational Foundation of Shandong Province (J06P14, 2006GG3210009).

REFERENCES

[1] Arasu—Manku, G. S.: Approximate Quantiles and Frequency Counts over Sliding Windows. Proceedings of PODS 2004.

[2] Agrawal, R.—Srikant, R.: Fast Algorithms for Mining Association Rules. Proceedings of VLDB, 1994.

[3] Babcock—Babu, S.—Datar, M.—Motwani, R.—Widom, J.: Models and Issues in Data Stream Systems. Proceedings of ACM Symp. on Principles of Database Systems, 2002.

[4] Babcock—Datar, M.—Motwani, R.: Sampling from a Moving Window over Streaming Data. Proceedings of 13th Annual ACM-SIAM Symp. on Discrete Algorithms, 2002.

[5] Chang, J. H.—Lee, W. S.: Finding Recent Frequent Itemsets Adaptively over Online Data Streams. Proceedings of KDD 2003.

[6] Chang, J. H.—Lee, W. S.: A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams. Journal of Information Science and Engineering, Vol. 20, 2004, No. 4, pp. 753–762.

[7] Chang, J. H.—Lee, W. S.: estWin: Online Data Stream Mining of Recent Frequent Itemsets by Sliding Window Method. Journal of Information Science, Vol. 31, 2005, No. 2, pp. 79-60.

[8] Chi, Y.—Wang, H.—Yu, P. S.—Muntz, R. R.: Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. Proceedings of the 4th IEEE Int'l Conf. on Data Mining, 2004.

[9] Datar, M.—Gionis, A.—Indyk, P.—Motwani, R.: Maintaining Stream Statistics over Sliding Windows. Proceedings of the 13th Annual ACM-SIAM Symp. On Discrete Algorithms, 2002.

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

[11] Manku, G. S.—Motwani, R.: Approximate Frequency Counts over Data Streams. Proceedings of VLDB, 2002.

[12] Cheng, J.—Ke, Y.—Ng, W.: Maintaining Frequent Itemsets over High-Speed Data Streams. In Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2006.

[13] Lin, C. H.—Chiu, D. Y.—Wu, Y. H.—Chen, A. L. P.: Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. SIAM International Conference on Data Mining, SDM-2005.

[14] Jin, R.—Agrawal, G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. In proceedings of the 5th IEEE International Conference on Data Mining, 2005.

[15] Karp, R. M.—Shenker, S.: A Sample Algorithm for Finding Frequent Elments in Streama and Bags. Journal of ACM Transactions on Database Systems, Vol. 28, 2003, pp. 51–55.

Congying Han received master degree in optimal control and operations research at Shandong University of Science and Technology in 2002, currently is a Ph. D. student at the Shanghai Jiaotong University, Department of Mathematics, and now works in School of Information Science and Engineering, Shandong University of Science and Technology. Her main areas of interest are data mining, pattern recognition, parallel computing, optimization algorithm and machine learning.

Lijun Xu received the Ph. D. degree in computer application at Shanghai JiaoTong University in Shanghai in 2006, currently is a technical professional at China Foreign Exchange Trade System. His main areas of interest are data mining, data warehouse and pattern recognition.

Guoping He graduated from Institute of Applied Mathematics, Chinese Academy of Sciences and received his M. Sc. and Ph. D. degrees from Chinese Academy of Sciences in 1988 and 1995 respectively. He was appointed professor by Shandong University of Science and Technology in 1996 and was engaged as professor and doctor advisor by Shanghai Jiaotong University in 2000. He was once the visiting professor at Hong Kong University of Science and Technology in 1999. He is the author and co-author of numerous scientific papers. His research interest includes data mining, pattern recognition, non-linear optimization and parallel algorithm.