

# Mining Frequent Itemsets Over Arbitrary Time Intervals in Data Streams

Chris Giannella<sup>1\*</sup>, Jiawei Han<sup>2\*\*</sup>, Edward Robertson<sup>1\*\*\*</sup>, Chao Liu<sup>2</sup>

<sup>1</sup> Department of Computer Science

Indiana University, Bloomington, IN 47404

{cgiannel,edrbtn}@cs.indiana.edu

<sup>2</sup> Department of Computer Science

University of Illinois, Urbana, IL 61801

{hanj,chaoliu}@cs.uiuc.edu

Received: / Revised version:

**Abstract** Mining frequent itemsets over a stream of transactions presents difficult new challenges over traditional mining in static transaction databases. Stream transactions can only be looked at once and streams have a much richer frequent itemset structure due to their inherent temporal nature. We examine a novel data structure, an FP-stream, for maintaining information about itemset frequency histories. At any time, requests for itemsets frequent over user-defined time intervals can be serviced by scanning the maintained FP-stream producing an approximate answer with error guaranteed to be no worse than a user-specified frequency and temporal threshold. We develop an algorithm for constructing and updating an FP-stream structure and present experiments illustrating the time and space required for maintenance.

**Keywords:** Frequent itemsets, data streams, tilted time windows

---

\* Contact author; supported by NSF grant IIS-0082407.

\*\* Supported by NSF grant IIS-0308215.

\*\*\* Partially supported by NSF grant IIS-0082407.

## 1 Introduction

Frequent itemset mining has been studied extensively in the last ten years. It has become one of the core sub-areas of data mining. Several algorithms have been developed and studied *e.g.* Apriori [3] and FP-growth [21] (see [1] and [20] for detailed surveys up to 2001). Ideas from frequent itemset mining and their associated methods have been used in a number of areas: association rule mining [3], sequential pattern mining [4], structured pattern mining [27], iceberg cube computation [6], cube gradient analysis [24], *etc.*

Recently, the mining and management of stream data has received considerable attention. In this model the data arrives as a potentially infinite stream of elements. It is assumed that the stream can only be scanned once, hence, once an element has passed, the element cannot be revisited unless it is stored in main memory. Some applications for which this model is appropriate include network traffic analysis, web click stream mining, power consumption measurement, sensor network data analysis, and dynamic tracing of stock fluctuation.

Several systems for managing and querying stream data are currently under development [5]. In addition to managing and querying data streams, mining streams for interesting patterns is also an important challenge. Recently, several studies on mining data streams have appeared including classification [15,17,23,25,32], clustering [2,19,29], and mining frequent itemsets [7,18,28].

In this paper, we study the problem of mining frequent itemsets over arbitrary time intervals on stream data. We examine a novel data structure, an FP-stream, for maintaining information about itemset frequency histories. This data structure is dynamically updated after each new batch of transactions arrives. At any time, a request for itemsets frequent over a user-defined interval can be serviced by scanning the maintained FP-stream. An approximate answer is given whose error is guaranteed to be no greater than user-specified frequency and temporal thresholds.

Itemset frequency histories are maintained using logarithmic tilted-time window tables. These tables store frequencies over exponentially increasing time granularities (*e.g.* every second for the last minute, every two seconds for the previous minute, every four seconds for the minute before that, *etc.*). Moreover, entries in the time window table are weighted by an *aging function*. Older entries are weighted less.

A tree structure (similar to the FP-tree [21]) is used to store a collection of itemsets with their tilted-time window tables. With the arrival of a new batch of transactions, the FP-stream is updated. Pre-existing itemsets may be dropped or part of their tilted-time window table entries may be dropped. New itemsets may

be added. The dropping condition guarantees that no itemset which is frequent over *arbitrary time intervals* consistent with pre-defined exponential time granularities will be absent from the FP-stream. Moreover, for each itemset in the structure, its frequency approximation will not be different than the true frequency by more than a user-defined fraction (taking into account age weighting).

Previous work [28] studied the problem of maintaining all frequent itemsets over the entire history of the stream. This will not be desirable in circumstances where the goals of mining the stream are time sensitive. For example, consider a shopping transaction stream that began a year ago. Itemsets frequent over the life of the stream will be of no use in detecting purchasing trends or combinations of items that have become popular only recently. As another example, in network monitoring, changes in the frequent patterns in the past several minutes are valuable and can be used for detection of network intrusions [14].

Our approach addresses these weaknesses by allowing the user, at any time, to issue queries requesting itemsets frequent over arbitrary time intervals.

We summarize the contributions of the paper. First, we describe a novel data structure, an FP-stream, for maintaining information about itemset frequency histories. We develop an algorithm to build and incrementally maintain an FP-stream. By scanning the maintained FP-stream, arbitrary time interval or arbitrary sliding window queries can be answered over data streams with user-defined error bound guarantees. We present experiments demonstrating the behavior of the algorithm over a variety of parameter settings.

**Paper Layout:** Section 2 describes related work, basic notation, and the goal we are trying to reach. Section 3 addresses the problem of maintaining frequency history information about a single itemset. Logarithmic tilted-time window tables are defined and their update algorithm presented. Section 4 describes how a time window query is answered for a single itemset. We describe how temporal and frequency errors arise and are managed. Section 5 addresses the problem of maintaining frequency history information about a collection of itemsets. The FP-stream data structure is presented. Section 6 describes the algorithm for updating an FP-stream and for answering queries. Section 7 describes the experimental set-up and data used to study the behavior of our algorithm. Section 8 reports the results of our experiments. Section 9 describes our previous work on the FP-stream data structure. Section 10 concludes the paper.

## 2 Related Work, Basic Notation, and Goal

### 2.1 Related Work

Recently we introduced the FP-stream data structure for mining frequent itemsets in streams over arbitrary time intervals [18]. However, we did not develop query answering techniques, we considered maintenance only for a restricted form of tilted-time window tables, and we did not consider age weighting on the time windows. Moreover, the details of the correctness proofs for the maintenance algorithms were not included. In this paper, we develop query answering, a maintenance algorithm for a more general class of tilted-time window tables (including age weights), and we provide complete details of all correctness proofs. See Section 9 for a more detailed description of the differences between [18] and this paper.

Manku and Motwani [28] propose a method for mining all itemsets frequent over the life of the stream. They store itemsets in a tree structure. With each itemset stored, they record an under-estimate of the frequency and an error term. Upon the arrival of a new batch of transactions, the tree is updated; some itemsets may be dropped and others may be added. The dropping and adding conditions ensure that: (1) the estimated frequency of all itemsets in the tree are less than the true frequency by no more than  $\epsilon N$  ( $N$  is the length of the stream and  $\epsilon$  is a user-defined error bound) and (2) no frequent itemset is omitted from the tree. By scanning the tree and returning all itemsets whose estimated frequency is greater than  $(\sigma - \epsilon)N$  ( $\sigma$  is the support threshold), an approximation<sup>1</sup> to the frequent itemsets can be produced. We address a more general problem than Manku and Motwani: produce itemsets frequent over arbitrary time intervals and over sliding time intervals.

Chang and Lee [7] develop an algorithm for maintaining frequent itemsets in stream data assuming each transaction has a weight. The weights decrease with age; in affect, older transactions contribute less toward itemset frequencies. They maintain a tree of itemsets and use some basic anti-monotonicity inequalities to reduce update cost. Like us, they take into account age when maintaining itemsets. However, they do not allow the user to query for frequent itemsets over many time intervals (as we do). They provide itemsets frequent over the life of the stream (similar to [28]). We address a more general problem.

Datar *et al.* [12] propose a method for maintaining aggregates involving the last  $W$  elements in a stream (a sliding window). They consider the following basic problem and show how an algorithm for it can be adapted to

---

<sup>1</sup> The estimated frequency of each itemset will be an approximation of the true frequency *and* some itemsets will be returned which are not truly frequent.

more general problems: given a stream of 1s and 0s, maintain a count of the 1s over the last  $W$  elements. Their approach is to maintain a histogram with  $O(\log_2(W))$  buckets each recording the number of 1s in a collection of consecutive elements (covering all elements in  $W$ ). They do not consider the problem of maintaining frequent itemsets over the last  $W$  transactions. But their histograms (called exponential histograms) are similar to our logarithmic tilted-time window tables.

Cohen and Strauss [10] consider a generalization of the basic problem of Datar *et al.*: maintain a weighted count of 1's encountered thus far in the stream. The weights are a monotonically increasing function of age (a sliding window of size  $W$  can be expressed as the last  $W$  time units get weight one, all older get weight zero). Cohen and Strauss develop algorithms for maintaining an approximation to the weighted count in small space for several classes of weighting functions. They do not consider the problem of maintaining frequent itemsets over age weighted transactions. But their work is similar to ours in that we consider age weighted batches of transactions.

Cormode and Muthukrishnan [11] develop a one-pass, probabilistic algorithm for addressing the following problem over a stream of *item* insertions and deletions. Given integer  $k$  and reals  $\delta, \epsilon$ , produce at any time with probability  $1 - \delta$  the items whose frequency is at least  $1/(k + 1)$  times the number of items seen thus far (less deletions); they call these “hot” items. Their solution requires space  $O(k \log k + k \log(1/\delta))$  (independent of the stream length).

Charikar *et al.* [8] develop a one-pass randomized algorithm for addressing the following problem on streams of *items*. Given integer  $k$  and reals  $\delta, \epsilon$ , maintain a list of  $k$  items such that, with probability  $1 - \delta$ , every item in the list had frequency at least  $1 - \epsilon$  times the frequency of the  $k^{th}$  most frequent item. Their solution requires space at most logarithmic in the length of the stream, linear in  $k$ , and linear in the sum of the squares of the top  $k + 1$  frequencies.

The problems addressed in [11] and [8] differ from ours in two ways. First, we consider itemsets rather than just items. While itemsets could be encoded as individual items then the methods of [11] and [8] applied, it remains to be seen the potential difficulties that might arise. A study along these lines would be quite interesting. Second, we consider the discovery of frequent itemsets over arbitrary time intervals rather than the life of the stream. It is not clear how the methods of [11] and [8] can be extended to address this problem.

Several two-pass algorithms for frequent itemset mining have been developed: [22], [30], [31] (see [28] section 2.3 for a more detailed survey of two pass algorithms). Recently a two-pass algorithm for finding all

frequent *items* in a stream has been developed independently in [13] and [26]. This algorithm requires very small space and time. However, stream mining requires one pass algorithms, so all of these two-pass methods are not directly applicable.

A variety of other problems have been addressed on stream data: random sample maintenance, estimation of frequency moments, estimation of  $L_p$  norms, histogram and wavelet construction, quantile computation, *etc.*. See [5] for a good survey of methods for these problems up to 2002.

## 2.2 Basic Notation

Let  $\tau_0, \tau_1, \dots, \tau_N$  denote the time units elapsed thus far in the stream,  $\tau_0$  is the oldest,  $\tau_N$  most recent. Given integers  $N \geq b, a \geq 0$ , let  $T_a^b$  denote the time interval consisting of the time units  $\tau_a, \tau_{b-1}, \dots, \tau_b$ .<sup>2</sup>  $T_a$  is used as shorthand for the interval  $T_a^a$  consisting of time unit  $\tau_a$ . The length of  $T_a^b$ , denoted  $||T_a^b||$ , is the number of time units in  $T_a^b$ , *i.e.*  $||T_a^b|| = b - a + 1$ .

Let  $\mathcal{I}$  denote a set of items. An itemset (transaction) is a non-empty subset of  $\mathcal{I}$ . A stream,  $\mathbb{S}$ , is a sequence of transactions:  $s_0, s_1, \dots$  ( $s_0$  is the oldest). More than one stream transaction can arrive during a time unit. Given any collection of consecutive transactions  $S$  from  $\mathbb{S}$  and an itemset  $I$ , let  $f_I(S)$  denote the number of transactions  $s_i$  in  $S$  such that  $I \subseteq s_i$  (*i.e.* the frequency of  $I$  over  $S$ ). We sometimes write  $f(S)$  as shorthand for  $f_I(S)$  when clear from context.  $|S|$  denotes the number of transactions in  $S$ .

The collection of transactions from  $\mathbb{S}$  that arrived during the  $i^{\text{th}}$  time unit ( $i \geq 0$ ) is called the  $i^{\text{th}}$  batch and is denoted  $B_i$ . Notice that the batches may be of differing sizes. For  $b, a \geq 0$ , let  $B_a^b$  denote  $\bigcup_{i=a}^b B_i$ .<sup>3</sup>  $f_I(T_a^b)$  denotes the frequency of itemset  $I$  over  $T_a^b$ , formally stated,  $f_I(T_a^b) = f_I(B_a^b)$ .

Let  $\phi : \mathbb{N} \rightarrow [1, \infty)$  be called the *aging function*; it assigns an age to collections of batches. The age of collection  $B_a^b$  at time unit  $\tau_N$  is  $\phi(N - a)$ . We assume that a collection consisting of the newest batch,  $B_N^N$ , has age one ( $\phi(0) = 1$ ) and that age is monotonically increasing ( $\phi(x) \leq \phi(y)$ , if  $x \leq y$ ).

Let  $\gamma \geq 0$  and assume that time unit  $\tau_N$  has just elapsed.  $I$  is said to be  $\gamma$ -frequent over  $T_a^b$  at  $N$  if  $f_I(T_a^b) \geq \phi(N - a)\gamma|B_a^b|$ . This is a generalization of the standard notion of frequent itemsets. If  $\phi = 1$  (maps always to one), then  $\gamma$ -frequent over  $T_a^b$  at  $N$  is equivalent to the standard meaning of  $\gamma$ -frequent.

---

<sup>2</sup> If  $a > b$ , then  $T_a^b$  is the empty time interval.

<sup>3</sup> If  $a > b$ ,  $B_a^b$  is the empty collection of transactions.

### 2.3 Goal

Let  $1 \geq \sigma > 0$  be called the support threshold and  $\sigma > \epsilon_f \geq 0$  be called the frequency error threshold. Often an itemset will be said to be frequent over a collection of transactions at  $N$  (the current time unit) as shorthand for  $\sigma$ -frequent at  $N$ . Let  $1 \geq \epsilon_t \geq 0$  be called the temporal error threshold (explained below).

Our goal is to develop an algorithm which, at any time unit  $\tau_N$ , can be issued a query requesting itemsets frequent over  $T_a^b$  at  $N$  and will produce an approximate answer: a set of itemsets and their approximate frequencies over  $T_{\hat{a}}^{\hat{b}}$  (an approximation of  $T_a^b$ ). The approximate answer must satisfy three error guarantees with respect to thresholds  $\epsilon_f$  and  $\epsilon_t$  as explained next.

There are two types of errors that can occur in the approximate answer: *temporal error* and *frequency error*. Temporal error results from the fact that the frequencies returned are assumed to be over a time period  $T_{\hat{a}}^{\hat{b}}$ . We define the temporal error as the sum of the differences between the starting and ending points of each interval:

$$|b - \hat{b}| + |a - \hat{a}|. \quad (1)$$

Frequency error results from the fact that the frequencies returned with the itemsets are approximations to the true frequencies over  $T_a^b$ . For each itemset  $I$  returned, let  $\hat{f}_I(T_{\hat{a}}^{\hat{b}})$  denote the approximate frequency. We define the frequency error of  $I$  to be the difference between the true and approximate frequency:

$$|f_I(T_a^b) - \hat{f}_I(T_{\hat{a}}^{\hat{b}})|. \quad (2)$$

The result is required to meet the following error guarantees (assuming time unit  $\tau_N$  has just elapsed):

1.  $\frac{|b-\hat{b}|}{N-\hat{b}+1} + \frac{|a-\hat{a}|}{N-\hat{a}+1} \leq \epsilon_t$ ;
2. for any itemset  $I$ , if  $I$  is  $\sigma$ -frequent at  $N$ , then  $I$  is in the result;
3. for any itemset  $I$  in the result,  $f_I(T_a^b) \geq \hat{f}_I(T_{\hat{a}}^{\hat{b}}) > f_I(T_a^b) - \phi(N - \hat{a})\epsilon_f|B_{\hat{a}}^{\hat{b}}|$ .

The first guarantee ensures that the temporal error, normalized to account for the age of the query, does not exceed a pre-defined threshold. The second ensures that no frequent itemsets are omitted from the returned result. The third ensures that the frequency error of any itemset does not exceed a pre-defined threshold (taking into account aging) and that the approximate frequency does not exceed the true frequency. The second and third guarantees are similar to that in [28].

### 3 Logarithmic Tilted-Time Window Tables

This section addresses the issue of maintaining frequency history information about a single itemset. We maintain exact itemset frequencies. In Section 4 we describe how approximations to the maintained frequencies are introduced to save storage space and in Section 5 we describe how to extend our approach to collections of itemsets. The complete algorithm for answering queries is described in Section 6.2.

In order to maintain itemset frequency history information, we use a *tilted-time window framework* [9]. Its design is based on the fact that people are likely interested in recent changes at fine time granularities, but old changes at coarse granularities. We use a *logarithmic* tilted-time frame. Here the granularities from most fine to most coarse are: one unit, two units, four units, *etc.*

#### 3.1 Definition

A logarithmic tilted-time window table is based on a stratified partition of the time units that have passed. The partition represents levels of increasing time granularity and consists of a collection of time windows (intervals):  $T_{a_m}^{b_m}, T_{a_{m-1}}^{b_{m-1}}, \dots, T_{a_0}^{b_0}$  where  $b_i \geq a_i, 0 \leq i \leq m$ . These windows are consecutive and cover all units of time seen thus far:  $b_m = N, a_0 = 0$ , and  $a_i = b_{i-1} + 1$  for  $1 \leq i \leq m$ . Moreover, the windows are stratified in terms of their length. Each strata is called a *granularity level*. The first granularity level consists of  $T_{a_m}^{b_m}, \dots, T_{a_{\ell_1}}^{b_{\ell_1}}$ ; the second consists of  $T_{a_{\ell_1-1}}^{b_{\ell_1-1}}, \dots, T_{a_{\ell_2}}^{b_{\ell_2}}$ ; the  $k^{th}$  consists of  $T_{a_{\ell_{k-1}-1}}^{b_{\ell_{k-1}-1}}, \dots, T_{a_0}^{b_0}$ .

At any level  $1 \leq L \leq k$ , all of the windows have the same time length  $2^{L-1}$ . This is called the granularity of level  $L$ . For example,  $\|T_{a_m}^{b_m}\| = \dots = \|T_{a_{\ell_1}}^{b_{\ell_1}}\| = 1$ . Each level  $L$  is assumed to contain at most  $max_L \geq 2$  windows and at least  $max_L - 1$  (except before the first  $max_L - 1$  windows at level  $L$  are created, *e.g.* before the first  $max_1 - 1$  batches are encountered, level one will have less than  $max_1 - 1$  windows).  $max_1, max_2, \dots, max_u$  ( $u \geq 1$ ) are user specified constants used to control the granularity and size of the table; they are called *granularity constants*. If  $max_u = \infty$ , then the table contains only  $u$  levels. The  $u^{th}$  level grows indefinitely long. If  $max_u \neq \infty$ , then the table contains an unbounded number of levels but the  $u^{th}$  level and every level after are assumed to contain at most  $max_u$  windows and at least  $max_u - 1$ .

The batch sizes associated with windows at each level are stored in the *global partition structure*. This structure records a list of batch sizes at each level:  $|B_{a_m}^{b_m}|, \dots, |B_{a_{\ell_1}}^{b_{\ell_1}}|; |B_{a_{\ell_1-1}}^{b_{\ell_1-1}}|, \dots, |B_{a_{\ell_2}}^{b_{\ell_2}}|; \dots \dots; |B_{a_{\ell_{k-1}-1}}^{b_{\ell_{k-1}-1}}|, \dots, |B_{a_0}^{b_0}|$ . The time window lengths need not be recorded since all windows at level  $L$  have length  $2^{L-1}$ .

The logarithmic tilted table for every itemset,  $I$ , records a list of frequencies at each level  $f_I(T_{a_m}^{b_m}), \dots, f_I(T_{a_{\ell_1}}^{b_{\ell_1}}); f_I(T_{a_{\ell_1-1}}^{b_{\ell_1-1}}), \dots, f_I(T_{a_{\ell_2}}^{b_{\ell_2}}); \dots; f_I(T_{a_{\ell_{k-1}-1}}^{b_{\ell_{k-1}-1}}), \dots, f_I(T_{a_0}^{b_0})$ . Note: there is only one global partition structure but a separate tilted time window table for each itemset.

### 3.2 Updating

Next we show how the global partition structure is updated.

Upon arrival of a new batch,  $B_N$ , a new entry  $|B_N^N|$  is created for level one. If level one has less than  $max_1 + 1$  entries (after the creation of the new entry), then the algorithm stops. If level one has  $max_1 + 1$  entries then the last (oldest) two entries  $|B_{N-max_1}^{N-max_1}|$  and  $|B_{N-max_1+1}^{N-max_1+1}|$  are merged and the result becomes a new entry at level two  $|B_{N-max_1}^{N-max_1+1}|$ . The merger removes the last two entries from level one. If level two has fewer than  $max_2 + 1$  entries, then the algorithm stops. Otherwise, the last two entries at level two are merged and the result becomes a new entry at level three. This process continues until a level is reached with fewer than  $max_2 + 1$  entries after the addition of the new entry.

To illustrate, consider the following global partition table ( $max_1 = 2$ ) after three batches have arrived:  $|B_2^2|; |B_0^1|$ . This table has two levels, each allowing at most  $max_1 = 2$  entries.

Upon arrival of the fourth batch, a new entry is created for level 1. Since there are not enough entries at level 1 ( $2 < max_2 + 1 = 3$ ), then a merger does not occur. The table appears as:  $|B_3^3|, |B_2^2|; |B_0^1|$ .

Upon arrival of the fifth batch, a new entry is created for level 1. There are enough entries for a merger, so the last two entries are merged a new entry is created for level 2. Level 2 now has two entries (so no merger occurs). The table appears as:  $|B_4^4|; |B_2^3|; |B_0^1|$ .

The updating method for the tilted-time window table for each itemset is analogous. After the first three batches have arrived, the table is  $f_I(T_2^2), f_I(T_0^1)$ . After the arrival of the fifth batch the table is  $f_I(T_4^4), f_I(T_2^3), f_I(T_0^1)$ . Each window in the table represents the frequency of its corresponding entry in the partition structure.

## 4 Answering Queries

Our goal (Section 2.3) is to develop a method which, given a user-defined query for itemsets frequent over  $T_a^b$  at  $N$ , will produce an approximate answer satisfying certain error guarantees. As a first step, we describe how this goal can be met for a single itemset,  $I$ . In this section, we describe how queries are answered over a

single tilted-time window table (for  $I$ ). In Section 5 we discuss how the method is extended to collections of itemsets.

We maintain the global partition structure and the tilted-time window table for  $I$ . When the user issues a query, the first step is to scan the global partition structure and find the best time window endpoints to approximate  $a$  and  $b$ . Let  $T_{a_m}^{b_m}, T_{a_{m-1}}^{b_{m-1}}, \dots, T_{a_0}^{b_0}$  be the tilted-time windows represented by the global partition table. If  $a$  and  $b$  are not in the same window then set  $\hat{a}$  to the closest  $a_i$ , i.e.  $i = \operatorname{argmin}\{|a - a_i| : 0 \leq i \leq m\}$  (if two such  $i$ 's exist, choose the smaller). Set  $\hat{b}$  to the closest  $b_j$ , i.e.  $j = \operatorname{argmin}\{|b - b_j| : 0 \leq j \leq m\}$  (if two such  $j$ 's exist, choose the smaller). Note, it's possible that  $\hat{a} > \hat{b}$ .

If  $a$  and  $b$  are in the same window,  $T_{a_k}^{b_k}$ , then  $\hat{a}$  and  $\hat{b}$  are not set as described above. In this case, if  $b - a + 1 < \|T_{a_k}^{b_k}\|/2$ , set  $\hat{b}$  to  $b$  and  $\hat{a}$  to  $\hat{b} + 1$  (the empty time interval). Otherwise, set  $\hat{b}$  to  $b_k$  and  $\hat{a}$  to  $a_k$ .

Once  $\hat{a}$  and  $\hat{b}$  have been computed, the tilted-time window table is scanned to produce the desired frequency.

#### 4.1 Temporal Approximation

We develop an upper-bound on the temporal error in terms of the granularity constants  $\max_1, \dots, \max_u \geq 2$  (Theorem 1). This theorem shows how the granularity constants can be set to ensure the temporal error bound guarantee described in Section 2.3 (guarantee 1) is met (Corollary 1). Theorem 2 shows how setting the constants affects the space required to store the partition table.

If  $\max_u \neq \infty$ , let  $Mn = \min\{\max_1, \dots, \max_u\}$  and  $Mx = \max\{\max_1, \dots, \max_u\}$ , otherwise, if  $u > 1$ , let  $Mn = \min\{\max_1, \dots, \max_{u-1}\}$  and  $Mx = \max\{\max_1, \dots, \max_{u-1}\}$ . Recall that  $N$  is the number of time units that have elapsed thus far.

**Theorem 1** Let  $Terr = \frac{|b-\hat{b}|}{N-\hat{b}+1} + \frac{|a-\hat{a}|}{N-\hat{a}+1}$ . Assume  $a$  appears inside window  $T_{a_i}^{b_i}$  at level  $L_i$  and  $b$  inside window  $T_{a_j}^{b_j}$  at level  $L_j$  (note,  $a_j \geq b_i$  and  $L_j \leq L_i$ ).

1. If  $L_i = L_j = 1$ , then  $Terr = 0$ .
2. If  $L_j = 1, L_i > 1$ , then  $Terr \leq \frac{1}{Mn-1} \left[ \frac{2^{L_i-2}}{2^{L_i-1}-1} \right]$ .
3. If  $L_i, L_j > 1$ , then  $Terr \leq \frac{1}{Mn-1} \left[ \frac{2^{L_j-2}}{2^{L_j-1}-1} + \frac{2^{L_i-2}}{2^{L_i-1}-1} \right]$ .

**Proof:** 1. If  $L_i = L_j = 1$ , then  $\|T_{a_i}^{b_i}\| = \|T_{a_j}^{b_j}\| = 1$ , thus,  $a = a_i$  and  $b = b_i$ . Therefore,  $\hat{a} = a$  and  $\hat{b} = b$ .

2. If  $L_j = 1, L_i > 1$ , then  $\hat{b} = b$  so  $Terr = \frac{|a-\hat{a}|}{N-\hat{a}+1}$ . It can be seen from the way  $\hat{a}$  is defined, that  $|a - \hat{a}| \leq \|T_{a_i}^{b_i}\|/2 = 2^{L_i-2}$ . We assert that  $N - a - 1$  is bounded below by  $(Mn - 1)2^{L_i-1} - 1$ . Since  $L_i, Mn > 1$ , then this bound is greater than zero. Hence, the desired result follows.

To see the assertion, consider that  $N - a - 1$  is the number of time units  $a$  lies from the current time unit. This is bounded below by the number of time units in levels  $1, \dots, L_i - 1$ . Let  $mx_k$  denote the minimum number of windows that must appear in level  $1 \leq k \leq L_i - 1$ . If  $k \leq u$ , then  $mx_k = max_k - 1$ , otherwise  $mx_k = max_u - 1$ .<sup>4</sup> Clearly,  $Mn - 1 \leq \min\{mx_k : 1 \leq k \leq L_i - 1\}$ . Since each window at level  $k$  has  $2^{k-1}$  units, then

$$\begin{aligned} N - a - 1 &\geq \sum_{k=1}^{L_i-1} mx_k 2^{k-1} \\ &\geq (Mn - 1) \sum_{k=1}^{L_i-1} 2^{k-1} \\ &= (Mn - 1)[2^{L_i-1} - 1]. \end{aligned}$$

3. An analogous argument to part 2 shows the desired result.  $\square$

**Corollary 1** *If  $Mn \geq \frac{2}{\epsilon_t} + 1$ , then guarantee 1 in Section 2.3 is met. Moreover, for the class of sliding window queries (i.e.  $b = N$ ), the guarantee is met if  $Mn \geq \frac{1}{\epsilon_t} + 1$ .*

A larger  $Mn$  implies that a smaller  $\epsilon_t$  guarantee can be met. However, a larger  $Mn$  implies a larger partition structure (and hence a larger tilted time window table).

**Theorem 2** *Let  $m$  denote the number of windows that appear in the partition table and  $L$  denote the last level.*

1. *Assume  $max_u \neq \infty$ , we have  $m \geq (Mn - 1)(\log_2(\frac{N}{Mx} + 1) - 1)$ .*
2. *Assume  $max_u = \infty$ . If  $L = 1$ , then  $m = N$ . If  $1 < L < u$ , then  $m \geq (Mn - 1)(\log_2(\frac{N}{Mx} + 1) - 1)$ . If  $1 < L = u$ , then  $m \geq (Mn - 1)(2^{u-1} - 1) + \frac{N}{2^{u-1}} - Mx(1 - \frac{1}{2^{u-1}})$ .*

**Proof:** 1. The number of time windows in levels  $1, \dots, L - 1$  is a lower bound on  $m$ . The minimum number of windows that may appear in each level  $1 \leq k \leq L - 1$ , is  $mx_k$  (see proof of Theorem 1 part 2). Hence

$$m \geq \sum_{k=1}^{L-1} mx_k \geq (Mn - 1)(L - 1). \quad (3)$$

---

<sup>4</sup> The number of windows in level  $L_i$  may be less than  $mx_u - 1$  if  $L_i$  is the last level (not enough time units may have passed to fill level  $L_i$ ). However, enough time units will have passed to fill all before the last, including levels  $1, \dots, L_i - 1$ .

$L$  must be large enough to cover all  $N$  time units. The maximum number of time units that could be contained in a table with  $L$  levels is  $\sum_{k=1}^L (max_k + 1)2^{k-1} \leq Mx \sum_{k=1}^L 2^{k-1} = Mx(2^L - 1)$ . Hence  $L$  must be at least  $\log_2(\frac{N}{Mx} + 1)$ . Plugging into (3) we get  $m \geq (Mn - 1)(\log_2(\frac{N}{Mx} + 1) - 1)$  as desired.

2. If  $L = 1$ , then all windows in the table are of size one, hence  $m = N$ . Assume  $1 < L < u$ , then the same argument from part 1 can be applied.

Assume  $1 < L = u$ . The minimum number of windows that appear in levels  $1, \dots, u - 1$  is  $\sum_{k=1}^{u-1} (max_k - 1)2^{k-1} \geq (Mn - 1)(2^{u-1} - 1)$ . This is also a lower bound on  $m$ , but not a very good one since the  $u^{th}$  level may be quite large. To improve the bound we develop a rough lower-bound on the number of windows in the  $u^{th}$  level. The number of time units in the  $u^{th}$  level is at least  $N - \sum_{k=1}^{u-1} max_k 2^{k-1} \leq N - Mx(2^{u-1} - 1)$ . Thus the number of time windows at the  $u^{th}$  level is at least

$$\frac{N - Mx(2^{u-1} - 1)}{2^{u-1}} = \frac{N}{2^{u-1}} - Mx(1 - \frac{1}{2^{u-1}}).$$

□

To get a better sense of the trade-off between meeting the temporal error guarantee and the size of the table, consider the case where  $u = 1$ ,  $max_1 \neq \infty$ ,  $\epsilon_t = 0.1$ , and assume  $10^6$  time units have elapsed (if the units are minutes, then 695 days have elapsed). To meet the guarantee,  $max_1 \geq 21$  (or  $max_1 \geq 11$  for sliding window queries). The partition table, and hence the tilted time window table will contain at least 291 windows (or at least 155 for sliding window queries).

#### 4.2 Frequency Approximation

We describe next how parts of the tilted-time window table for a fixed  $I$  are maintained. Table entries are dropped (and new entries ignored) to save space at the expense of producing frequency approximations. While this space is not critical for maintaining a single tilted-time window table, it likely will be for maintaining the collection of itemsets required to address a query requesting *all* itemsets frequent over  $T_a^b$  at  $N$ .

To reduce the storage space required, we selectively drop parts of the tilted time window table and selectively ignore new entries. The dropping and adding conditions are designed to ensure that the guarantees from Section 2.3 related to frequency error (2 and 3) are met.

Suppose the tilted-time window is empty. Upon arrival of a new time unit  $\tau_N$ ,  $|B_N^N|$  is always added to the global partition structure. However,  $f_I(T_N^N)$  is added to the table only if the following condition is met.

**Adding Condition 1**  $f_I(T_N^N) \geq \epsilon_f |B_N^N|$ .

If the tilted-time window table is not empty,  $f_I(T_N^N)$  is always added, but may be dropped as described next.

Let  $f_I(T_{a_m}^{b_m}), \dots, f_I(T_{a_0}^{b_0})$  denote the time windows in the table. We drop tail  $f_I(T_{a_k}^{b_k}), \dots, f_I(T_{a_0}^{b_0})$ , for some  $0 \leq k \leq m$ , if a certain dropping condition is satisfied. When issued a query over  $T_{a_i}^{b_j}$ , an approximation  $\hat{f}_I(T_{a_i}^{b_j})$  is returned. If  $k = m$  (the entire tilted-time window table is dropped), then  $\hat{f}_I(T_{a_i}^{b_j})$  is defined to be zero, otherwise  $\hat{f}_I(T_{a_i}^{b_j})$  is defined to be  $f_I(T_{a'}^{b_j})$  where  $a' = \max\{a_i, a_{k+1}\}$ . If  $m \geq i > k$ , the approximation is exact, however, if  $k \geq i \geq 0$ , then the approximation may be less than the true frequency,  $f_I(T_{a_i}^{b_j})$ .

Once windows have been dropped, an additional problem may emerge during updating. Recall that the tilted-time window table is updated along side the global partition structure. Moreover, each window in the table is assumed to represent the frequency over its corresponding entry in the partition structure. Once windows have been dropped and the table and partition are updated with new time units, every window in the table may not match the frequency over its corresponding entry in the partition. Consider the following example.

**Example 1** *Suppose  $u = 1$ ,  $\max_u = 2$ , and four time units have arrived. The global partition is  $|B_3^3|, |B_2^2|, |B_0^1|$  and the tilted table is  $f_I(B_3^3), f_I(B_2^2); f_I(B_0^1)$ . Suppose the last two windows are dropped. When the next time unit arrives, the partition is updated to  $|B_4^4|, |B_2^3|, |B_0^1|$  and the table becomes  $f_I(B_4^4); f_I(B_3^3)$ . The corresponding entry in the partition for  $f_I(B_3^3)$  is  $B_2^3$  ( $f_I(B_3^3)$  is the second entry in the table and  $B_2^3$  is the second entry in the partition). However, the frequency over  $B_2^3$  may not be the same as  $f_I(B_3^3)$ .*

This inaccuracy is inevitable, but the next theorem shows that only the last (oldest) window in the table is inaccurate.

**Theorem 3** *Suppose that a global partition structure and a corresponding tilted-time window table are maintained; periodically tails of the table are dropped. After each time unit is processed (the structure and table are updated, dropping is carried out, if necessary), all windows in the table, except the last (oldest), match the frequency over their corresponding entry in the partition structure.*

**Proof:** By induction on the number of time units that have elapsed. In the base case, only one time unit has elapsed; clearly the result holds.

Now assume the result holds immediately after time unit  $i > 1$  has elapsed. The processing of the  $(i + 1)^{st}$  unit consists of two steps: (i) insert  $|B_{i+1}|$  into the partition and  $f_I(T_{i+1})$  into the table; (ii) drop a tail of the

table, if the dropping conditions are met. Let  $W_m, \dots, W_1$  denote the time windows in the table immediately after step (i) of the  $(i+1)^{st}$  unit is processed ( $W_m$  is the most recent window).

$W_m$  is  $I$ 's frequency over the  $(i+1)^{st}$  time unit and its corresponding entry in the structure is the batch at the  $(i+1)^{st}$  time unit. Hence,  $W_m$  matches the frequency over its corresponding entry in the partition at the end of step (i). Now consider  $W_j$  with  $1 < j < m$ . It was produced in one of two ways:  $W_j$  is simply one of the windows,  $\overline{W}_j$ , among those at the end of the  $i^{th}$  unit;  $W_j$  if formed by merging two consecutive windows  $\overline{W}_j^1, \overline{W}_j^2$  among those at the end of the  $i^{th}$  unit. In either case, since  $W_j$  is not the last window at the end of step (i), then none of  $\overline{W}_j$ ,  $\overline{W}_j^1$ , and  $\overline{W}_j^2$  is the last window at the end of the  $i^{th}$  time unit. By induction, all of these windows match the frequency over their corresponding entries in the partition table at the end of unit  $i$ . Therefore,  $W_j$  matches the frequency over its corresponding entry in the partition at the end of step (i) during the  $(i+1)^{st}$  unit.

If a tail is dropped during step (ii), clearly the last window not dropped is among  $W_j$ ,  $1 < j \leq m$  (since  $W_1$  is the oldest and must have been dropped). Thus, the result holds.  $\square$

The last window in the table represents the frequency over some batch of transactions (not necessarily that over its corresponding entry in the partition). To account for this, we store the batch size of the batch matching the frequency in the tilted-time window table; we call this *the last batch entry*. Therefore, each window in the table (except the last) matches the frequency over its corresponding entry in the partition structure and the last window match the frequency over the last batch entry (stored in the tilted-time window table). Consider Example 1. The last window is  $f_I(B_3^3)$ . So we would store  $|B_3^3|$  as the last batch entry.

**Tilted-time window table update algorithm:** Upon arrival of a new time unit  $\tau_N$ , the partition structure is updated exactly as described in Section 3.2. The tilted-time window table is updated as follows. If the table is empty, then insert  $f_I(T_N^N)$  into the table if the adding condition is met. Store  $|B_N^N|$  in the table as the last batch entry. If the table is not empty, then insert  $f_I(T_N^N)$ . The merging of windows follows the merging done in inserting  $|B_N^N|$  into the partition structure with some additional work done to maintain the last batch. We illustrate with an example.

**Example 2** Let  $u = 1$ ,  $max_u = 2$ . After the first two time units have arrived (assume no dropping and that the adding condition is met), the global partition structure is  $|B_1^1|, |B_0^0|$  and the tilted-time window table is  $f_I(T_1^1), f_I(T_0^0)$  with last batch entry  $|B_0^0|$ . Upon arrival of the next time unit the partition structure and the table are updated. The partition becomes  $|B_2^2|; |B_0^0|$  and the table becomes  $f_I(T_2^2); f_I(T_0^0)$ . The last batch entry

must be updated to reflect the merger of  $f_I(T_0^0)$  and  $f_I(T_1^1)$ . It is updated to  $|B_0^1|$  by adding the last batch from the previous step ( $|B_0^0|$ ) to the batch size for  $f_I(T_1^1)$ ,  $|B_1^1|$  (the partition table contains  $|B_1^1|$ ). Next we check if the dropping conditions apply (assume they do not).

Upon arrival of the fourth time unit the partition and table are updated to  $|B_3^3|, |B_2^2|; |B_0^1|$  and  $f_I(T_3^3)$ ,  $f_I(T_2^2)$ ;  $f_I(T_0^1)$  with last batch entry  $|B_0^1|$ . Suppose the dropping conditions apply and the last two windows are dropped. The table becomes  $f_I(T_3^3)$  and the last batch becomes  $|B_3^3|$  (the partition table contains  $|B_3^3|$ ). Upon arrival of the fifth batch (assume no dropping) the partition and table become  $|B_4^4|; |B_2^3|, |B_0^1|$  and  $f_I(T_4^4)$ ;  $f_I(T_3^3)$  with last batch entry  $|B_3^3|$ .

**Lemma 1** *Suppose that a global partition structure and a corresponding tilted-time window table are maintained as described above. After each time unit is processed all windows in the table, except the last (oldest), match the frequency over their corresponding entry in the partition structure; the last matches the frequency over the batch corresponding to the last batch entry in the tilted-time window table.*

**Proof:** Follows from Theorem 3. □

### 4.3 Dropping Condition

This condition is designed to ensure that frequency error guarantee 3 in Section 2.3 is met. Let  $f_I(T_{a_m}^{b_m})$ ,  $\dots$ ,  $f_I(T_{a_0}^{b_0})$  denote the time windows in the table ( $T_{a_0}^{b_0}$ ) is the time window corresponding to the last batch entry; assume time unit  $\tau_N$  has just elapsed. Consider the following condition on tail  $f_I(T_{a_k}^{b_k})$ ,  $\dots$ ,  $f_I(T_{a_0}^{b_0})$  ( $0 \leq k \leq m$ ).

**Dropping Condition 1 (Arbitrary Query)** *For all  $0 \leq i \leq k$ ,  $f_I(T_{a_i}^{b_i}) < \phi(N - a_i)\epsilon_f |B_{a_i}^{b_i}|$ ;*

Notice that computation of the dropping condition requires knowledge of the sizes of the batches associated with each time window. Lemma 1 shows that these computations can be carried out using the global partition table and the tilted-time window table (with last batch entry).

Recall that to answer a query over  $T_a^b$ , we first compute  $T_a^{\hat{b}}$ , then scan the tilted-time window table over  $T_a^{\hat{b}}$ . The next theorem shows that guarantee 3 is met.

**Theorem 4** *Suppose that a global partition structure and a corresponding tilted-time window table are maintained as described in Section 4.2. Assume that  $N$  time units have passed. Then for any  $T_a^b$ ,  $f_I(T_a^b) \geq \hat{f}_I(T_a^{\hat{b}}) > f_I(T_a^{\hat{b}}) - \phi(N - \hat{a})\epsilon_f |B_a^{\hat{b}}|$ .*

**Proof:** If  $\hat{a} > \hat{b}$ , then  $f_I(T_{\hat{a}}^{\hat{b}}) = 0 = \hat{f}_I(T_{\hat{a}}^{\hat{b}})$ , so the result holds. Assume  $\hat{a} \leq \hat{b}$ .

Let  $B_N, B_{N-1}, \dots, B_0$  represent the batches that have arrived thus far (time units  $\tau_N, \dots, \tau_0$ ). We treat the case of an incoming batch ignored due to adding conditions as if it were inserted into the tilted-time window table, then immediately dropped. Hence, we can model the behavior of the algorithm as a series of pairs  $(k_p, d_p), \dots, (k_1, d_1)$  indicating  $B_{k_i}, \dots, B_{k_{(i-1)}+1}$  were dropped at the end of time unit  $d_i$  for  $1 \leq i \leq p$  (assuming  $k_0 = -1$ ). Take note:  $N \geq k_p > k_{p-1} > \dots > k_1 \geq 0$  and  $d_i \geq k_i$ .

If  $\hat{a} > k_p$ , then  $\hat{f}_I(T_{\hat{a}}^{\hat{b}}) = f_I(T_{\hat{a}}^{\hat{b}})$ , so the result holds. Assume  $\hat{a} \leq k_p$ . Let  $Mn = \min\{k_p, \hat{b}\}$ . We have  $f_I(T_{\hat{a}}^{\hat{b}}) = f_I(T_{\hat{a}}^{Mn}) + f_I(T_{k_p+1}^{\hat{b}})$ . By definition of  $\hat{f}_I$  it follows that  $f_I(T_{\hat{a}}^{\hat{b}}) = f_I(T_{\hat{a}}^{Mn}) + \hat{f}_I(T_{\hat{a}}^{\hat{b}})$ . Hence  $f_I(T_{\hat{a}}^{\hat{b}}) \geq \hat{f}_I(T_{\hat{a}}^{\hat{b}})$ .

We assert that  $f_I(T_{\hat{a}}^{Mn}) < \phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{Mn}|$ . Since  $Mn \leq \hat{b}$ , it will follow that  $f_I(T_{\hat{a}}^{Mn}) < \phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{\hat{b}}|$  and thus,  $\hat{f}_I(T_{\hat{a}}^{\hat{b}}) > f_I(T_{\hat{a}}^{\hat{b}}) - \phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{\hat{b}}|$  (the desired result will hold).

Assume  $\hat{a}$  and  $Mn$  fall in the same region as determined by  $k_p, \dots, k_0$  (i.e. there exists  $p \geq h \geq 1$ , such that  $k_{(h-1)} + 1 \leq \hat{a}, Mn \leq k_h$ ). Then  $Mn = \hat{b}$ . By definition  $\hat{a}$  falls on the right end of a time window at the current time and  $\hat{b}$  falls on the left end of a time window. Thus,  $\hat{a}$  falls on the right end of a time window at time  $d_h$  (the time when  $B_{k_h}, \dots, B_{k_{(h-1)}+1}$  were dropped) and  $\hat{b}$  on the left end of a time window. Let  $T_{i_\ell}^{\hat{b}}, T_{i_{(\ell-1)}}^{i_\ell-1}, \dots, T_{i_1}^{i_2-1}$  where  $\hat{a} = i_1$  denote the time windows between  $\hat{a}$  and  $\hat{b}$  at time  $d_h$ . Note:  $N \geq d_h \geq i_\ell > i_{\ell-1} > \dots > i_1 = \hat{a}$ .

These windows were dropped so,  $f_I(T_{i_\ell}^{\hat{b}}) < \phi(d_h - i_\ell)\epsilon_f |B_{i_\ell}^{\hat{b}}|$ ,  $f_I(T_{i_{(\ell-1)}}^{i_\ell-1}) < \phi(d_h - i_{(\ell-1)})\epsilon_f |B_{i_{(\ell-1)}}^{i_\ell-1}|, \dots, f_I(T_{i_1}^{i_2-1}) < \phi(d_h - i_{i_1})\epsilon_f |B_{i_1}^{i_2-1}|$ .

Since  $\phi$  is monotonically increasing, we have  $\phi(N - \hat{a}) \geq \phi(d_h - i_1) > \phi(d_h - i_2) > \dots > \phi(d_h - i_\ell)$ . Therefore, it follows by summing the inequalities in previous paragraph that  $f_I(T_{\hat{a}}^{\hat{b}}) < \phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{\hat{b}}|$ , as desired.

Assume  $\hat{a}$  and  $Mn$  fall in different regions as determined by  $k_p, \dots, k_0$ . Let  $1 \leq z(\hat{a}) \leq p$  be the smallest integer such that  $\hat{a} \leq k_{z(\hat{a})}$  (since  $\hat{a} \leq k_p$  such an integer exists). Let  $0 \leq z(M) \leq p$  be the largest integer such that  $Mn \geq k_{z(M)}$  (since  $Mn \geq -1 = k_0$  such an integer exists). We have

$$f_I(T_{\hat{a}}^{Mn}) = f_I(T_{\hat{a}}^{k_{z(\hat{a})}}) + \sum_{i=z(\hat{a})}^{z(M)-1} f_I(T_{k_{i+1}}^{k_{(i+1)}}) + f_I(T_{k_{z(M)}+1}^{Mn}). \quad (4)$$

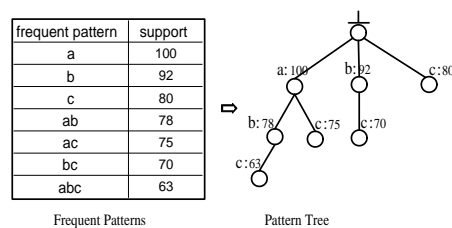
Using a similar argument as above concerning the window endpoints, the dropping condition can be applied to show  $f_I(T_{\hat{a}}^{k_{z(\hat{a})}}) < \phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{k_{z(\hat{a})}}|$  and  $f_I(T_{k_{z(M)}+1}^{\hat{b}}) < \phi(N - \hat{a})\epsilon_f |B_{k_{z(M)}+1}^{\hat{b}}|$  and  $f_I(T_{k_{i+1}}^{k_{(i+1)}})$

$< \phi(N - \hat{a})\epsilon_f |B_{k_{i+1}}^{k_{i+1}}|$  for all  $z(\hat{a}) \leq i \leq z(M) - 1$ . The assertion follows from plugging the inequalities into equation (4).  $\square$

## 5 The FP-stream Data Structure

A logarithmic tilted-time window table is used to maintain history information for a single itemset. But, our stream of transactions contains information about a potentially large number of itemsets. An efficient method of storing a large collection of itemsets and their tilted-time window tables is needed.

We propose that the collection be represented using a tree structure.<sup>5</sup> As an example, Figure 1 depicts the representation of a collection of itemsets with a single associated frequency (support). Each node in the tree represents a itemset (from root to node). The frequency is recorded at each node.



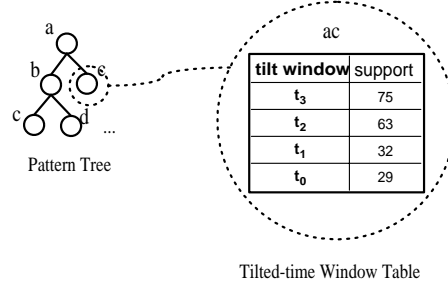
**Fig. 1** An itemset tree

Since we are interested in storing tilted-time window tables with itemsets, the structure in Figure 1 is extended. A tilted-time window table is embedded at each node. Figure 2 depicts an example of an itemset tree with tilted-time window tables embedded. This structure along with a global partition table is called an FP-stream.

### 5.1 Exploiting Anti-Monotonicity

When a new time unit  $\tau_N$  arrives, the FP-stream is updated. We saw earlier how tilted-time window tables are updated independently. Naively we could update a collection of itemsets by applying the algorithm separately for each itemset. However, anti-monotonicity of frequency can be exploited to increasing pruning. For example, given itemsets  $I' \subsetneq I$ , if  $I'$  can be removed from the FP-stream and preserve the frequency error guarantees,

<sup>5</sup> A similar approach was taken to represent a collection of itemsets in [28].



**Fig. 2** itemset tree with tilted-time window tables embedded

then so can  $I$ . The following example demonstrates, however, that the update algorithm applied independently for each itemset does not capture such anti-monotonicity pruning ( $I'$  will be dropped but  $I$  will remain in the tree).

**Example 3** Let  $u = 1$ ,  $max_1 = 2$ , and  $\phi = 1$ . Assume seven batches (all of size  $B$ ) have arrived ( $B_0$  is the oldest). The frequencies for itemsets  $I' \subsetneq I$  are listed below.

	$B_0$	$B_1$	$B_2$	$B_3$	$B_4$	$B_5$	$B_6$
$I'$	$2\epsilon_f B$	$0$	$0$	$\epsilon_f B$	$0$	$0$	$0$
$I$	$\epsilon_f B$	$0$	$0$	$\epsilon_f B$	$0$	$0$	$0$

After the first three batches, the tilted-table of  $I'$  is  $0; 2\epsilon_f B$  with last batch entry  $|B_0^1|$ . The table for  $I$  is empty as all of its entries are dropped after the third batch. After the sixth batch the tilted-table of  $I'$  is  $0, 0; \epsilon_f B, 2\epsilon_f B$  (last batch entry  $|B_0^1|$ ). The table for  $I'$  is  $0, 0; \epsilon_f B$  (last batch entry  $|B_3^3|$ ).

After the seventh batch is inserted (but before dropping) the table for  $I'$  is  $0; 0; 3\epsilon_f B$  ( $|B_0^3|$ ) and the table for  $I$  is  $0; 0; \epsilon_f B$  ( $|B_3^3|$ ). Since the last batch for  $I$  is of size  $B$ , then no entries from  $I$  are dropped. However, the last batch for  $I'$  is of size  $4B$ , hence all entries in  $I'$  are dropped.

Note that, after the seventh batch, the last entry in the table for  $I$  can be safely dropped (i.e. ensure that guarantee 3 holds) if  $f_I(T_0^3) < \epsilon_f |B_0^3|$ . However, there is not enough information in the table for  $I$  to carry out this comparison, namely,  $f_I(T_0^2)$  is not known due to an earlier dropping. Hence, the algorithm must assume that  $f_I(T_0^2)$  is as large as possible:  $\epsilon_f |B_0^2|$ ; as a result, the last entry is dropped only if  $f_I(T_3^3) < \epsilon_f |B_3^3|$  (which does not hold).

Information about  $I$  can be inferred from anti-monotonicity, because  $I' \subsetneq I$ . Since the last entry in the table of  $I'$  was dropped after the seventh batch, we know  $f_{I'}(T_0^3) < \epsilon_f |B_0^3|$ . Thus,  $f_{I'}(T_0^2) < \epsilon_f |B_0^2| + (\epsilon_f |B_3^3| - f_{I'}(T_3^3))$ . By anti-monotonicity,  $f_I(T_0^2) < \epsilon_f |B_0^2| + (\epsilon_f |B_3^3| - f_I(T_3^3))$ . We know that  $f_I(T_3^3) = \epsilon_f |B_3^3|$ ,

so  $f_I(T_0^2) < \epsilon_f |B_0^2|$ . Therefore the algorithm's assumption can be improved and, as a result, the last window of  $I$  can be safely dropped.

**Modified dropping condition:** The dropping condition can be modified to exploit anti-monotonicity.<sup>6</sup> Let  $f_I(T_{a_m}^b), \dots, f_I(T_{a_0}^b)$  denote the time windows in the table for itemset  $I$  after batch  $B_N$  has been inserted but no dropping has been carried out. If  $|I| = 1$  use the same dropping condition as before. Assume  $|I| > 1$ . First, consider the set of all  $I' \subsetneq I$  such that, after inserting  $B_N$  into the table of  $I'$  then dropping, it has  $m' + 1 < m + 1$  entries. For one such  $I'$  drop tail  $f_I(T_{a_{(m-m'-1)}}^b), \dots, f_I(T_{a_0}^b)$ ; if no such  $I'$  exists nothing is dropped (*i.e.*  $m' = m$ ).<sup>7</sup> Second, find the largest  $m \geq k \geq m - m'$  such that the dropping condition in Section 4.3 applies, drop tail  $f_I(T_{a_k}^b), \dots, f_I(T_{a_{(m-m')}}^b)$ .

Consider Example 3.  $I'$  is dropped after the seventh batch (has zero entries in its table). Thus all the entries from the table of  $I$  can be dropped.

**Theorem 5** *Suppose that a global partition structure and a corresponding tilted-time window table are maintained using the modified dropping condition above. Assume that  $N$  time units have passed. If the modified dropping condition is used, then for any  $T_a^b$ , we have  $f_I(T_a^b) \geq \hat{f}_I(T_a^b) > f_I(T_a^b) - \phi(N - \hat{a})\epsilon_f |B_a^b|$ .*

**Proof:** By induction on  $|I|$ . The base case of  $|I| = 1$  follows from Theorem 4. Now assume  $|I| \geq 2$ .

As done in the proof of Theorem 4 we model the behavior of the algorithm with pairs  $(k_p, d_p), \dots, k_1$ . As seen in that proof, it suffices to show that  $f_I(T_a^{Mn}) < \phi(N - \hat{a})\epsilon_f |B_a^{Mn}|$  where  $Mn = \min\{k_p, \hat{b}\}$ . Assume  $\hat{a}$  and  $Mn$  fall in the same region as determined by  $k_p, \dots, k_0$  (*i.e.* there exists  $p \geq h \geq 1$ , such that  $k_{(h-1)} + 1 \leq \hat{a}, Mn \leq k_h$ ). Then  $Mn = \hat{b}$ .  $\hat{a}$  falls on the right end of a time window at time  $d_h$  and  $\hat{b}$  falls on the left end. Let  $T_{i_\ell}^{\hat{b}}, T_{i_{(\ell-1)}}^{i_\ell-1}, \dots, T_{i_1}^{i_2-1}$  where  $\hat{a} = i_1$  denote the time windows between  $\hat{a}$  and  $\hat{b}$  at time  $d_h$ . If none of the  $f_I(T)$ 's above are dropped due to some  $I' \subsetneq I$ , then the proof of Theorem 4 applies unchanged to show the desired result. Assume  $f_I(T_{i_{(\ell-m'-2)}}^{i_{(\ell-m'-1)}-1}), \dots, f_I(T_{i_1}^{i_2-1})$  are dropped (at  $d_h$ ) due to some  $I'$ .

By induction (at time unit  $d_h$ ),  $\hat{f}_{I'}(T_{i_1}^{i_{(\ell-m'-1)}-1}) > f_{I'}(T_{i_1}^{i_{(\ell-m'-1)}-1}) - \phi(d_h - i_1) |B_{i_1}^{i_{(\ell-m'-1)}-1}|$ . Clearly,  $\hat{f}_{I'}(T_{i_1}^{i_{(\ell-m'-1)}-1}) = 0$ . Hence, by anti-monotonicity

$$\phi(N - i_1) |B_{i_1}^{i_{(\ell-m'-1)}-1}| > f_I(T_{i_1}^{i_{(\ell-m'-1)}-1}). \quad (5)$$

<sup>6</sup> The dropping condition becomes recursively defined.

<sup>7</sup> Later we will describe how such an  $I'$  (if exists) is chosen. However, we show next that any choice of  $I'$  preserves the frequency error guarantees.

The remaining  $f_I(T_{i_\ell}^{\hat{b}}), \dots, f_I(T_{(\ell-m'-1)}^{i_{(\ell-m')-1}})$  are dropped because of the old dropping conditions. Hence an analogous argument to that given in the proof of Theorem 4 shows  $\phi(d_h - i_{(\ell-m')} + 1)\epsilon_f |B_{i_{(\ell-m'-1)}}^{\hat{b}}| > f_I(T_{i_{(\ell-m'-1)}}^{\hat{b}})$ .

By monotonicity of  $\phi$ , the inequality in the previous paragraph, and inequality (5) we get  $\phi(N - \hat{a})\epsilon_f |B_{\hat{a}}^{\hat{b}}| > f_I(T_{\hat{a}}^{\hat{b}})$  as desired.

The case where  $\hat{a}$  and  $Mn$  do not fall in the same region is analogous.  $\square$

Theorem 5 allows any  $I' \subsetneq I$  to be used to drop part of the table for  $I$ . Ideally, we would like to choose an  $I'$  which results in the most dropping. Doing so will require checking all subsets of  $I$  (of size one less). This complicates the procedure for updating an FP-stream. To simplify matters we consider only the immediate prefix of  $I$  (with respect to an ordering of the items).

Let  $\prec$  be a total linear ordering on  $\mathcal{I}$  (items). A unique total linear ordering is induced on the collection of subsets, namely, lexicographic order. Given an itemset  $I = \{a_1, \dots, a_q\}$  where  $a_j \prec a_i$  if  $j < i$ , the immediate prefix of  $I$  is  $\{a_1, \dots, a_{q-1}\}$ . A proper prefix of  $I$  is any  $I' = \{a_1, \dots, a_p\}$  where  $1 \leq p < q$  ( $I$  is a proper postfix of  $I'$ ).

When applying the modified dropping condition to an itemset  $I$  we only consider its immediate prefix. The process of dropping parts of tilted-time window tables according to the modified dropping condition (using only immediate prefixes) is called *tail pruning*.

The next Lemma shows some facts that are useful for additional pruning and for scanning the FP-stream to produce answers to queries.

**Lemma 2** *Let  $I'$  be a proper prefix of  $I$ . Assume batch  $B_N$  has been processed using tail pruning.*

1. *If  $I'$  is not in the FP-stream, then neither is  $I$ .*
2. *Assume  $I'$  is in the FP-stream. Let  $F_1, \dots, F_{q'}$  be the frequencies for the time windows of the table for  $I'$  ( $F_1$  is the newest) and  $\hat{F}_1, \dots, \hat{F}_q$  be the frequencies in the table for  $I$ . It is the case that:  $q' \geq q$  and  $F_j \geq \hat{F}_j$  for all  $1 \leq j \leq q$ .*

**Proof:** Part 1 is proven by induction on  $N$ . In the base case,  $N = 0$ , any itemset  $J$  is in the FP-stream, if and only if  $f_J(T_0) \geq \epsilon_f |B_0|$ . Assume  $I'$  is not in. Since  $I' \subsetneq I$ , then it follows that  $f_I(T_0) < \epsilon_f |B_0|$ . Hence,  $I$  must not be in the FP-stream.

In the induction case,  $N > 0$ , assume  $I'$  is not in the FP-stream after the  $N^{th}$  batch. If  $I'$  was not in after the  $(N - 1)^{st}$  batch, then, by induction, neither was  $I$ . Hence a similar argument as in the base case shows that  $I$  cannot be in after the  $N^{th}$  batch.

Assume  $I'$  was in the FP-stream after the  $(N - 1)^{st}$  batch. Since  $I'$  was not in after the  $N^{th}$  batch, it must have been dropped while the batch was being processed. Since  $I'$  is a proper prefix of  $I$ , then there exists  $I_1, \dots, I_p$  where  $I_1 = I'$ ,  $I_p = I$  and  $I_j$  is the immediate prefix of  $I_{j+1}$  for  $1 \leq j < p$ . By the dropping condition it follows that  $I_1, \dots, I_p$  were all dropped while batch  $N$  was being processed.

Consider part 2. Since  $I'$  is a proper prefix of  $I$ , then there exists  $I_1, \dots, I_p$  as described in the previous paragraph. By the dropping condition, the table for  $I_j$  has more windows than the table for  $I_{j+1}$   $1 \leq j < p$ . Thus  $I'$  has more windows than  $I$  ( $q' \geq q$ ). By Theorem 3 frequencies  $F_1, \dots, F_{q-1}$  and  $\hat{F}_1, \dots, \hat{F}_{q-1}$  match the frequencies of  $I'$  and  $I$  (respectively) over the first  $q-1$  entries in the partition structure. These frequencies are *over the same batches of transactions*. Thus, anti-monotonicity applies to show  $F_j \geq \hat{F}_j$  for all  $1 \leq j \leq q-1$ . Let  $W_I$  denote the collection of transactions corresponding to the last batch entry for  $I$  (i.e.  $F_q$  is  $f_I(W_I)$ ). Let  $W_{I'}$  denote the collection of transactions corresponding to the last batch entry for  $I'$ . It can be seen that  $W_I \subseteq W_{I'}$ . Hence, by anti-monotonicity,  $F_q = f_I(W_I) \geq f_{I'}(W_{I'}) = \hat{F}_q$ .  $\square$

Processing the  $N^{th}$  batch consists of two steps.

1) Scan all the itemsets that appear in  $B_N$  (in lexicographic order). For each itemset,  $I$ , do the following. If  $I$  is in the FP-stream, update the table with  $f_I(B_N)$ . Apply the tail pruning and drop from  $I$ 's table as appropriate. If all of  $I$ 's table is dropped, then drop all its postfixes. We call this type of pruning *Type I pruning* and its correctness follows from Lemma 2 part 1.

If  $I$  is not in the FP-stream and  $f_I(T_N) \geq \epsilon_f |B_N|$ , then insert  $I$  (with table  $f_I(T_N)$ ). Otherwise ( $f_I(T_N) < \epsilon_f |B_N|$ ), stop scanning all postfixes of  $I$  in  $B_N$ . We call this type of pruning *Type II pruning* and its correctness follows from Lemma 2 part 1.

2) Scan the itemsets in the FP-stream in lexicographic order (equivalent to performing a depth-first search of the itemset tree). For each itemset  $I$ , apply tail pruning. If  $I$  is completely dropped from the tree, then drop all postfixes too (Type I pruning).  $\square$

In the next section, the algorithm is spelled out in detail. The lexicographic scan of the itemsets that appear in  $B_N$  is carried out by first creating an FP-tree ([21]), then performing a depth-first search.

## 6 FP-stream Updating Algorithm and Query Answering

### 6.1 FP-stream Update

The algorithm treats the first batch differently from the rest as an initialization step. Scan  $B_0$  and compute the frequency for each item and build an FP-tree. Create ordering  $\prec$  as follows. Let  $\iota_1, \dots, \iota_m$  be the items that appear in  $B_0$ ; assume  $i \leq j$  if  $f_{\iota_i}(B_0) \leq f_{\iota_j}(B_0)$  (break ties arbitrarily). Define  $\iota_i \prec \iota_j$  if  $i < j$ . For the items,  $\iota$ , not appearing in  $B_0$ , define an arbitrary ordering so long as  $\iota_m \prec \iota$ . This ordering remains fixed for all remaining batches. Finally, an FP-stream structure is created by mining all  $\epsilon_f$ -frequent itemsets from the FP-tree. The batch in memory and transaction FP-tree are discarded.

All the remaining batches  $B_i$ , for  $i \geq 1$ , are processed according to the algorithm below.

**Time-stamping** To facilitate the insertion of zeros into tilted-time window tables, each table in the FP-stream has an associated time-stamp. When the table is created the time-stamp is set to the current time unit. When an insertion is made, the time stamp is reset to the current time unit. The use of time-stamps will become clear in the algorithm description.

#### **Algorithm 1** *Incremental update of the FP-stream structure with incoming stream data*

INPUT: (1) An FP-stream structure, (2) a *min\_support* threshold,  $\sigma$ , (3) a frequency error rate,  $\epsilon_f$ , (4) an incoming batch,  $B_i$ , of transactions, (5) an item ordering  $\prec$ , and (6) an age function  $\phi$ .

OUTPUT: The updated FP-stream structure.

METHOD:

1. Initialize the FP-tree to empty.
2. Scan  $B_i$  sorting each transaction according to  $\prec$  then insert the transaction into the FP-tree (set the time-stamp to  $i$ ).
3. Update the FP-stream as follows.
  - (a) [**Step 1**] Mine itemsets out of the FP-tree using the FP-growth algorithm in [21] modified as follows (*i.e.* perform a depth-first search of the FP-tree).

For each itemset  $I$  visited during the FP-growth algorithm, if  $I$  is not in the FP-stream do the following.

- i. If  $f_I(B) \geq \epsilon_f |B|$ , insert  $I$  into the FP-stream (set the time-stamp to  $i$ ).
- ii. Otherwise, FP-growth stops mining postfixes of  $I$  in the FP-tree (Type I pruning).

If  $I$  is in the FP-stream, do the following.

- i. Let  $j$  be the time-stamp for  $I$ . Insert  $i - 1 - j$  zeros into the tilted-time window table for  $I$  according to the algorithm in Section 4.2 (updates the table with missing zeros). Add  $f_I(B)$  to the tilted-time window table and set the time-stamp to  $i$ .
  - ii. Apply tail pruning to drop windows from  $I$ 's table (using the modified dropping condition of Section 5.1).
  - iii. If  $I$ 's table becomes empty, drop  $I$  and all its postfixes from the FP-stream (Type II Pruning). FP-growth stops mining postfixes of  $I$  in the FP-tree (Type I Pruning).
  - iv. If  $I$ 's table is not empty, then FP-growth continues mining postfixes of  $I$  in the FP-tree.
- (b) [**Step 2**] Scan the FP-stream structure (depth-first search). For each itemset  $I$  encountered, let  $j$  be its time-stamp and do the following.
- i. If  $j < i$  ( $I$  was not updated when  $B_i$  was mined), then insert  $i - 1 - j$  zeros into  $I$ 's table. Set  $I$ 's time-stamp to  $i$ . Apply tail pruning to  $I$ 's table. If all of  $I$ 's windows are dropped then drop  $I$  and all its postfixes from the FP-stream (Type II pruning). The scan immediately backtracks to  $I'$  the prefix of  $I$  in the FP-stream. Since  $I'$  has already been scanned (and not dropped), continue the search with the new leftmost child of  $I'$  (the nearest postfix to  $I'$  in the FP-stream).
  - ii. If  $j = i$  ( $I$  was updated when  $B_i$  was mined), then continue to scan the FP-stream with the leftmost child of  $I$  (*i.e.* the nearest postfix of  $I$  in the FP-stream).

## 6.2 Query Answering

When the user issues a query requesting itemsets frequent over  $T_a^b$  at  $N$ , we first compute  $\hat{a}$  and  $\hat{b}$  (Section 4). Next the FP-stream is scanned and all itemsets,  $I$ , found such that  $\hat{f}_I(T_a^b) \geq \phi(N - \hat{a})(\sigma - \epsilon_f)|B_a^b|$  are returned (along with  $\hat{f}_I(T_a^b)$ ).

If granularity constants  $max_1, \dots, max_u$  are set according to Theorem 1, then guarantee 1 from Section 2.3 will be met. Theorem 5 shows that guarantee 3 holds. Note that for any itemset,  $I$ , if  $I$  is  $\sigma$ -frequent at time unit  $\tau_N$ , then guarantee shows that  $\hat{f}_I(T_a^b) > \phi(N - \hat{a})(\sigma - \epsilon_f)|B_a^b|$ . Hence  $I$  will be included in the output, so guarantee 2 holds.

A straight-forward, depth-first scan of the FP-stream is used to extract the itemsets which form the result. Lemma 2 part 2 shows that anti-monotonicity pruning can be used: when an itemset  $I'$  is found to not be included in the result, none of its postfixes are considered.

## 7 Experimental Set-up and Data Used

In this section, we describe the data and experiment set-up used to study the behavior of our algorithm. Our goal is to measure the time and space usage of the maintenance algorithm from Section 6. We do not examine query answering. This is left to future work.

### 7.1 Synthetic Data

The stream data was generated by the IBM synthetic market-basket data generator

[www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html/#assocSynData](http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html/#assocSynData)

In all the experiments, transactions were generated using 1K distinct items at 10 items per transaction on average. The default values for all other parameters of the synthetic data generator were used (*i.e.*, number of patterns 10000, average length of the maximal pattern 4, correlation coefficient between patterns 0.25, and average confidence in a rule 0.75).

The underlying statistical model used to generate the transactions does not change as the stream progresses. We feel that this does not reflect reality well. In reality, seasonal variations may cause the underlying model (or parameters of it) to shift in time. A simple-minded way to capture some of this shifting effect is to periodically, randomly permute some item names. To do this, we use an item mapping table,  $M$ . The table initially maps all item names to themselves (*i.e.*  $M(i) = i$ ). However, every five batches, 100 random permutations are applied to the table<sup>8</sup>.

### 7.2 Real Data

We used the KDD Cup 2000 “BMS-POS” dataset [33]. It consists of point-of-sales data from an electronics retailer. Items are product categories and each transaction represents all of the product categories purchased by a customer at one time. This dataset contains 515,597 transactions, 1,657 different items, and average of 6.5 items per transaction, and a maximum of 164 items per transaction.

---

<sup>8</sup> A random permutation of table entries  $i$  and  $j$  means that  $M(i)$  is swapped with  $M(j)$ . When each transaction  $\{i_1, \dots, i_k\}$  is read from input, before it is processed, it is transformed to  $\{M(i_1), \dots, M(i_k)\}$ .

### 7.3 Experimental Set-up

To simplify matters we assume all batches are of the same size: 10K transactions. We recorded the time to process each batch, the space occupied by the FP-stream and the number of itemsets maintained after each batch was processed. Our algorithm was implemented in Visual Studio C++ 6.0 with the default compiler settings and debug building model. All experiments were run on a PC with 1GB of RAM, 2.5GB of virtual memory, an Intel Pentium 4, 1.5GHz CPU, under Windows XP professional with no other users active.

Since the algorithm has many parameters, conducting an exhaustive array of experiments is very difficult. Instead, we have focused on three parameters and fixed the rest:  $\epsilon_f$ , the aging function, and the granularity constants. We used an aging function of the following form

$$\phi(x) = \begin{cases} 1 & \text{if } x = 0 \\ x^\alpha & \text{if } x = 1, 2, 3, \dots \end{cases}$$

where  $\alpha$  is a non-negative real number. Note, when  $\alpha = 0$ ,  $\phi = 1$  (no aging is applied). This aging function is very similar to the polynomial decay studied in [10].<sup>9</sup> We vary  $\alpha$  in our experiments. For the granularity constants, we use  $u = 1$  and  $max_1 \neq \infty$ . In other words, the tilted time window tables (and global partition structure) contains an unbounded number of levels, each with a maximum of  $max_1$  windows and minimum of  $max_1 - 1$  (see Section 3.1 for more details). We vary  $max_1$  in our experiments. Finally, we vary  $\epsilon_f$  in our experiments. Since  $\sigma$  does not play a role in the construction and maintenance of the FP-stream (only in query answering), it is not considered in our experiments. We leave the examination of query answering as future work.

### 7.4 Goals

The goal of our experiments is to examine the time and space behavior of our algorithm over a variety of parameter settings. More specifically our goals are to determine the following: the effective processing rate, the existence of a time and space usage upper-bound, and the time and space sensitivity to  $\epsilon_f, \alpha, max_1$ .

The effective processing rate of a batch refers to size of the batch (in our case 10000) divided by the time required to process the batch. Since the purpose of our algorithm is to process an unbounded data stream,

---

<sup>9</sup> In [10] the frequencies are multiplied by the reciprocal of  $\phi$  while we multiply the number of transactions by  $\phi$ .

The overall effect is the same.

it is essential that the algorithm not fall behind the stream (*i.e.* complete processing a batch before the next arrives). As such the effective processing rate of the stream can be measured as the maximum effective processing rate over all batches (excluding unusual outliers).

The time and space usage upper-bound refers to an upper-bound on the time and space used by the algorithm over an unbounded number of batches (if such a bound exists). Again, since our purpose is to process an unbounded stream, we cannot allow unbounded time and space consumption. Finally, the time and space sensitivity to  $\epsilon_f, \alpha, max_1$  refers to the effect of varying each of these parameters on the time and space usage of our algorithm.

## 8 Experimental Results

Figure 3 describes the presentation of our experimental results.

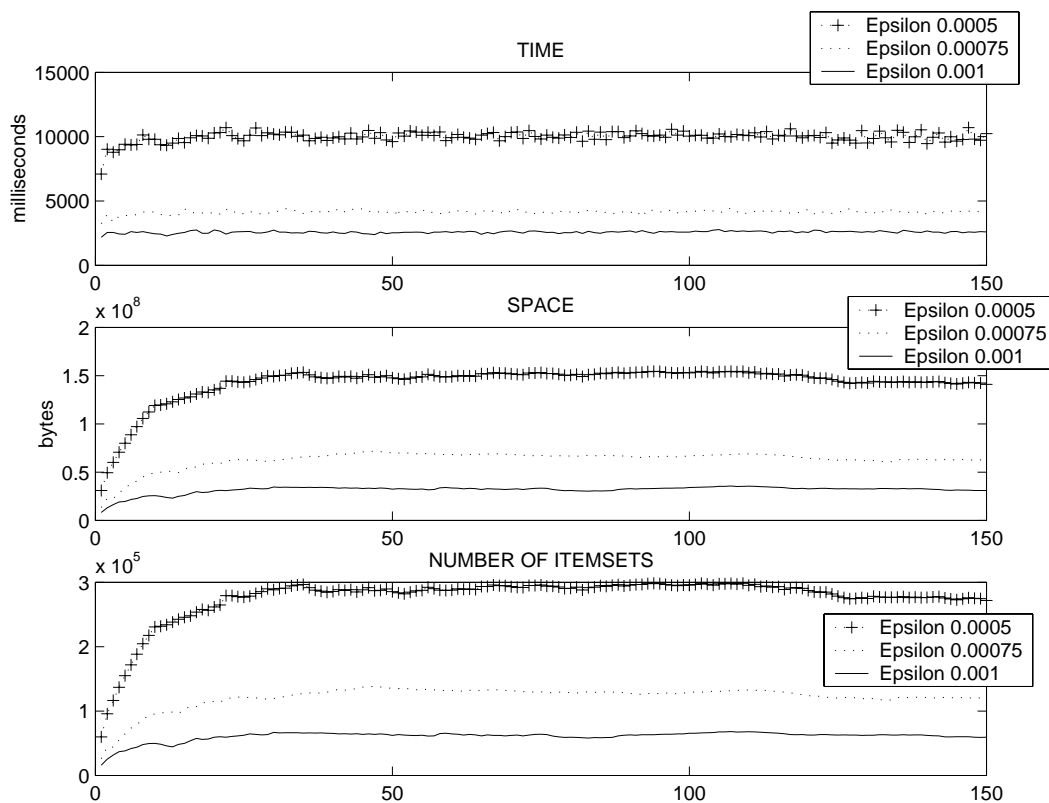
Parameters	Data set	Figure
$\epsilon_f = 0.001, 0.00075, 0.0005$ $\alpha = 0.1; max_1 = 7$	Synthetic	4
$\epsilon_f = 0.0015, 0.002$ $\alpha = 0.1; max_1 = 7$	KDD Cup (Real)	5
$\epsilon_f = 0.0025, 0.003$ $\alpha = 0.1; max_1 = 7$	KDD Cup (Real)	6
$\alpha = 0.05, 0.1, 0.2$ $\epsilon = 0.00075; max_1 = 7$	Synthetic	7
$\alpha = 0.05, 0.1, 0.2$ $\epsilon_f = 0.0025; max_1 = 7$	KDD Cup (Real)	8
$max_1 = 2, 7, 12$ $\epsilon_f = 0.00075; \alpha = 0.1;$	Synthetic	9
$max_1 = 2, 7, 12$ $\epsilon_f = 0.0025; \alpha = 0.1$	KDD Cup (Real)	10

**Fig. 3** Experimental results

All figures depict the status of the FP-stream after each batch of transactions is processed (the x-axis is labeled by batch number). For figures with three graphs, the top shows time, the middle space, and the

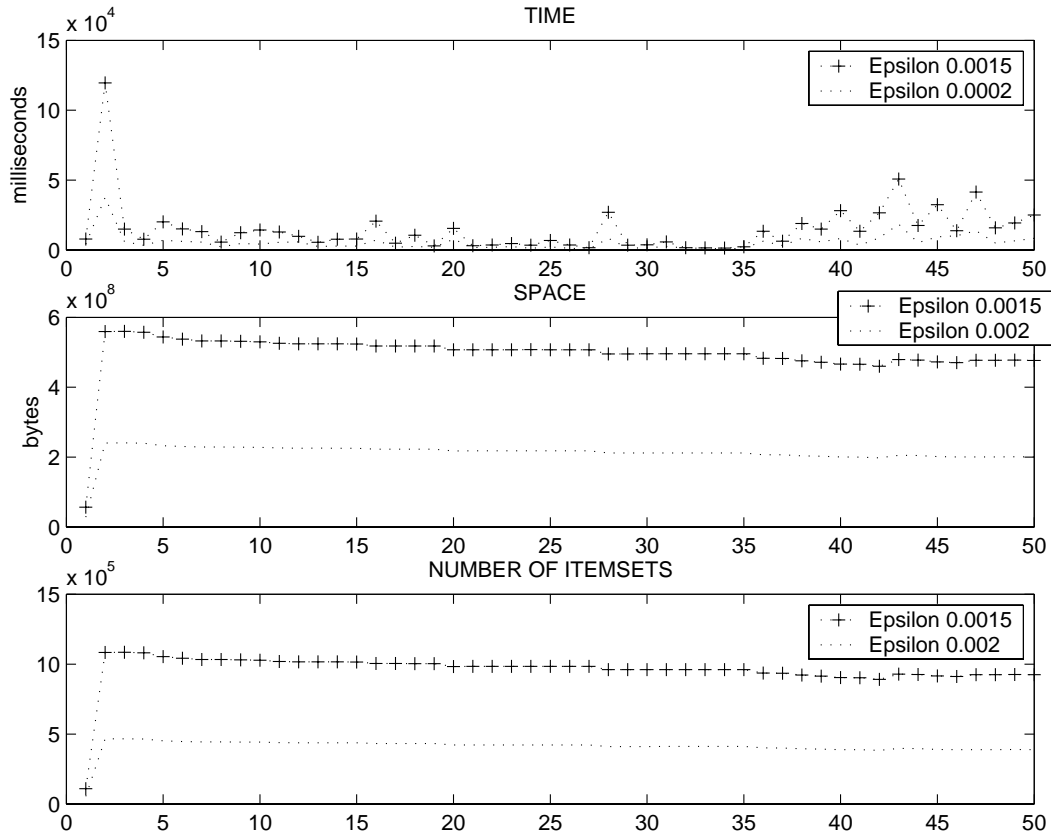
bottom the number of itemsets in the FP-stream. For figures with two graphs, the top shows time and the bottom space.

### 8.1 Varying $\epsilon_f$



**Fig. 4** Synthetic data;  $\alpha = 0.1$ ;  $max_1 = 7$

Figure 4 depicts the behavior of the algorithm on synthetic data with  $\epsilon_f$  varying. We see the time and space usage are bounded above and relatively stable. As expected, the time and space usage increase as  $\epsilon_f$  decreases. Even for  $\epsilon_f = 0.0005$ , the maximum batch processing time is relatively modest: 10 seconds. The effective processing rate is approximately 1000 transactions per second (maintaining approximately 300000 itemsets in the FP-stream). For  $\epsilon_f = 0.00075$  the effective processing rate is approximately 2500 transactions per second (maintaining approximately 110000 itemsets).



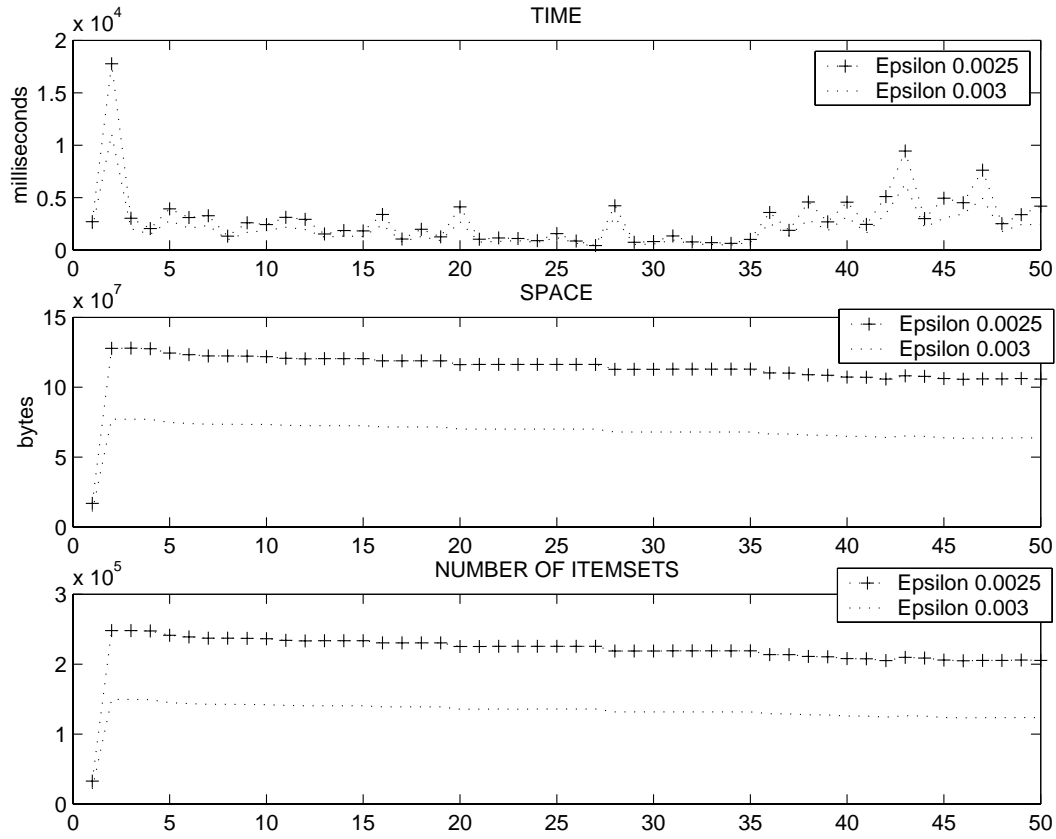
**Fig. 5** KDD Cup data;  $\alpha = 0.1$ ,  $max_1 = 7$

Figures 5, 6 depict the behavior on the KDD Cup real data set with  $\epsilon_f$  varying. The overall trends demonstrate that the time and space usage is bounded above. The space usage is quite stable, but the time usage is spiky.

Except for the large initial time spike, each batch is processed in at most 50 seconds for  $\epsilon_f = 0.0015$  (while maintaining 1 million itemsets); this amounts to an effective processing rate of 200 transactions per second.<sup>10</sup>

Recall that  $\epsilon_f$  is the error bound for itemset frequencies. We can guarantee that the frequency returned from the FP-stream for all itemsets over all time intervals is within  $(age)100\epsilon_f$  percent of the true frequency where “age” is the weight as determined by the aging function relative to the start of the time interval (the weight of all intervals starting at the current time is one and the weight increases monotonically with starting

<sup>10</sup> We also tried  $\epsilon_f = 0.001$ , but the algorithm was unable to complete. This is probably due to the vast number of itemsets maintained.



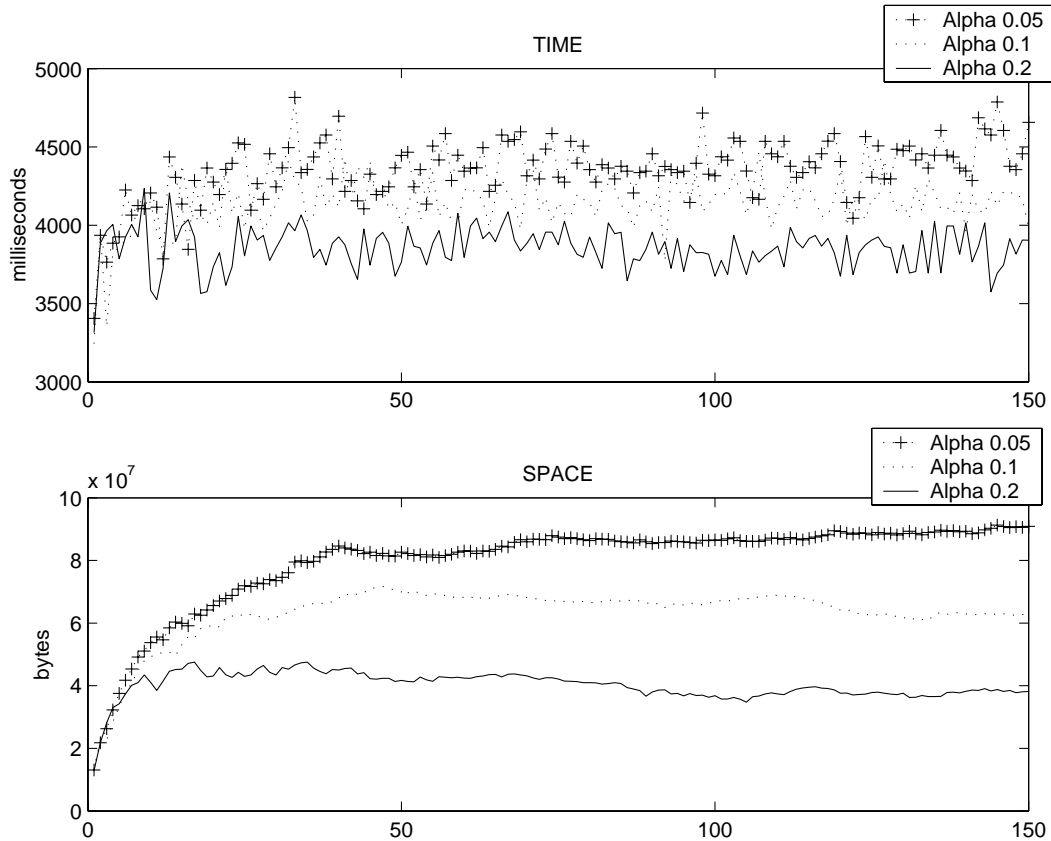
**Fig. 6** KDD Cup data;  $\alpha = 0.1$ ,  $max_1 = 7$

time). So for  $\epsilon_f = 0.0005$  as seen in figure 4, 0.005-frequent itemsets (over arbitrary intervals) can be returned with an error of (age)10 percent.

Interestingly, the space behavior on synthetic data seems to be more erratic than on real data. However, the time behavior for synthetic data seems less erratic.

## 8.2 Varying $\alpha$

Figure 7 depicts the time and space behavior on synthetic data with  $\alpha$  varying (the aging function). Since the number of itemsets demonstrates a similar behavior to space, we only present space. The space and time usage appears to be bounded from above (although jagged). As  $\alpha$  increases, the age weighting increases, hence, more dropping will occur. As expected, the space usage decreases with increasing  $\alpha$ . The time behavior appears more jagged than the previous synthetic data figures, however, this is in large part due to the difference in scales on the y-axes. Not surprising, the time usage also decreases with increasing  $\alpha$ .

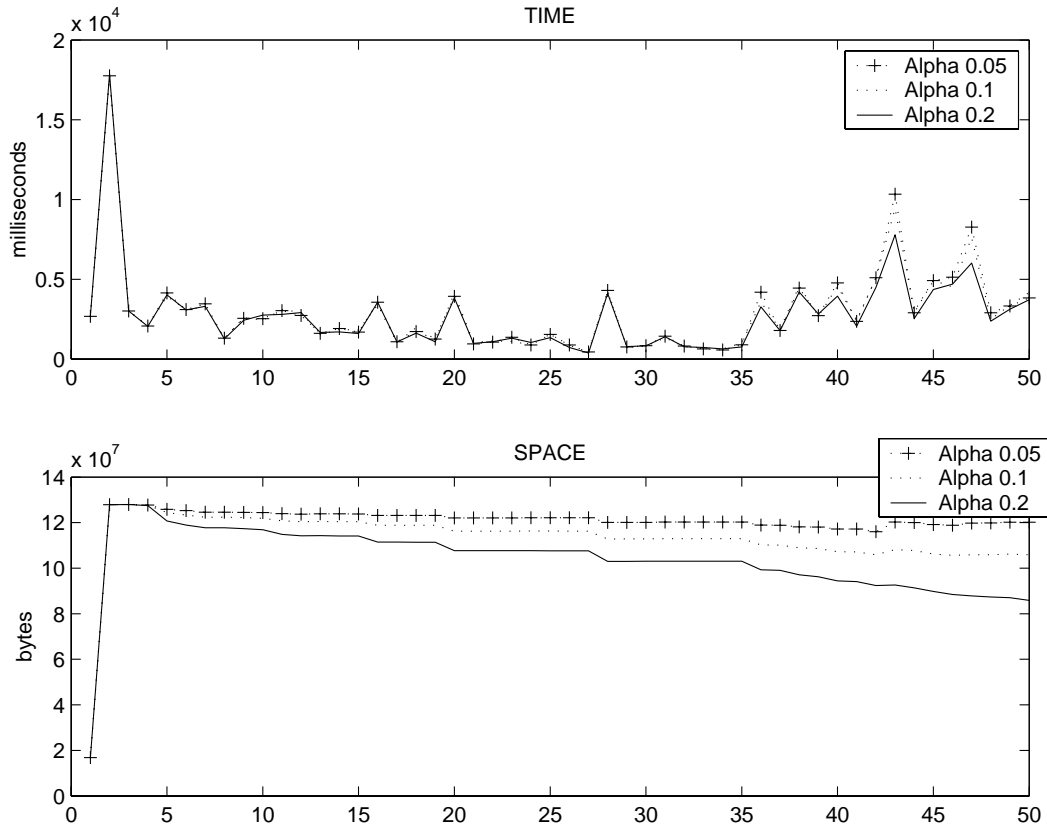


**Fig. 7** Synthetic data;  $\epsilon_f = 0.00075$ ;  $max_1 = 7$

Figure 8 depicts the time and space behavior with on the KDD Cup real data. Like earlier KDD Cup graphs, the time behavior is quite spiky with a very large initial spike. However, if one removes the two spikes at batches 43 and 47, the time and space usage does not appear to grow in an unbounded fashion. As expected, the space usage decreases with increasing  $\alpha$  (due to more dropping). The time usage also exhibits this behavior but the decrease is very slight.

### 8.3 Varying $max_1$

Figure 9 depicts the time and space behavior on synthetic data with  $max_1$  varying (the length of the tilted-time-window tables). Since the number of itemsets demonstrates a similar behavior to space, we only present space. The space and time usage appears to be bounded from above (although jagged). As  $max_1$  increases, the number of entries in the tilted-time-window tables at each granularity level also increases. Therefore, the FP-stream maintains frequency information at finer temporal granularities; a smaller temporal error can be



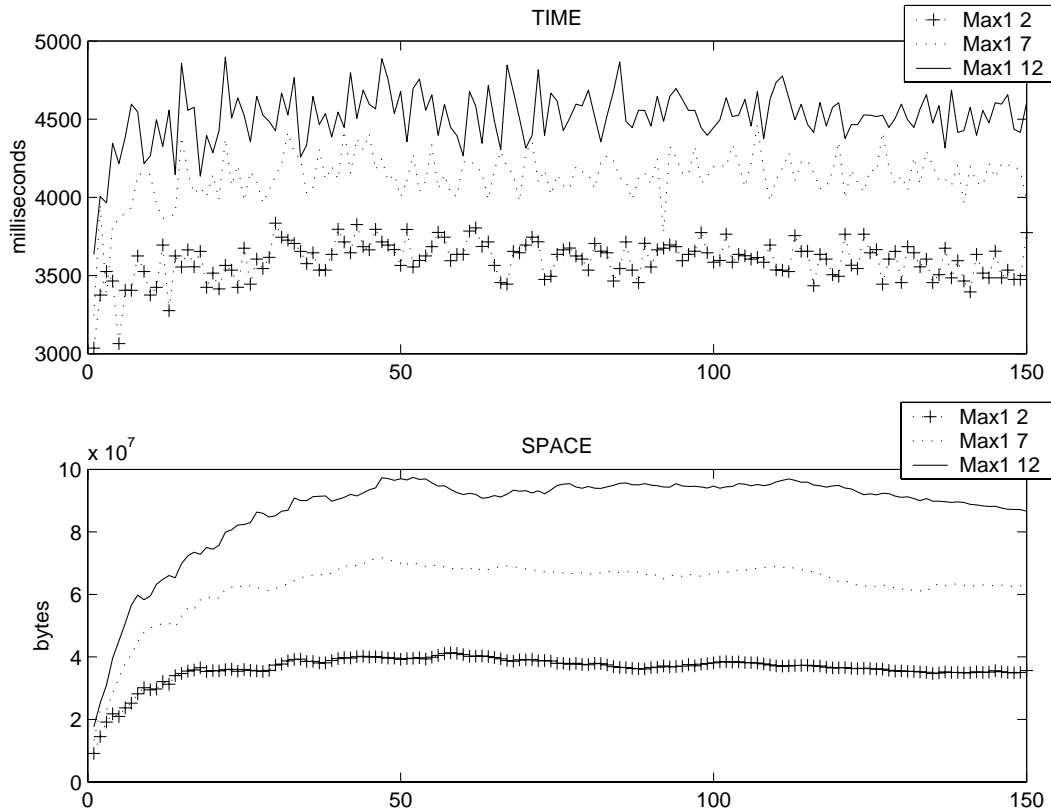
**Fig. 8** KDD Cup data;  $\epsilon_f = 0.0025$ ,  $max_1 = 7$

guaranteed. As expected, the space usage increases with increasing  $max_1$ . Not surprising, the time usage does too.

Figure 10 depicts the time and space behavior on the KDD Cup real data. As seen earlier, the time is quite spiky with a very large initial spike. But, the time and space usage does not appear to grow in an unbounded fashion (if the spikes at batches 43 and 47 are removed). As expected, the space usage decreases with decreasing  $max_1$ . The time usage also exhibits this behavior but the decrease is very small.

## 9 Previous FP-stream Work

We introduced the FP-stream data structure in [18]. However, we did not develop its maintenance and query answering in detail. In doing so, several issues emerged that required changes in the design. In addition we generalize the work of [18] in several ways.



**Fig. 9** Synthetic data;  $\epsilon_f = 0.00075$ ;  $\alpha = 0.1$

**Age weights and granularity constants:** We generalize by including an age weighting in the dropping condition. Hence older windows can be dropped more quickly if desired. Also, we generalize the definition of tilted-time window tables by including granularity constants (the user can specify the maximum number of windows that can appear at each granularity level). [18] deals only with tables allowing at most two windows per granularity level.

**Synchronized tilted-time table updates:** A global partition structure was not kept in [18]. Updating of tables for each itemset was not synchronized *i.e.* the  $i^{th}$  entry in each table did not correspond to the frequency over the same time window. As a result, choosing the approximate time endpoints (*e.g.*  $\hat{a}, \hat{b}$ ) is difficult. Synchronizing the updates with a global partition structure allows these endpoints to be computed easily.

**Temporal error:** By creating a global partition structure and synchronizing tilted-time window tables, we allow the temporal error of our answers to user’s queries to be quantified. No such temporal error was discussed in [18].

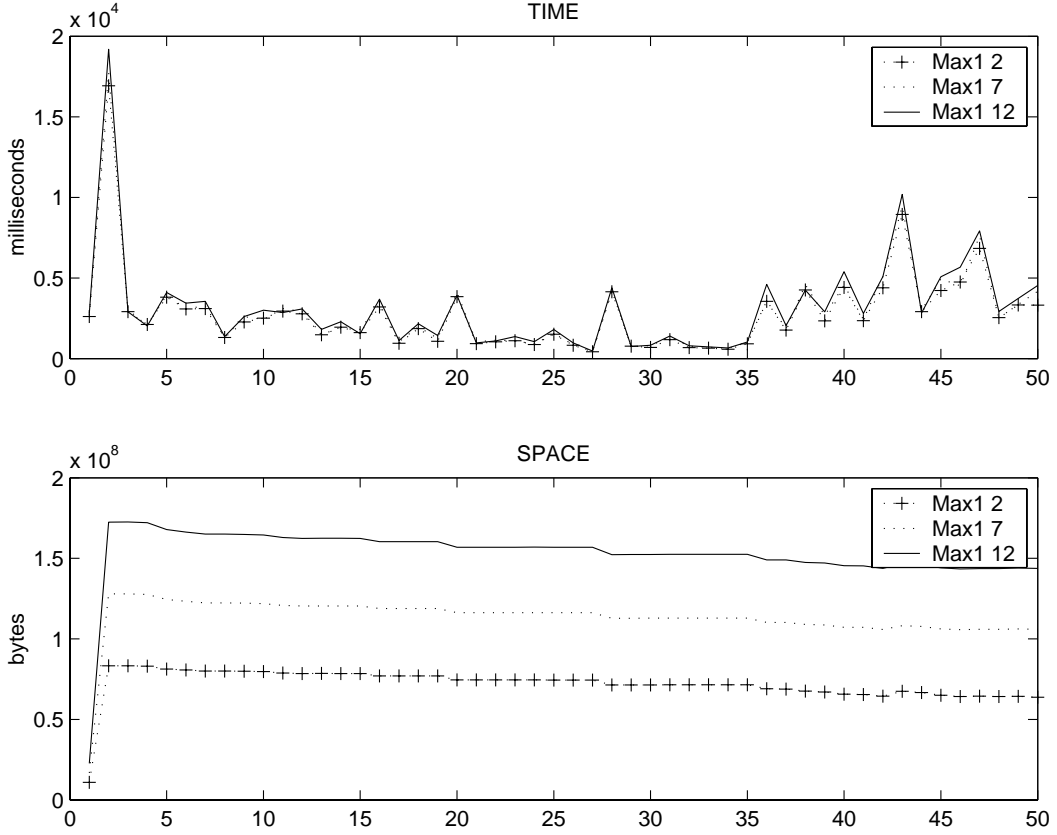


Fig. 10 KDD Cup data;  $\epsilon_f = 0.0025$ ;  $\alpha = 0.1$

**Dropping condition:** The dropping condition from [18] have been changed. There a tail  $f_I(T_{a_m}^{b_m}), \dots, f_I(T_{a_k}^{b_k}), \dots, f_I(T_{a_0}^{b_0})$  is dropped if for all  $k \geq i \geq 0$ ,  $f_I(T_{a_i}^{b_i}) < \sigma |B_{a_i}^{b_i}|$  and  $\sum_{j=i}^m f_I(T_{a_j}^{b_j}) < \epsilon_f \sum_{j=i}^m |B_{a_j}^{b_j}|$ .

The guarantee provided in [18] is that, for arbitrary queries: if  $f_I(T_a^b) \geq \sigma |B_a^b|$ , then  $\hat{f}_I(T_a^b) \geq (\sigma - \epsilon_f) |B_a^b|$ . In other words,  $\sigma$ -frequent itemsets are not missed. However, the next example shows that this guarantee does not hold for the general class of tilted-time window tables we consider (with granularity constants). Since we do not wish to miss  $\sigma$ -frequent itemsets, we modify the dropping condition of [18] to our more general setting.

**Example 4** Let the granularity constant list be  $\max_1 = 300$ . Let  $\epsilon_f = 0.1\sigma$ . Assume 301 batches have arrived all of size  $B$  with frequencies (oldest to most recent):  $\sigma B, 0.9\sigma B, 1.1\sigma B, 0, \dots, 0$ .

After the first 300 batches, using the dropping condition of [18] above, the table will be:  $0, \dots, 0, 1.1\sigma B, 0.9\sigma B, \sigma B$ . Nothing will be dropped because the oldest window is not less than  $\sigma B$ .

Upon the arrival of the next batch but before dropping the table is:  $0, \dots, 0, 1.1\sigma B; 1.9\sigma B$ . Notice that the last two batches have been merged to form a batch at the next granularity. Since  $1.9\sigma B < 2\sigma B$  and  $3\sigma B < 301\epsilon_f B = 3.1\sigma B$ , then the last window is dropped.

Now consider a query over the last three batches (last two windows)  $f_I(T_0^2)$ . We have  $f_I(T_0^2) = 3\sigma B$ , so, if the guarantee of [18] is to hold, then  $\hat{f}_I(T_0^2) > 3(\sigma - \epsilon_f)B = 2.7\sigma B$ . Since the last window was dropped,  $\hat{f}_I(T_0^2) = f_I(T_2^2) = 1.1\sigma B$ . Therefore the guarantee does not hold.

## 10 Conclusions

### 10.1 Summary

We addressed the problem of mining frequent itemsets in a data stream environment. We developed a data structure (FP-stream) allowing queries requesting itemsets frequent over arbitrary time intervals to be addressed. The answer returned is guaranteed to satisfy user-defined temporal and frequency errors. We developed an algorithm for maintaining an FP-stream and demonstrate its performance with experiments over a variety of parameter settings.

Our approach differs from previous ones most importantly in that we allow mining of itemsets over many time intervals rather than only over the entire stream. As a result our approach can discover patterns that previous approaches cannot.

### 10.2 Immediate Future Work

One of the primary directions for immediate future work is based on the following intuition. If the user does not need queries over arbitrary time intervals, rather, a more restricted class, then more can be dropped from the FP-stream. The dropping conditions could be made weaker to support a more restricted class of queries. For example, if the user were only interested in sliding window queries (queries over intervals starting at the current time to some arbitrary point in the past), the dropping conditions could be made weaker. We are currently developing a weaker dropping condition in the sliding window query case.

To conclude the paper, we touch on some broader issues.<sup>11</sup>

---

<sup>11</sup> The next subsection is a paraphrasing of the last subsection in our earlier paper [18].

### 10.3 Broader Stream Mining Issues

Dong *et al.* [16] argue that “online mining of the changes in data streams is one of the core issues” in data mining streams and that the previous work has not adequately addressed this issue. Dong *et al.* present three types of research problems: modeling and representation of changes, mining methods, and interactive exploration of changes.

Modeling and representation of changes means the development of query languages for specifying mining queries on changes in streams and the development of techniques of representing and summarizing the mined changes. Mining methods means the development of efficient algorithms for evaluating specific change mining queries. Finally, interactive exploration of changes means the development of techniques for supporting a user’s evaluation of changes. For example, initially a user may want to examine changes at a high level, then more closely study the details of interesting high-level changes.

We envision the FP-stream model as a foundation upon which frequent itemset change mining queries can be answered. For example, the change in frequency of itemsets across time intervals can be computed. Moreover, the user can first examine the frequent itemsets at a low granularity (less details), then drill-down to examine the frequent itemsets at higher granularities (more details).

**Acknowledgments** We thank Xifeng Yan, Phillip Yu, and Jian Pei for their help in the development of an early manifestation of our ideas [18].

### References

1. Adamo J.-M. *Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms*. Springer-Verlag, New York, N.Y., 2001.
2. Agrawal C., Han J., Wang J., and Yu P.S. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB)*, 2003.
3. Agrawal R. and Srikant R. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, 1994.
4. Agrawal R. and Srikant R. Mining Sequential Patterns. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.
5. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and Issues in Data Stream Systems. In *Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principals of Database Systems (PODS)*, pages 1–16, 2002.

6. Beyer K. and Ramakrishnan R. Bottom-up Computation of Sparse and Iceberg Cubes. In *Proceedings of the 1999 International Conference on Management of Data (SIGMOD)*, pages 359–370, 1999.
7. Chang J. and Lee W. Finding Recent Frequent Itemsets Adaptively over Online Data Streams. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003) – poster*, pages 226–235, 2003.
8. Charikar M., Chen K., and Farach-Colton M. Finding Frequent Items in Data Streams. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 693–703, 2002.
9. Chen Y., Dong G., Han J., Wah B., Wang J. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, pages 323–334, 2002.
10. Cohen E. and Strauss M. Maintaining Time-Decaying Stream Aggregates. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principals of Database Systems (PODS)*, pages 223–233, 2003.
11. Cormode G. and Muthukrishnan S. What’s Hot and What’s Not: Tracking Most Frequent Items Dynamically. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principals of Database Systems (PODS)*, pages 296–306, 2003.
12. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining Stream Statistics Over Sliding Windows. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.
13. Demaine E., Lopez-Ortiz A., and Munro J. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA 2002)*, 2002.
14. Dokas P., Ertöz L., Kumar V., Lazarevic A., Srivastava J., and Tan P.-N. Data Mining for Network Intrusion Detection. In *Proceedings of the 2002 National Science Foundation Workshop on Data Mining*, pages 21–30, 2002.
15. Domingos P. and Hulten G. Mining High-Speed Data Streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-00)*, pages 71–80, 2000.
16. Dong G., Han J., Lakshmanan L., Pei J., Wang H., Yu P. Online Mining of Changes from Data Streams: Research Problems and Preliminary Results. In *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*, 2003.
17. Gama J., Rocha R., Medas P. Accurate Decision Trees for Mining High-Speed Data Streams. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003) – poster*, pages 523–534, 2003.
18. Giannella C., Han J., Pei J., Yan X., Yu P. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In *Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT, H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.)*, 2003.
19. Guha S., Mishra N., Motwani R., and O’Callaghan L. Clustering Data Streams. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

20. Han J. and Kamber M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
21. Han J., Pei J., and Yin Y. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 International Conference on Management of Data (SIGMOD)*, pages 1–12, 2000.
22. Hidber C. Online Association Rule Mining. In *Proceedings of the 1999 International Conference on Management of Data (SIGMOD)*, pages 145–156, 1999.
23. Hulten G., Spencer L., and Domingos P. Mining Time-Changing Data Streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-01)*, 2001.
24. Imielinski T., Khachiyan L., and Abdulghani A. Cubegrades: Generalizing Association Rules. *Data Mining and Knowledge Discovery*, 6:219–258, 2002.
25. Jin R. and Agrawal G. Efficient Decision Tree Construction on Streaming Data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003) – poster*, pages 571–576, 2003.
26. Karp R., Shenker S., and Papadimitriou C. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55, 2003.
27. Kuramochi M. and Karypis G. Frequent Subgraph Discovery. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 313–320, 2001.
28. Manku G. and Motwani R. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, 2002.
29. O’Callaghan, Mishra N., Meyerson A., Guha S., Motwani R. High-Performance Clustering of Streams and Large Data Sets. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2002.
30. Savasere A., Omiecinski E., and Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB)*, pages 432–444, 1995.
31. Toivonen H. Sampling Large Databases for Association Rules. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, 1996.
32. Wang H., Fan W., Yu P., Han J. Mining Concept-Drifting Data Streams Using Ensemble Classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003)*, pages 226–235, 2003.
33. Zheng Z., Kohavi R., and Mason L. Real World Performance of Association Rule Algorithms. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2001)*, pages 401–406, 2001.