Mining Maximal Frequent Itemsets in Data Streams

Hua-Fu Li
Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan 300, R.O.C.
hfli@csie.nctu.edu.tw

Suh-Yin Lee
Department of Computer Science and Information Engineering, National Chiao-Tung University Hsinchu, Taiwan 300, R.O.C.
sylee@csie.nctu.edu.tw

Man-Kwan Shan
Department of Computer Science, National Chengchi University Taipei, Taiwan 116, R.O.C.
mkshan@cs.nccu.edu.tw

Abstract- Mining *streaming* data brings not only unique opportunities but also new difficult challenges of online algorithm design, such as *one streaming data scan*, *bounded memory requirement*, *fast processing time*, and *short response time*. In this paper, we propose a *single-pass* algorithm, called DSM-MFI (Data Stream Mining for Maximal Frequent Itemsets), to mine the set of all maximal frequent itemsets (MFI) in a continuous stream of transactions. In single one scan of incoming streaming data, an in-memory *summary data structure*, called IPM-Forest (Item-Prefix Maximal-itemset Forest), is developed to store all the frequent information about the maximal frequent itemsets of the data streams. In DSM-MFI, two efficient mechanisms, namely *Transaction Item-prefix Projection* (TIP) and *Top-Down Maximal frequent itemset Finding* (TDMF), is used to improve the performance of mining MFI in data streams. More specifically, TIP makes the space requirement of DSM-MFI predicable and reconstructs the smallest parts of IPM-Forest. In addition, TDMF finds all maximal frequent itemsets by a "*Max*To3" approach from the IPM-Forest generated so far. Based on our knowledge, DSM-MFI is the first algorithm for online mining maximal frequent patterns in continuous data streams.

Keywords: Data mining, data streams, maximal frequent itemsets, online algorithm, single-pass mining.

1. Introduction

Mining maximal frequent itemsets [2, 6, 7, 14, 25] is an essential step in many data mining tasks. The problem of mining maximal frequent itemsets was first proposed by Bayardo [6]. The problem can be defined as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. Let database *DB* be a set of transactions, where each transaction T is a set of items, such that $T \subseteq I$. Each transaction is associated with a unique transaction identifier, called *TID*. A set $X \subseteq I$ is also called an *itemset*. Notice that, in order to simplify the presentation, items within an itemset are kept in *lexicographic* order. A *k*-itemset is represented by $(x_1, x_2, \ldots, x_k)$, where $x_1 < x_2 < \ldots < x_k$. The *support* of an itemset X, denoted $sup(X)$, is the number of transactions in which that itemset occurs as a *subset*. An itemset X is *frequent* if its support is no less than a user-specified minimum support threshold *MinSup*; that is, $sup(X) \geq MinSup$ if X is a frequent itemset. We denoted *FI* as the set of all frequent itemsets. An itemset is *closed* if there is no superset that has the same support. Let *FCI* be the set of all frequent closed itemsets. A frequent itemset is called *maximal* if it is not a subset of any other frequent itemsets. We denoted *MFI* as the set of all maximal frequent itemsets. Thus we have MFI $\subseteq$ FCI $\subseteq$ FI.

Recently, database and data mining communities have focused on a new data model, where data arrives in the form of *continuous streams*. It is often refer to *data streams* or *streaming data*. Many applications generate large amount of data streams in real time, such as sensor data generated from sensor networks, transaction flows in retail chains, Web click and record streams in Web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, etc. Data stream mining differs from traditional data mining in two main aspects [5]:

(1) The volume of a continuous stream over its lifetime could be huge and fast changing.

(2) The queries require timely answers, and the response time is short.

Hence, it is not possible to store all the data in main memory or even in secondary storage. This motivates the design for in-memory *summary data structure* with small memory footprints that can support both one-time and continuous queries. In other words, data stream mining algorithms have to sacrifice the correctness of its analysis result by allowing some counting error. The processing model of data streams is shown in Figure 1.

Therefore, data mining technologies have been studied for traditional datasets cannot be easily solved for the data stream domain. This is because these algorithms require *multiple scans* of data which is unrealistic for streaming data. More importantly, the characteristics of the data stream can change over time and the evolving pattern needs to be captured. As a consequence, in this paper, we called an algorithm *stream-efficient* if it satisfies the following performance requirements.

First, each data record in streaming data should be examined at most once. Second, memory usage for mining data streams should be restricted finitely even though new data element are continuously generated from the data stream. Third, each data record should be processed as fast as possible. Finally, the results generated by the online algorithm should be instantly available when user requested.

With years of research into this area, several researchers have been developed for mining data streams, such as frequent itemset mining [12, 8, 13, 20, 22], frequent closed structures [19], statistics [24], data clustering [3, 15, 21], decision tree construction and data classification [1, 10, 16, 23], change detection and mining [12, 11, 17], and regression analysis [9], mining streaming Webclicks [18], etc.

In this paper, we focus on the problem of mining maximal frequent itemsets in data streams and present a space-predicable and stream-efficient algorithm DSM-MFI to mine all maximal frequent itemsets in streaming data. Moreover, DSM-MFI has two main features, namely one scan of streaming data for itemsets' support collection and an extended prefix tree-based compact pattern representation. The comprehensive experiments show that DSM-MFI is efficient on both sparse and dense datasets, and scalable to the continuous streaming data. Based on our knowledge, this problem has never been discussed in the literature.

The remainder of the paper is organized as follows. Section 2 defines the problem of mining maximal frequent itemsets in data streams. Section 3 introduces the stream-efficient algorithm DSM-MFI. In Section 4, the proposed algorithm is evaluated experimentally, and Section 5 concludes this paper.

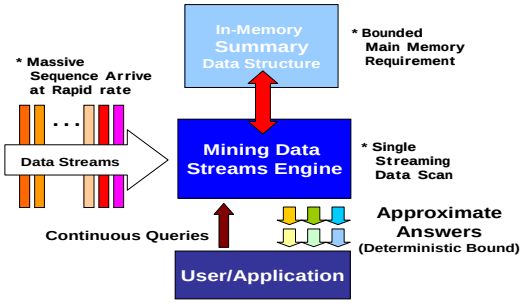

Figure 1. Processing model of data streams

2. Problem statement

Let $\Psi = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. Let *data stream* $DS = B_1, B_2, \ldots, B_N, \ldots$, be an infinite sequence of *blocks*, where each block is associated with a block identifier i, and N is the identifier of the "*latest*" block B_N. In this paper, the processing model of data stream we used is also called a *landmark model*. Each block B_i consists of a timestamp ts_i, and a set of transactions; that is, $B_i = [ts_i, T_1, T_2, \ldots, T_k]$, where $k \geq 0$. A *transaction* T is a set of items, such that $T \subseteq \Psi$. Each transaction is associated with a unique transaction identifier, called *TID*. The *current length CL* of the data stream is $|B_1|+|B_2|+\ldots+|B_N|$. A set $X \subseteq \Psi$ is also called an *itemset*, where items within an itemset are kept in lexicographic order. An itemset X with k items is denoted as $(x_1x_2\ldots x_k)$, such that $X \subseteq \Psi$. The current length of data stream with respect to itemset X, denoted as CL_X, is $|B_j|+|B_{j+1}|+\ldots+|B_N|$, where B_j is the block where the itemset X appears in, from the first, where $N \geq j \geq 1$. The *support* of an itemset X, denoted as $sup(X)$, is the number of transactions in which that itemset occurs as a subset. An itemset X is *frequent* if $sup(X) \geq MinSup*CL_X$, where $MinSup \in (0, 1)$ is a user-specified minimum support threshold. A frequent itemset is called a *maximal frequent itemset* if it is not a subset of any other frequent itemset.

Given a user-specified minimum support threshold *MinSup* and a continuous stream of transactions *DS*, the problem of mining maximal frequent itemsets in data streams is to discover the set of all maximal frequent itemsets in *one* streaming data scan.

3. The proposed algorithm

In this section, we describe the proposed method, named *DSM-MFI* (Data Stream Mining for Maximal Frequent Itemsets), for mining maximal frequent itemsets in data streams. Algorithm DSM-MFI uses two methods, namely *Transaction Item-prefix Projection* (TIP) and *Top-Down Maximal frequent itemset Finding* (TDMF), to further improve the performance of mining MFI in data streams. In Section 3.1, we illustrate the principles of algorithm DSM-MFI by mining an example data stream. Moreover, the space complexity of DSM-MFI is discussed in Section 3.2, and the guarantees of the accuracy and completeness of DSM-MFI are discussed in Section 3.3.

3.1 Mining MFI over data streams by DSM-MFI

First of all, we define the in-memory summary data structure IPM-Forest and describe the construction process of IPM-Forest. Then we use a running example to explore it.

Definition 1 An *Item-Prefix Maximal-itemset Forest* (or **IPM-Forest** for short) is an in-memory extended prefix-tree-based summary data structure defined as follows.

1. **IPM-Forest** consists of a *Dynamic Header Table* (or **DHT** for short), and a set of *item-prefix Maximal itemset trees* (or ***item-prefix*.M-trees** for short).
2. Each entry of the **DHT** consists of four fields: *item-id*, *support*, *block-id*, and *head-link*, where *item-id* registers the identifier of item, *support* records the number of transactions containing the item carrying the *item-id*, the value of *block-id* assigned to a new entry is the identifier of current block, and *head-link* points to the root node of *item-id*.M-tree. Notice that the root node of *item-id*.M-tree is *item-id*; that is, each entry *i* of DHT is an item-prefix and it is also the root node of *i*.M-tree.
3. Each node in the ***item-prefix*.M-tree** consists of four fields: *item-id*, *support*, *block-id*, and *node-link*, where *item-id* is the identifier of the inserting item, *support* registers the number of transactions represented by a portion of the path reaching the node with item-id, the value of *block-id* assigned to a new node is the identifier of current block, and *node-link* links to the next node carrying the same *item-id* in the same *item-prefix*.M-tree. If no such node, the node-link is null.
4. Each *item-prefix*.M-tree has a specific *item-prefix Dynamic Header Table* (or ***item-prefix*.DHT** for short), and four fields is associated with the *item-prefix*.DHT, namely *item-id*, *support*, *block-id*, and *head-link*. The *item-prefix*.DHT operates the same as DHT except that *node-link* links to the first node carrying the *item-id* in the *item-prefix*.M-tree. Notice that |*item-prefix*.DHT|=|DHT| in worst case, where |DHT| denotes the total number of entries in DHT.

The construction of IPM-Forest is described as follows. First of all, algorithm DSM-MFI reads a transaction T from the current block B_N. Then, DSM-MFI projects the transaction into small transactions and inserts these transactions into DHT and IPM-Forest. In details, each transaction, such as $T = (x_1x_2\ldots x_k)$, in the current block B_N is projected into the IPM-Forest by inserting k *item-prefix transactions* in it. In other words, transaction T is converted into k small transactions; that is, $(x_1x_2x_3\ldots x_k)$, $(x_2x_3\ldots x_k)$, …, $(x_{k-1}x_k)$, (x_k). These small transactions are called *item-prefix transactions*, since the first item in each small transaction is an item-prefix of original transaction T. We called this step *Transaction Item-prefix Projection* (or TIP for short), and denoted it as $TIP(T) = \{x_1|T, x_2|T, \ldots, x_k|T\}$, where $x_1|T = (x_1x_2\ldots x_k)$, $x_2|T = (x_2x_3\ldots x_k)$, …, $x_{k-1}|T = (x_{k-1}x_k)$, $x_k|T = (x_k)\}$, and T

= ($x_1x_2…x_k$). DSM-MFI drops transaction *T* after *TIP*(*T*). Next, the set of items in these item-prefix transactions are inserted into the *item-prefix*.M-trees as a branch, and updates the *item-prefix*.DHTs according to the item-prefixes. If an itemset share a prefix with an itemset already in the tree, the new itemset will share a prefix of the branch representing that itemset. In addition, a support counter is associated with each node in the tree. The counter is updated when the next item-prefix transaction causes the insertion of a new branch. After pruning all infrequent information in *item-prefix*.M-tree and *item-prefix*.DHT, IPM-Forest contains all essential information about maximal frequent itemsets of the current stream. Let's us examine an example as follows.

Example 1. Let the first block B_1 of the data stream be (*acdef*), (*abe*), (*cef*), (*acdf*), (*cef*), (*df*), the second block B_2 be (*def*), (*bef*), (*be*), and *MinSup* = 30%, where *a*, *b*, *c*, *d*, *e*, *f* are items in the stream. Hence, there are six transactions in B_1, and four transactions in B_2. Algorithm DSM-MFI mines the maximal frequent patterns by the following steps. Notice that each incoming block is processed by DSM-MFI in two steps: constructing IPM-Forest with respect to the incoming block, and pruning all infrequent information from the current IPM-Forest before processing next incoming block.

Step 1 [IPM-Forest Construction]: *Read first block into main memory for constructing the IPM-Forest*.

(a) *First transaction acdef*: First of all, DSM-MFI reads the first transaction *acdef* from B_1, and maintains the DHT. Now, the content of the DHT is [*a*:1, *c*:1, *d*:1 *e*:1, *f*:1]. Notice that we omits the field of *block-id*, since all entries in the current DHT have the same value; that is, one. Then, DSM-MFI calls the *TIP*(*acdef*). Therefore, DSM-MFI inserts these item-prefix transactions, *acdef, cdef, def, ef* and *f*, into [*a*.M-tree, *a*.DHT], [*c*.M-tree, *c*.DHT], [*d*.M-tree, *d*.DHT], [*e*.M-tree, *e*.DHT] and [*f*.M-tree, *f*.DHT], respectively. Hence, the result is shown in Figure 2. Notice that we use [*x*.M-tree, *x*.DHT] to represent the construction of current IPM-Forest with respect to item-prefix *x*. In the following steps, we omit the *head-links* of each *item-prefix*.DHT in the continuous construction process of IPM-Forest for concise representation.

(b) *Second transaction abe*: First, DSM-MFI reads the second transaction *abe* and maintains the DHT. Now, the content of the DHT is [*a*:2, *c*:1, *d*:1 *e*:2, *f*:1, b:1]. Then, DSM-MFI calls the *TIP*(*abe*). Hence, DSM-MFI inserts these item-prefix transactions *abe*, *be* and *e* into [*a*.M-tree, *a*.DHT], [*b*.M-tree, *b*.DHT] and [*e*.M-tree, *e*.DHT], respectively. Therefore, the result is shown in Figure 3.

(c) *Third transaction cef*: DSM-MFI reads the third transaction *cef* and maintains the DHT. Now, the content of the DHT is [*a*:2, *c*:2, *d*:1 *e*:3, *f*:2, b:1]. Then, it calls the *TIP*(*cef*). Afterward, DSM-MFI inserts the item-prefix transactions *cef*, *ef* and *f* into [*c*.M-tree, *c*.DHT], [*e*.M-tree, *e*.DHT] and [*f*.M-tree, *f*.DHT], respectively. The result is shown in Figure 4.

(d) *Fourth transaction acdf*: First, DSM-MFI reads the fourth transaction *acdf* and maintains the DHT. Now, the content of the DHT is [*a*:3, *c*:3, *d*:2 *e*:3, *f*:3, b:1]. Next, DSM-MFI calls the *TIP*(*acdf*). Therefore, it inserts these item-prefix transactions *acdf, cdf, df* and *f* into [*a*.M-tree, *a*.DHT], [*c*.M-tree, *c*.DHT], [*d*.M-tree, *d*.DHT] and [*f*.M-tree, *f*.DHT], respectively. The result is shown in Figure 5.

(e) *Fifth transaction cef*: At this time, DSM-MFI reads the fifth transaction *cef* and maintains the DHT. Now, the content of the DHT is [*a*:3, *c*:4, *d*:2 *e*:4, *f*:4, b:1]. Then, DSM-MFI calls the *TIP*(*cef*). Thus, it inserts the item-prefix transactions *cef*, *ef* and *f* into [*c*.M-tree, *c*.DHT], [*e*.M-tree, *e*.DHT] and [*f*.M-tree, *f*.DHT], respectively. Hence, the result is shown in Figure 6.

(f) *Sixth transaction df*: At this time, DSM-MFI reads the sixth transaction *df* and maintains the DHT. Now, the content of the DHT is [*a*:3, *c*:4, *d*:3 *e*:4, *f*:5, b:1]. Then DSM-MFI calls the *TIP*(*df*). Accordingly, DSM-MFI inserts the item-prefix transactions *df* and *f* into [*d*.M-tree, *d*.DHT] and [*f*.M-tree, *f*.DHT], respectively. Consequently, the result is shown in Figure 7.

Step 2 [IPM-Forest Reconstruction]: DSM-MFI algorithm prunes all infrequent itemsets from the *item-prefix*.M-trees and *item-prefix*.DHTs after processing the first block B_1. At this time, DSM-MFI prunes the *b*.M-tree and *b*.DHT, since item *b* is an *infrequent* item-prefix; that is, $sup(b) < MinSup*CL_b$. Then, DSM-MFI reconstructs the *a*.M-tree and *a*.DHT by eliminating the information about item *b*. The result is shown in Figure 8.

Step 3 & Step 4: *Read second block B_2 into main memory for constructing the new IPM-Forest*. The construction process of IPM-Forest with respect to block B_2 is the same as Step 1 and Step2.

The result is shown in Figure 9. Notice that the *block-id* of entry *b* in the DHT is two. This means that entry *b* is created by DSM-MFI in second block. According the whole data stream, it is not correct. The *block-id* of entry *b* is 1, since it first appeared in the block B_1. This is because of *b* is not a frequent item (or item-prefix) with respect to block B_1. Hence, if the block-id of an entry *i* in the DHT is not just 1, there are two probabilities as follows. First, it is really a new entry *i*, that is, it first appears in the streaming data seen so far. Second, it means that entry *i* is not a new entry, but it is not a frequent item with respect to the data stream seen so far.

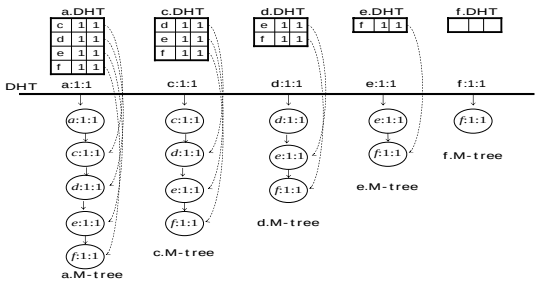

Figure 2. IPM-Forest construction after inserting first transaction *acdef*

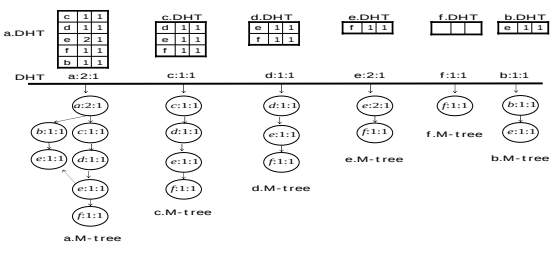

Figure 3. IPM-Forest construction after inserting second transaction *abe*

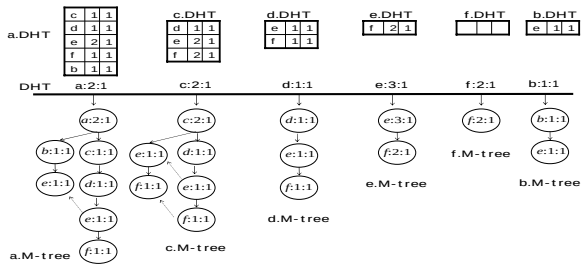

Figure 4. IPM-Forest construction after inserting third transaction *cef*

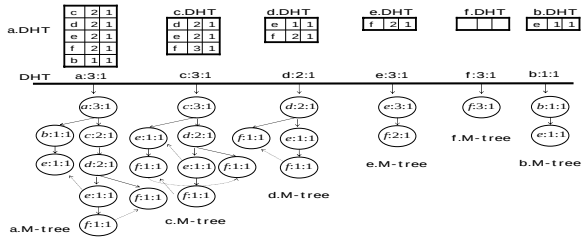

Figure 5. IPM-Forest construction after inserting fourth transaction *acdf*

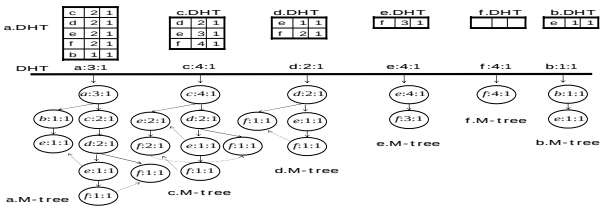

Figure 6. IPM-Forest construction after inserting fifth transaction *cef*

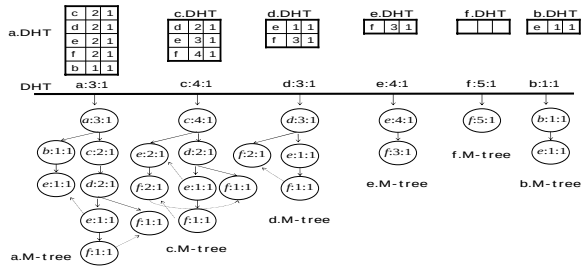

Figure 7. IPM-Forest construction after inserting sixth transaction *df*

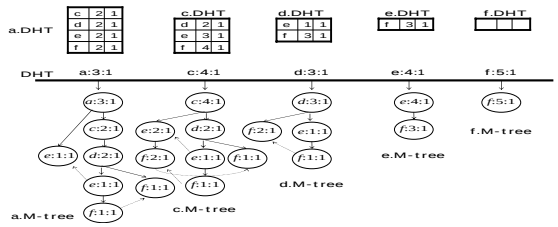

Figure 8. Current IPM-Forest after pruning any infrequent information with respect to infrequent item *b*

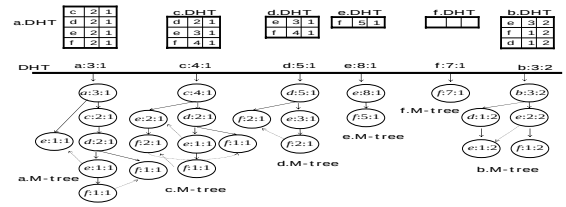

Figure 9. IPM-Forest construction after inserting the second incoming block B_2 = {*def, bef, bde, be*}

The description, as stated above, is the construction process of IPM-Forest with respect to two incoming blocks over a continuous stream. From this process, we can see that one needs exactly *one streaming data scan*. Let us, for the moment, consider the *TDMF* (Top-Down Maximal frequent itemset Finding) principle of DSM-MFI. The principle is given below.

First of all, given an entry *i* (from left to right) in the current DHT, DSM-MFI generates some MFI-candidates by a "*Max*To*3*" approach for minimal enumerating the combination of maximal frequent itemsets within the item range of *i*DHT. Then DSM-MFI checks these MFI-candidates whether they are frequent ones or not by traversing the *i*.M-tree. It also makes use of Apriori property [4]: if any length *k* pattern is not frequent in the database, its length (*k*+1) super patterns can never be frequent. The *i*.M-tree traversing principle is described as follows. First, DSM-MFI generates a candidate *Max*imal itemset, *k*+1-itemset, containing all items within the *i*.DHT (assume that |*i*.DHT| = *k*). Second, DSM-MFI traverses the *i*.M-tree via the *node-link*s of the frequent item whose support is minimal for counting this candidate. After that if the candidate is not a frequent itemset, DSM-MFI generates sub-candidates with *k* items from this *k*+1-itemset, that is, $(x_1x_2 \ldots x_{k-1}x_k)$ $(x_1x_2 \ldots x_{k-1}x_{k+1})\ldots$ $(x_2x_3 \ldots x_kx_{k+1})$. Next, DSM-MFI executes the same tree traversing for itemset's support counting, until it finds the frequent "*3*-itemsets". Why we stop the *i*.M-tree traversing principle at 3-itemsets? It is because of that the set of frequent 2-itemsets is generated by only combining the item-prefix *i* with the frequent items within the *i*.DHT.

Example 2. Let us mine the maximal frequent itemsets in the IPM-Forest in Figure 9. Suppose the minimum support threshold *MinSup* is set to 30%; that is, support threshold of frequent itemsets from B_1 to B_2 is 3, i.e., $(|B_1|+|B_2|)$*$MinSup$, but in B_2 is only 1.2, i.e., $|B_2|$*$MinSup$. Now, we start the TDMF scheme from frequent item *a* (the leftmost entry in the DHT). At this moment, the maximal frequent itemset is only 1-itemset (*a*), since the support of items, such as *c*, *d*, *e* and *f*, in the *a*.DHT are less than $MinSup*CL_a$. Now, $MinSup*CL_a$ is 3.

Next, we start the TDMF on the frequent item *c* (the second entry in DHT). DSM-MFI generates a candidate maximal frequent itemset, 3-itemset (*cef*), and traverses the *c*.M-tree for counting itemsets' support. At a result, the candidate (*cef*) is a frequent itemset, since its support is 3. Now, DSM-MFI stores the maximal frequent itemset (*cef*) in memory for further performance improvement based on Apriori property.

Furthermore, we start on the frequent item *d*, and generate a maximal candidate (*def*) based on the *d*.DHT. By traversing the *d*.M-tree using *f*'s node-link, we find that the candidate (*def*) is not a frequent itemset, since its support is 2. Hence, we directly generate two maximal frequent 2-itemsets, (*de*) and (*df*), based on the *d*.DHT without traversing *d*.M-tree again. At this moment, *four* maximal frequent itemsets, (*a*), (*cef*), (*de*) and (*df*), are generated by TDMF of DSM-MFI.

Moreover, on item *e* and *f*, since their maximal candidates are the subsets of the maximal frequent itemset *cef*, we don't start the

TDMF process. This is because of these maximal frequent itemsets already found by algorithm DSM-MFI.

Finally, we start the TDMF process on item *b*. DSM-MFI directly generates a maximal frequent itemset (*be*), since the 2-itemset is not only a candidate maximal itemset but it is also a frequent itemset. Now, *five* maximal frequent itemsets, (*a*), (*cef*), (*de*), (*df*) and (*be*), are generated by DSM-MFI.

From the mining process, as discussed above, we can see that the true support of itemset (*be*) is 4 not just 3. This is because of the *sup*(*be*) =1 was pruned before processing block B_2. Hence, algorithm DSM-MFI has the following guarantee, namely estimated support are less than the true support of frequent itemsets *X* by at most $MinSup*(CL_{X'} - CL_X)$, where $CL_{X'}$ is the actual length of data stream with respect to itemset *X*. For example, the true support of itemset (*be*) is 4, but the estimate support is 3. Therefore, the missing support with respect to itemset (*be*) is 1, which is less than $MinSup*(CL_{be'} - CL_{be}) = 3 - 1.2 = 1.8$.

3.2 Space complexity of mining MFI in data streams

Assume that there is a DHT generated by algorithm DSM-MFI. Let the number of frequent items in the DHT be *k*. Therefore, we know there are at most $C^{k}_{\lfloor k/2 \rfloor}$ maximal frequent itemsets in the data stream seen so far. If we construct an IPM-Forest for all these maximal frequent itemsets, the tree has height $\lfloor k/2 \rfloor$. In the first level, there are $C^{\lfloor k/2 \rfloor+1}_{1}$ nodes, in the second level, there are $C^{\lfloor k/2 \rfloor+2}_{2}$ nodes, in the *i*-th level, there are $C^{\lfloor k/2 \rfloor+i}_{i}$ nodes, and in the last level, the $\lfloor k/2 \rfloor$ level, there are $C^{k}_{\lfloor k/2 \rfloor}$ nodes. Thus, the total number of nodes is

$$C^{\lfloor k/2 \rfloor+1}_{1} + C^{\lfloor k/2 \rfloor+2}_{2} + C^{\lfloor k/2 \rfloor+i}_{i} + \ldots + C^{k}_{\lfloor k/2 \rfloor} = \sum_{i=1}^{\lfloor k/2 \rfloor} C^{\lfloor k/2 \rfloor+i}_{i}$$

□

The space requirement of DSM-MFI consists of three parts: the *working space* needed to create a DHT, and the *storage space* needed for the *i*.DHT and the set of *i*.M-trees, where item *i* is an entry of the DHT. In worst case, the working space for DHT requires *k* entries. For storage, there are at most $\sum_{j=1}^{K}\sum_{i=1}^{\lfloor j/2 \rfloor} C^{\lfloor j/2 \rfloor+i}_{i}$ nodes of the set of *i*.M-trees, and $(k^2-k)/2$ nodes for all *i*.DHT. Thus, the total space requirement of DSM-MFI is $\frac{1}{2}(k^2+k)+\sum_{j=1}^{K}\sum_{i=1}^{\lfloor j/2 \rfloor} C^{\lfloor j/2 \rfloor+i}_{i}$.

3.3 Accuracy and completeness guarantee

In this section, we first discuss the accuracy guarantee of the maximal frequent itemsets generated by DSM-MFI. Then, we discuss the completeness of DSM-MFI.

Let the ***t***rue support of a maximal frequent itemset *X* be ${}^{t}sup(X)$. Let the ***e***stimated support of a maximal frequent itemset *X* generated by algorithm DSM-MFI be ${}^{e}sup(X)$, where ${}^{e}sup(X)$ is the support stored in the IPM-Forest. Let *X.block-id* is the *block-id* of itemset *X* stored in the current IPM-Forest. Moreover, we assume that the average size of each block is a constant value *k* for simplify discussion, that is, each block contains *k* transactions. Let current *block-id* of the incoming stream be *block-id*(*N*). Now, we have the following theorem of *accuracy* guarantee for DSM-MFI's outputs.

Theorem 1 ${}^{t}sup(X) - {}^{e}sup(X) \le MinSup*(X.block\text{-}id - 1)*k$.

Proof: We prove by induction. Base case (*X.block* = 1): ${}^{t}sup(X) = {}^{e}sup(X)$. Thus, ${}^{t}sup(X) - {}^{e}sup(X) \le MinSup*(X.block\text{-}id - 1)*k$.

Induction step: Consider an itemset (*X*, *X.support*, *X.block-id*) that get deleted for some *block-id*(*N*)> 1. This pattern was inserted in the IPM-Forest when *block-id*(*N*+1) was being processed. The pattern *X* whose *block-id* is *block-id*(*N*+1) in the DHT could possibly have been deleted as late as the time when ${}^{e}sup(X) \le MinSup*(block\text{-}id(N+1) - X.block\text{-}id + 1)*k$. Therefore, ${}^{t}sup(X)$ of *X* when that deletion occurred was no more than $MinSup*(block\text{-}id(N+1) - X.block\text{-}id + 1)*k$. Furthermore, ${}^{e}sup(X)$ is the true support of *X* since it was inserted. It follows that ${}^{t}sup(X)$ which is the true support of *X* in first block though current block, is at most ${}^{e}sup(X) + MinSup*(block\text{-}id(N) - 1)*k$. Thus, we have ${}^{t}sup(X) - {}^{e}sup(X) \le MinSup*(X.block\text{-}id - 1)*k$.

Now, we use this theorem to describe the *completeness* of DSM-MFI. Based on Theorem 1, all maximal frequent itemsets whose support is no less than *MinSup*k* are generated by DSM-MFI. It is a superset of maximal frequent itemsets in mining streaming data, and it actually contains the set of all maximal frequent itemsets.

4. Performance evaluation

4.1 Simulation model

The parameters of synthetic data generated by IBM synthetic data generator [4] are described as follows.

IBM Synthetic Dataset: *T*10.*I*5.*D*1M and *T*30.*I*20.*D*1M. The first synthetic dataset *T*10.*I*5 has average transaction size *T* of 10 items and the average size of frequent itemset *I* is 5-items. It is a sparse dataset. In the second dataset *T*30.*I*20, the average transaction size *T* and average frequent itemset size *I* are set to 30 and 20, respectively. It is a dense dataset. Both synthetic datasets have 1,000,000 transactions. In the experiments, the synthetic data stream is broken into blocks with size 50*K* for simulating the continuous characteristic of streaming data, where 1*K* denotes 1,000. Hence, there are total 20 blocks in these experiments. Moreover, the default value of user-defined minimum support threshold *MinSup* is 0.1%.

4.2 Scalability study of DSM-MFI

In this experiment, we examine the two primary factors, *execution time* and *memory usage*, for mining maximal frequent itemsets in a data stream environment, since both should be bounded online as time advances. Therefore, in Figure 10, the execution time grows smoothly as the dataset size increases from 2,000*K* to 10,000*K*. The memory usage in Figure 11 for both synthetic datasets is stable as time progresses, indicating the scalability and feasibility of algorithm DSM-MFI. Notice that, the synthetic data stream used in Figure 11 is broken into 20 blocks with size 50*K* for simulating the continuous characteristic of data streams.

5. Conclusions

In this paper, we address the problem of mining maximal frequent itemsets in a streaming environment. A novel in-memory summary data structure called *IPM-Forest* is developed for storing essential information about maximal frequent itemsets of the stream seen so far. Moreover, we propose a single-pass algorithm *DSM-MFI* to find all maximal frequent itemsets from the IPM-Forest generated so far. Experiments with synthetic data show that DSM-MFI is efficient on both sparse and dense datasets, and

scalable to very long data streams. Based on our knowledge, DSM-MFI is the *first* algorithm which satisfies the following performance issues, such as *one streaming data scan*, *limited memory usage*, *fast processing time* for each incoming transaction and *short response time* of continuous queries, for mining maximal frequent itemsets in a continuous stream of transactions. Future work includes sliding window-based mining maximal frequent itemsets in data streams, mining sequential patterns in streaming data, and constraint-based frequent pattern mining in data streams.

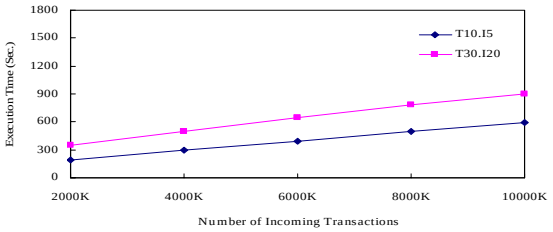

Figure 10. Required resources (execution time) of DSM-MFI for IBM synthetic datasets: *T*10.*I*5 *vs*. *T*30.*I*20

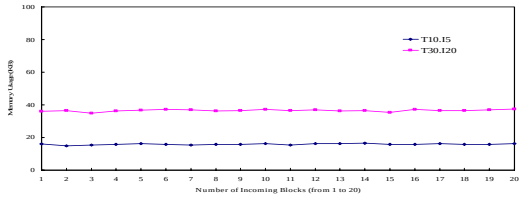

Figure 11. Required resources (memory usage) of DSM-MFI from block B_1 to block B_{20}

Acknowledgements

The authors thank the reviewers' precious comments for improving the quality of this paper. The work was supported by the National Science Council of R.O.C. under grant no. NSC93-2213-E009-043.

References

1. C.C. Aggarwal. A Framework for Diagnosing Changes in Evolving Data Streams. In *ACM SIGMOD*, 2003.
2. R. Agrawal, C. Aggarwal and V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed Computing*, 2001.
3. C. C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A Framework for Clustering Evolving Data Streams. In *Proc. of the 29th VLDB conference*, 2003.
4. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Conf. of the 20th VLDB conference*, pages 487-499, 1994.
5. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. of the 2002 ACM Symposium on Principles of Database Systems* (*PODS* 2002), ACM Press, 2002.
6. Roberto Bayardo. Efficiently Mining Long Patterns from Databases. In *ACM SIGMOD Conference*, 1998.
7. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *International Conference on Data Engineering*, Apr. 2001.
8. J. Chang and W. Lee. Finding Recent Frequent Itemsets Adaptively over Online Data Streams. In *Proc. of the 9th ACM SIGKDD International Conference & Data Mining* (*KDD-2003*), 2003.
9. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proceedings of 2002 International Conference on Very Large Data Bases* (*VLDB'02*), Hong Kong, China, Aug. 2002.11.
10. P. Domingos and G. Hulten. Mining High-Speed Data Streams. In *Proc. of the ACM Conference on Knowledge and Data Discovery* (*SIGKDD*), 2000.
11. G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang and P.S. Yu. Online Mining of Changes from Data Streams: Research Problems and Preliminary Results. In P*roceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*, June 2003.
12. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Data Streams under Block Evolution. *SIGKDD Exploration*, 3(2):1-10, Jan. 2002.
13. C. Giannella, J. Han, J. Pei, X. Yan and P. S. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In *Proc. of the NSF Workshop on Next Generation Data Mining*, 2002.
14. K. Gouda and M. Zaki. Efficiently Mining Maximal Frequent Itemsets. In *Proc. of the IEEE International Conference on Data Mining*, 2001.
15. S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *Proc. of the Annual Symp. on Foundations of Computer Science* (*FOCS*), 2000.
16. G. Hulten, L. Spencer, and P. Domingos. Mining Time-Changing Data Streams. In *Proc. of the ACM Conference on Knowledge Discovery and Data Mining* (*SIGKDD*), 2001.
17. H. F. Li and S. Y. Lee. Single-Pass Algorithms for Mining Frequency Change Patterns with Limited Space in Evolving Append-only and Dynamic Transaction Data Streams. In *IEEE International Conference on e-Technology, e-Commerce and e-Service* (*EEE-04*), Mar. 2004.
18. Hua-Fu Li, Suh-Yin Lee and Man-Kwan Shan. On Mining Web-Click Streams for Path Traversal Patterns. In *Proc. of the Thirteen International World Wide Web Conference* (*WWW-04*), New York, May 2004.
19. Hua-Fu Li, Suh-Yin Lee and Man-Kwan Shan. Mining Frequent Closed Structures in Streaming Melody Sequences. In *IEEE International Conference on Multimedia and Expo* (*ICME*-2004), June 2004.
20. G. S. Manku and R. Motwani. Approximate Frequency Counts Over Data Streams. In *Proc. of the 28th VLDB conference*, 2002.
21. L. O'Callaghan, N. Mishra, A. Meyerson, S.Guha, and R. Motwani. High-Performance Clustering of Streams and Large Data Sets. In *Proc. of the 2002 International Conference of Data Engineering* (*ICDE*), 2002.
22. W.G. Teng, M.-S. Chen, and P. S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *Proc. of the 29th VLDB Conference*, 2003.
23. H. Wang, W. Fan, P. S. Yu, and J. Han. Mining Concept-Drifting Data Streams using Ensemble Classifiers. In *ACM SIGKDD*, 2003. Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proc. 28th Int. Conf. on Very Large Data Bases*, 2002.
24. Q. Zou, W. Chu, and B. Lu. SmartMiner: A Depth First Algorithm Guided by Tail Information for Mining Maximal Frequent Itemsets. In *Proc. of the IEEE International Conference on Data Mining*, 2002.