Online Mining (Recently) Maximal Frequent Itemsets over Data Streams

Hua-Fu Li[a], Suh-Yin Lee[a], Man-Kwan Shan[b]
[a]*Department of Computer Science and Information Engineering*
National Chiao-Tung University, Hsinchu, Taiwan
hfli@csie.nctu.edu.tw, sylee@csie.nctu.edu.tw
[b]*Department of Computer Science, National Chengchi University, Taipei, Taiwan*
mkshan@cs.nccu.edu.tw

Abstract

A data stream is a massive, open-ended sequence of data elements continuously generated at a rapid rate. Mining data streams is more difficult than mining static databases because the huge, high-speed and continuous characteristics of streaming data. In this paper, we propose a new one-pass algorithm called DSM-MFI (stands for Data Stream Mining for Maximal Frequent Itemsets), which mines the set of all maximal frequent itemsets in landmark windows over data streams. A new summary data structure called summary frequent itemset forest (abbreviated as SFI-forest) is developed for incremental maintaining the essential information about maximal frequent itemsets embedded in the stream so far. Theoretical analysis and experimental studies show that the proposed algorithm is efficient and scalable for mining the set of all maximal frequent itemsets over the entire history of the data streams.

1. Introduction

In recent years, various data mining techniques have been explored in the literature. One of the most important data mining problems is mining maximal frequent itemsets from a large database [2, 6, 7, 14, 25]. The problem of mining maximal frequent itemsets was first proposed by Bayardo [6]. The problem is defined as follows. Let $\Psi = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. Let database *DB* be a set of transactions, and the *size* of *DB* is denoted by $|DB|$. A transaction *T* with *m* items is denoted by $T = \{x_1, x_2, \ldots, x_m\}$, such that $T \subseteq \Psi$. A *k*-itemset is a set with *k* items and denoted by $(x_1, x_2, \ldots, x_k)$. The *support* of an itemset *X* is the number of transactions containing *X* as a subset divided by the total number of transactions in the database, i.e., $|DB|$, and denoted by $sup(X)$. An itemset *X* is called *frequent* if $sup(X) \geq minusp$, where *minusp* is a user-defined minimum support threshold in the range of [0, 1]. The set of all frequent itemsets is denoted by *FI*. A frequent itemset is called *maximal* if it is not a subset of any other frequent itemsets. The set of all maximal frequent itemsets is denoted by *MFI*. The problem aims to find out the set of all maximal frequent itemsets with support greater than the user-defined minimum support threshold.

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 as *data streams* or *streaming data*. Many applications generate large amount of data streams in real time, such as sensor data generated from sensor networks, online transaction flows in retail chains, Web record and click-streams in Web applications, call records in telecommunications, performance measurement in network monitoring and traffic management, etc.

The problem of mining of data streams is different from mining static datasets in the following aspects [5]. First, each data element of stream should be examined at most once. Second, the memory usage in the process of mining data streams should be bounded even though new data elements are continuously generated from the streams. Third, each element in the stream should be processed as fast as possible. Fourth, the analytical outputs of the stream should be instantly available when the user requested. Finally, the errors of outputs should be constricted as small as possible. The processing model of data streams is shown in Figure 1. The continuous characteristic of streaming data makes it essential to use the algorithms which require only one scan over the stream for knowledge discovery. The huge nature of stream makes it impossible to store all the data into main memory or even in secondary storage. This motivates the design of summary data structure with small footprints that can support both one-time and continuous queries. In other words, one-

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

pass data stream mining algorithms have to sacrifice the correctness in the analytical results by allowing some counting errors. Consequently, previous *multiple-pass* algorithms studied for mining static datasets are not feasible for mining data streams.

With years of research into this research, several data stream mining problems have been discussed, such as frequent itemset mining [12, 8, 13, 20, 22], closed frequent structure mining [19], computing statistics [24], data clustering [3, 15, 21], decision tree construction and data classification [1, 10, 16, 23], change detection and mining [12, 11, 17], regression analysis [9], Web click-stream mining [18], etc.

In this paper, we will focus on the problem of mining the set of all maximal frequent itemsets in landmark windows over data streams. The proposed algorithm DSM-MFI (stands for Data Stream Mining for Maximal Frequent Itemsets) is composed of four steps. First, it reads a block of transactions from the buffer in main memory, and sorts the items of transactions in the lexicographical order. Second, it constructs and maintains the in-memory summary data structure SFI-forest (stands for Summary Frequent Itemset forest). Third, it prunes the infrequent patterns from the summary data structure. Fourth, it searches the set of all maximal frequent itemsets from the current summary data structure. Steps 1 and 2 are performed in sequence for a new incoming data block. Steps 3 and 4 are usually performed periodically or when it is needed. The theoretical analysis and experimental results show that the proposed algorithm is efficient and scalable for mining the set of all maximal frequent itemsets over the entire history of the data streams.

The remainder of this paper is organized as follows. Problem definition is given in Section 2. Algorithm DSM-MFI to find the MFI is described in Section 3. The performance results are presented in Section 4. We discuss the sliding windows in Section 5. Section 6 concludes our study.

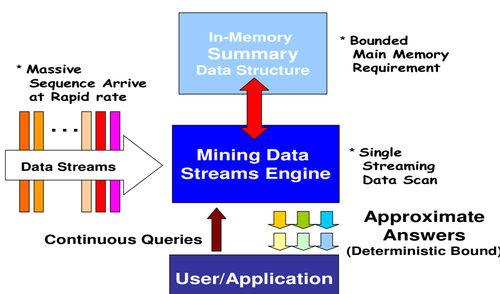

Figure 1. Processing model of data streams

2. Problem definition

Let $\Psi = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. A *data stream*, $DS = [W_1, W_2, \ldots, W_N)$, is an infinite sequence of *basic windows*, where each basic window W_i, $\forall i = 1, 2, \ldots, N$, is associated with a *window identifier i*, and N is the window identifier of the "*latest*" basic window B_N. A *basic window* consists of a fixed sized number of transactions, where each transaction is composed of a set of items (named *itemset*). The *size* of a basic window W is denoted by $|W|$. The *current length* (abbreviated as *CL*) of data stream is $|W_1| + |W_2| + \cdots + |W_N|$. A transaction T with k items is denoted by $T = (x_1, x_2, \ldots, x_k)$, such that $T \subseteq \Psi$. A k-itemset is an itemset with k items and denoted by $(x_1, x_2, \ldots, x_k)$.

Because it is unrealistic to store all the data into limited main memory or even in secondary storage, the single-pass algorithm for mining data streams has to sacrifice the correctness of their analytical results by allowing some frequency errors. Therefore, the *true support* of an itemset X is the number of transactions of the stream containing the itemset X as a subset, and denoted by $X.tsup$. The *estimated support* of an itemset X is the estimated true support stored in the summary data structure, and denoted by $X.esup$. Note that $1 \leq X.esup \leq X.tsup$. The current length of data stream with respect to an itemset X is $|W_j|+|W_{j+1}|+\cdots+|W_N|$, where basic window W_j is the first window containing X recorded in the current summary data structure, and is denoted by $X.CL$. In this paper, the itemsets embedded in the data streams can be divided into three types: *frequent itemset*, *significant itemset*, and *infrequent itemset*. An itemset X is called *frequent* if $X.esup \geq s \cdot X.CL$, where s is a user-defined minimum support threshold in the range of [0, 1]. An itemset X is called *significant* if $s \cdot X.CL > X.esup \geq \varepsilon \cdot X.CL$, where ε is a user-specified maximum support error threshold in the range of [0, s]. An itemset X is called *infrequent* if $X.esup < \varepsilon \cdot X.CL$. An itemset is called *maximal* if it is not a subset of any other frequent itemsets.

In this paper, we focus on mining the set of all maximal frequent itemsets in *landmark windows* over data streams. In the landmark model, knowledge is discovered based on the values between a specific window identifier called *landmark* and the present window identifier. In this paper, the landmark is 1, and it is an unrestricted window.

Consequently, given a data stream $DS = [W_1, W_2, \ldots, W_N)$, a minimum support threshold s in the range of [0, 1], and a maximum support error threshold

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 IEEE

ε in the range of [0, s], the problem of mining maximal frequent itemsets in landmark windows (*landmark* =1) is to find the set of all maximal frequent itemsets over the entire history of data streams.

3. The proposed algorithm: DSM-MFI

The proposed algorithm DSM-MFI is composed of four steps. First, it reads a window of transactions from the buffer in main memory, and sorts the items of transactions in a lexicographical order. Second, it constructs and maintains the in-memory summary data structure. Third, it prunes the infrequent information from the summary data structure. Fourth, it searches the maximal frequent itemsets from the current summary data structure. Steps 1 and 2 are performed in sequence for a new incoming basic window. Steps 3 and 4 are usually performed periodically or when it is needed.

3.1 Constructing the summary data structure

In this section, a new in-memory summary data structure called SFI-forest is defined, and an efficient algorithm is proposed to construct and maintain the summary data structure.

Definition 1 A *Summary Frequent Itemset forest* (abbreviated as **SFI-forest**) is an extended prefix tree-based summary data structure defined below.

1. *SFI-forest* is composed of a *frequent item list* (abbreviated as **FI-list**), and a set of *summary frequent itemset trees* (abbreviated as **SFI-trees**) of *item-suffixes*, denoted by *item-suffix*.SFI-trees.
2. Each node in the *item-suffix*.SFI-tree consists of four fields: *item-id*, *esup*, *window-id*, and *node-link*, where *item-id* is the item identifier of the inserting item, *esup* registers the number of transactions represented by a portion of the path reaching the node with the *item-id*, the value of *window-id* assigned to a new node is the window identifier of the current basic window, and *node-link* links up a node with the next node with the same *item-id* in the same SFI-tree, or null if there is none.
3. Each entry in the *FI-list* consists of four fields: *item-id*, *esup*, *window-id*, and *head of node-link* (a pointer links to the root node with *item-id* of the SFI-tree), abbreviated as *head-link*, where *item-id* registers which item identifier the entry represents, *esup* records the number of transactions containing the item with the *item-id*, the value of *window-id* assigned to a new entry is the window identifier of current basic window, the *head-link* points to the root node of the *item-id*.SFI-tree. Note that each entry with *item-id* in the FI-list is an *item-suffix* and it is also the *root node* of the *item-id*.SFI-tree.
4. Each *item-suffix*.SFI-tree has a specific opposite frequent item list (abbreviated as *OFI-list*) with respect to the item-suffix, and denoted by *item-suffix.OFI-list*. The *item-suffix*.OFI-list consists of four fields: *item-id*, *esup*, *window-id*, and *head-link*. The *item-suffix.OFI-list* operates the same as the *FI-list* except that the field head-link links to the first node carrying the *item-id* in the *item-suffix*.SFI-tree. Note that |*item-suffix.OFI-list*| = |*FI-list*| in the worst case, where |*FI-list*| denotes the total number of entries in the current FI-list.

Figure 2 outlines the SFI-forest construction of the DSM-MFI algorithm. The constriction scenario of SFI-forest is described as follows. First of all, DSM-MFI algorithm reads a transaction T from the current basic window W_N. Then, DSM-MFI projects the transaction T into many sub-transactions, and inserts these sub-transactions into the SFI-forest. The detail of projection is defined as follows. A transaction T with m items, i.e., $T = (x_1, x_2, \ldots, x_m)$, should be projected by inserting m *item-suffix sub-transactions* into the current SFI-forest. In other words, the transaction is converted into m sub-transactions; that is, $(x_1, x_2, \ldots, x_m)$, $(x_2, x_3, \ldots, x_m)$, …, and (x_m). These m sub-transactions are called *item-suffix transactions*, since the first item of each sub-transaction is an *item-suffix* of the original transaction T. The step is called *transaction projection*, and denoted by *Transaction-Projection*$(T) = \{x_1|T, x_2|T, \ldots, x_i|T, \ldots, x_m|T\}$, where $x_i|T = (x_i, x_{i+1}, \ldots, x_m)$, $\forall i = 1, 2, \ldots, m$. Note that the projection cost a transaction with m items fro constructing the summary data structure SFI-forest is $(m^2+m)/2$, i.e., $m + (m-1) + \cdots + 1$.

After performing the transaction projection of the incoming transaction T, DSM-MFI inserts the items of T into the FI-list, and removes T from the current window in the main memory. Then, the set of items of these item-suffix transactions are inserted into the *item-suffixes.SFI-forests* as branches, and updates the estimated support of corresponding *item-suffixes.OFI-lists*. If an itemset share a prefix with an itemset already in the SFI-tree, the new itemset will share a prefix of the branch representing that itemset. In addition, a counter of estimated support is associated with each node in the tree. The counter is updated when an item-suffix transaction causes the insertion of a new branch. Figure 3 shows the subroutines of construction and maintenance of SFI-forest.

1097-8585/05 $20.00 © 2005 **IEEE**

Algorithm 1 (SFI-forest construction)

Input: A data stream, DS = [W_1, W_2, …, W_N), a user-specified minimum support threshold $s \in (0, 1)$, and a user-defined maximum support error threshold $\varepsilon \in (0, s)$.
Output: A SFI-forest generated so far.

1: FI-list = {}; /*initialize the FI-list to empty.*/
2: **foreach** basic window W_j **do** /* j = 1, 2, …, N */
3: **foreach** transaction $T = (x_1, x_2, …, x_m) \in W_j$ (j = 1, 2, …, N) **do**
/* $m \geq 1$ and j is the current window identifier */
4: **foreach** item $x_i \in T$ **do** /* the maintenance of FI-list */
5: **if** $x_i \notin$ FI-list **then**
6: create a new entry of form (x_i, 1, j, *head-link*) into the FI-list; /* the entry form is (*item-id*, *item-id.esup*, *window-id*, *head-link*) */
7: **else** /* the entry already exists in the FI-list*/
8: $x_i.esup = x_i.esup + 1$;
/* increment the estimated support of item-id x_i by one*/
9: **end if**
10: **end for**
11: **call** Transaction-Projection(T, j);
/* project the transaction with each item-suffix x_i for constructing the x_i.SFI-tree */
12: **end for**
13: **call** SFI-forest-pruning(*SFI-forest*, ε, N);
/* Step 3 of DSM-MFI algorithm */
14: **end for**

Figure 2. Algorithm of SFI-forest Construction

Subroutine Transaction-Projection /* Step 2 of DSM-MFI */

Input: A transaction $T = (x_1, x_2, …, x_m)$ and the current window-id j;
Output: x_i.SFI-tree, $\forall i$ = 1, 2, …, m;

1: **foreach** item x_i, $\forall i$ = 1, 2, …, m, **do**
2: *SFI-tree-maintenance*([$x_i|X$], x_i.SFI-tree, j);
/* $X = x_1, x_2, …, x_m$ is the original incoming transaction T */
/* [$x_i|X$] is an item-suffix transaction with the item-suffix x_i*/
3: **end for**

Subroutine SFI-tree-maintenance /* Step 2 of DSM-MFI */

Input: An item-suffix transaction (x_i, x_{i+1}, …, x_m), the current window-id j, and x_i.SFI-tree, $\forall i$ =1, 2, …, m;
Output: A modified x_i.SFI-tree, $\forall i$ =1, 2, …, m;

1: **foreach** item x_l **do** /* $l = i$+1, i+2, …, m */
2: **if** $x_l \notin x_i$.OFI-list **then** /* x_i.OFI-list maintenance */
3: create a new entry of form (x_l, 1, j, *head-link*) into the x_i.OFI-list;
/*the entry form is (*item-id*, *item-id.esup*, *window-id*, *head-link*)*/
4: **else** /* the entry already exists in the x_i.OFI-list */
5: $x_l.esup = x_l.esup + 1$;
/* increment the estimated support of item-id x_l by one*/
6: **end if**
7: **endfor**
8: **foreach** item x_i, $\forall i$ = 1, 2, …, m, **do** /*x_i.SFI-tree maintenance*/
9: **if** SFI-tree has a child node with item-id y such that y.*item-id* = x_i.*item-id* **then**
10: $y.esup = y.esup$ +1;/*increment y's estimated support by 1*/
11: **else** create a new node of the form (x_i, 1, j, *node-link*);
/* initialize the estimated support of the new node to 1, and link its parent link to SFI-tree, and its node-link linked to the nodes with same item-id via the node-link structure. */
12: **end if**
13: **end for**

Subroutine SFI-forest-pruning /* Step 3 of DSM-MFI */

Input: A *SFI-forest*, a maximum support error threshold ε, and the current window identifier N;
Output: A SFI-forest which contains the set of all significant and frequent itemsets.

1: **foreach** entry x_i (i=1, 2, …, d) $\in$ FI-list, where d =|FI-list| **do**
2: **if** $x_i.esup < \varepsilon \cdot x_i.CL$ **then** /* x_i is an infrequent item */
3: delete those nodes (*item-id* = x_i) in other SFI-trees via node-link structure;
4: merge the fragmented sub-trees;
/* a simple way is to reinsert or to join the remainder sub-trees into the SFI-tree */;
5: delete x_i.SFI-tree;
6: delete x_i from other x_j.OFI-list if it exists in x_j.OFI-list (j = 1, 2, …, d; $j \neq i$);
7: delete the entry x_i from the FI-list;
8: **end if**
9: **end for**

Figure 3. Subroutines of SFI-forest construction algorithm

3.2 Pruning infrequent items from SFI-forest

According to the *Apriori* principle [4]: *if any length k pattern is not frequent, its length* (k+1) *super-patterns can never be frequent*, only the frequent 1-itemsets are used to construct candidate itemsets in the next pass. Thus, the set of itemsets stored in the current summary data structure containing the infrequent items is pruned. The pruning mechanism is usually performed periodically or when it is needed.

Let the maximum support error threshold be ε in the range of [0, s], where s is a user-specified minimum support threshold in the range of [0, 1]. The pruning method of summary data structure of DSM-MFI is that a 1-itemset X and its supersets are deleted from the current SFI-forest if $X.esup < \varepsilon \cdot X.CL$, where X is an entry of FI-list. For each entry of the form (*item-id*, *esup*, *window-id*, *head-link*) in the FI-list, if its *item-id.esup* is less than the threshold $\varepsilon \cdot$*item-id.CL*, it can be regarded as an infrequent item. At this time, three operations are performed in sequence. First, DSM-MFI deletes the *item-id*.OFI-list, *item-id*.SFI-tree, and the entry with *item-id* from the FI-list. Second, DSM-MFI removes the infrequent item with *item-id* from other OFI-lists by traversing the FI-list. Third, DSM-MFI deletes the infrequent item with item-id from other SFI-trees, and reconstructs these SFI-trees by reinserting these modified item-suffix transactions. After pruning all infrequent items from SFI-forest, SFI-forest contains the set of frequent itemsets and significant itemsets of the data stream so far.

The next step of DSM-MFI is to determine the set of all maximal frequent itemsets from the SFI-forest constructed so far. The step is performed only when the analytical results of the stream is requested.

3.3 Determining maximal frequent itemsets from the current summary data structure

Once FI-list, containing all frequent items of the stream generated so far, is constructed, DSM-MFI can derive the set of all maximal frequent itemsets by traversing the SFI-forest based on the *Apriori* principle. In this paper, an efficient mechanism called *top-down selection of Maximal Frequent Itemsets* (abbreviated as *todoMFI*) is proposed to mine the set of all maximal frequent itemsets from the current SFI-forest. It is especially useful at mining long frequent itemsets. The todoMFI algorithm is shown in Figure 4, and is described as follows.

Assume that there are k frequent 1-itemsets, such as $e_1, e_2, \ldots, e_k$, in the current FI-list, and each item e_i, $\forall i = 1, 2, \ldots, k$, has a e_i.OFI-list. Note that the size of e_i.OFI-list is denoted by $|e_i$.OFI-list$|$, and the items, namely $o_1, o_2, \ldots, o_j$, within the e_i.OFI-list are denoted by $e_i.o_1, e_i.o_2, \ldots, e_i.o_j$, respectively, where the value of j is $|e_i$.OFI-list$|$. For each entry e_i, $\forall i = 1, 2, \ldots, k$, in the FI-list, DSM-MFI first generates a candidate maximal frequent (j+1)-itemset, i.e., $(e_i, e_i.o_1, e_i.o_2, \ldots, e_i.o_j)$, by combining the item-suffix e_i and all the frequent items in e_i.OFI-list. Then, DSM-MFI uses the following traversal scheme to count its estimated support.

First, DSM-MFI starts with a specific frequent item $e_i.o_l$ $(1 \le l \le j)$, whose estimated support is smallest, and traverses the paths containing the item with $e_i.o_l$ via node-links of e_i.SFI-tree to count the estimated support of the candidate $(e_i, e_i.o_1, e_i.o_2, \ldots, e_i.o_j)$. After that, if the estimated support is greater than or equal to the threshold $s \cdot e_i.CL$, it is a maximal frequent itemset. Hence, all subsets of this maximal frequent itemset are also frequent itemsets, but not maximal frequent itemsets, according to the *Apriori* principle and the definition of maximal frequent itemset. On the other hand, if the estimated support of this candidate is less the threshold $s \cdot e_i.CL$, it is not a frequent itemset. Hence, DSM-MFI uses the same mechanism to test all the subsets of the (j+1)-itemset using a level-by-level order until the testing of candidate maximal frequent 3-itemsets. Only testing at 3-itemsets is because all frequent 2-itemsets can be generated by combining the item e_i and the frequent 1-itemsets of e_i.OFI-list. Note that a (j+1)-itemset can be decomposed into $C(j+1,j)$ j-itemsets.

Algorithm 2 (top-down selection of Maximal Frequent Itemsets)

Input: A current SFI-forest, the current window identifier N, a minimum support threshold s, and a maximum support error threshold ε.
Output: A set of all maximal frequent itemsets.

```
1:  MFI_temp-list = ∅;  /* MFI_temp-list is a temporary list used to store
                           the set of maximal frequent itemsets */
2:  foreach entry e in the current FI-list do
3:   do construct a candidate maximal frequent itemset E with size
        |E|    /* |E| = 1 + |e.OFI-list| */
4:      count E.esup by traversing the e.SFI-tree;
5:      if E.esup ≥ s·E.CL then
6:          if E ⊄ MFI_temp-list and E is not a subset of any other
               frequent itemsets in MFI_temp-list
             then
7:            add E into the MFI_temp-list;
8:            remove E's subsets from the MFI_temp-list;
9:        end if
10:     else /* if E is not a frequent itemset */
11:         enumerate E into itemsets with size |E|−1;
12:     end if
13:    until todoMFI finds the set of all maximal frequent itemsets
       with respect to entry e;
14: end for
```

Figure 4. Algorithm description of todoMFI

3.4 Theoretical analysis

In this section, we discuss the maximal estimated support error of maximal frequent itemsets generated by DSM-MFI algorithm, and the upper bound of space usage of a prefix tree-based summary data structure.

3.4.1 Maximal estimated support error analysis

Let *X.window-id* be the *window-id* of itemset X stored in the current SFI-forest, the size of basic window be k transactions, the maximum support error threshold be ε, and the current *window-id* of the data stream be *window-id*(N). Now, we have the following theorem of *maximal estimated support error guarantee* of maximal frequent itemsets generated by DSM-MFI algorithm.

Theorem 1 $X.tsup - X.esup \le \varepsilon \cdot (X.window\text{-}id - 1) \cdot k$.

Proof: We prove by induction. Base case ($X.window\text{-}id = 1$): $X.tsup = X.esup$. Thus, $X.tsup - X.esup \le \varepsilon \cdot (X.window\text{-}id - 1) \cdot k$.

Induction step: Consider an itemset of the form (X, *X.esup*, *X.window-id*) that get deleted for some *window-id*(N) > 1. The itemset was inserted in the SFI-forest when *window-id*(N+1) was being processed. The itemset X whose *X.window-id* is *window-id*(N+1) in the FI-list could possibly have been deleted as late as the time when $X.esup \le \varepsilon \cdot (window\text{-}id(N+1) - X.window\text{-}$

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 IEEE

id+1)·k. Therefore, $X.tsup$ of itemset X when that deletion occurred was no more than ε·($window\text{-}id$(N+1) − $X.window\text{-}id$+1)·k. Furthermore, the value $X.esup$ is the estimated true support of X since it was inserted. It follows that $X.tsup$ which is the true support of X in first basic window containing X though the current window, is at most $X.esup + ε·(window\text{-}id(N) − 1)·k$. Thus, we have $X.tsup − X.esup ≤ ε·(X.window\text{-}id−1)·k$. As a result, DSM-MFI generates no false negative.

3.4.2 Upper bound of space usage for constructing a prefix tree-based summary data structure

In this section, we discuss the upper bound of space usage for constructing a prefix tree-based summary data structure.

Theorem 2 A prefix tree-based summary data structure has at most 2^k nodes for storing the set of all frequent itemsets of data streams, where k is the number of frequent 1-itemsets generated so far.

Proof: Let k be the number of frequent 1-itemsets in landmark windows over data streams generated so far. Hence, the number of potential frequent itemsets is C(k, 1) regarding one item, C(k, 2) regarding two items, …, C(k, i) regarding i items, …, and C(k, k) regarding k items according to the *Apriori* heuristic. In a prefix tree-based summary data structure, an itemset is represented by a path and its support is maintained in the last node of the path. Thus, there are C(k, 1) nodes in the first level, C(k, 2) nodes in the second level, …, C(k, i) nodes in the ith level, …, and C(k, k) nodes in the kth level. There are totally C(k, 1) + C(k, 2) + ⋯ + C(k, i) + ⋯ + C(k, k) nodes in the prefix tree-based summary data structure. Consequently, the space upper bound of a prefix-tree based summary data structure is $O(2^k)$.

4. Performance evaluation

The simulation model of our experimental studies is described in Section 4.1. The required resources, i.e., memory usage and execution time, of DSM-MFI are evaluated in Section 4.2. Notice that, we do not compare DSM-MFI algorithm with other algorithms [2, 6, 7, 14, 25] in Section 4.2, since these approaches need at least *two* database scans for mining maximal frequent itemsets. It is not feasible for mining data streams.

4.1 Simulation model

All the experiments are performed on a 1GHz IBM X24 with 384MB, and the program is written in Microsoft Visual C++ 6.0. The parameters of synthetic data generated by IBM synthetic data generator [4] are described as follows.

IBM synthetic datasets: *T*10.*I*5.*D*1000K and ***T*30.*I*20.*D*1000K**. The first synthetic dataset T10.I5 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 T30.I20, 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 basic windows with size 50K for simulating the continuous characteristic of streaming data, where 1K denotes 1,000. Hence, there are total 20 windows in these experiments. Moreover, the default value of user-defined minimum support threshold s is 0.1%, and the maximum support error threshold ε is 0.1·s, i.e., 0.01%.

4.2 Scalability study of DSM-MFI

In this section, two primary factors, *execution time* and *memory usage*, are examined for mining the set of all maximal frequent itemsets in landmark windows over data streams, since both should be bounded online as time advances. Therefore, in Figure 5, the execution time grows smoothly as the dataset size increases from 2,000K to 10,000K. The memory usage in Figure 6 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 6 is broken into 20 basic windows with size 50K for simulating the continuous characteristic of data streams.

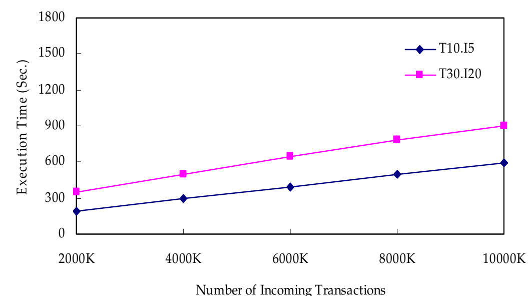

Figure 5. Required resources (execution time) of DSM-MFI for IBM synthetic datasets: T10.I5 *vs*. T30.I20

1097-8585/05 $20.00 © 2005 **IEEE**

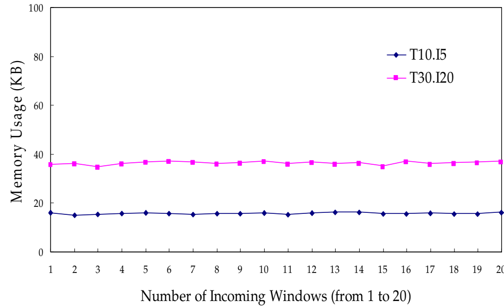

Figure 6. Required resources (Memory Usage) of DSM-MFI from window W_1 to window W_{20}

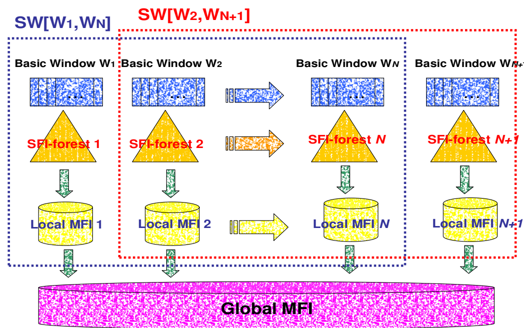

Figure 7. Mining model of DSM-RMFI

5. Discussions

In this section, we will discuss sliding window-based mining of maximal frequent itemsets over data streams.

5.1 Sliding Window Mining over Data Streams

In the sliding window model, the system stores only the N most recent basic windows, $SW[W_{i+1}, W_{i+N}]$. The main issue is that as a new basic window arrives, oldest window must be removed from the main memory and their contribution discarded from the answer.

A DSM-MFI-based algorithm called *DSM-RMFI* (stands for Data Stream Mining for Recently Maximal Frequent Itemsets) is proposed to mine the set of all maximal frequent itemsets in sliding windows over data streams. The mining model of DSM-RMFI algorithm is shown in Figure 7. For each basic window W_i, $\forall i = 1, 2, \ldots, N$, DSM-RMFI uses DSM-MFI algorithm to construct a SFI-forest i, and to find *local* maximal frequent itemsets (denoted by local MFI i) from the SFI-forest i. All local MFI i, $\forall i = 1, 2, \ldots, N$, are stored in a queue data structure. A list of global MFI, called Global MFI, is used to store the set of all local maximal frequent itemset from basic windows W_1 though W_N. DSM-RMFI only maintains a queue of local MFIs and a Global MFI into main memory. The set of all maximal frequent itemsets (global MFI) are generated by searching the Global MFI.

When a new basic window with window-id N+1 arrives, DSM-RMFI removes the local MFI 1 from the front of the queue of local MFIs and subtracts the support of the local MFI 1 from the global MFI stored in the Global MFI. After that, DSM-RMFI uses DSM-MFI algorithm to mine the set of all local maximal frequent itemsets from the new incoming basic window. Then, DSM-RMFI stores these local maximal frequent itemsets generated from the new window into local MFI N+1, and increases the support of global MFI, if the local maximal frequent itemsets already exist in the Global MFI. Otherwise, these new local maximal frequent itemsets are inserted into the Global MFI.

6. Conclusions

In this paper, we proposed a new, single-pass algorithm called DSM-MFI which mines the set of all maximal frequent itemsets over the entire history of the streaming data. In the DSM-MFI algorithm, a new in-memory summary data structure called SFI-forest is constructed for storing the frequent and significant itemsets of the streaming data generated so far. An efficient pattern selection method is developed to find the set of all maximal frequent itemsets from the current SFI-forest. Experiments with synthetic data datasets show that DSM-MFI is efficient on both sparse and dense datasets, and scalable to very long data streams. Moreover, a DSM-MFI-based sliding window mining algorithm is developed to mine the set of all maximal frequent itemsets in sliding windows over data streams.

Acknowledgements

The authors thank the reviewers' precious comments for improving the quality of the paper. The research is supported by National Science Council of R.O.C. under grant no. NSC93-2213-E-009-043.

7. 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] H.-F. Li, S.-Y. Lee and M.-K. 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] H.-F. Li, S.-Y. Lee and M.-K. 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.

1097-8585/05 $20.00 © 2005 **IEEE**