

# 行政院國家科學委員會專題研究計畫 期中進度報告

## 資料串流管理系統及串流資料探勘之研發(2/3)

計畫類別：個別型計畫

計畫編號：NSC93-2213-E-009-043-

執行期間：93 年 08 月 01 日至 94 年 07 月 31 日

執行單位：國立交通大學資訊工程學系(所)

計畫主持人：李素瑛

計畫參與人員：李華富 郭芳菲 何錦泉

報告類型：精簡報告

報告附件：出席國際會議研究心得報告及發表論文

處理方式：本計畫可公開查詢

中 華 民 國 94 年 10 月 30 日

## 1. 中文摘要

資料串流是一種資料量大、快速且無止境的資料序列。由於這些資料串流的特殊特性，探勘資料串流的難度是遠高於傳統靜態資料庫的探勘。在本計劃中，我們提出一個新的單次掃描演算法稱為 DSM-MFI。DSM-MFI 演算法可以在資料串流的標的模型中探勘出最大頻繁項目集合。在 DSM-MFI 演算法中，我們設計出一個新的摘要資料結構，稱為 SFI-forest。DSM-MFI 演算法可以將隱藏在資料串流中的候選最大項目集合擷取出來，儲存在 SFI-forest 中。最後再使用一個由上而下的最大頻繁項目集合檢查機制，稱為 todoMFI，將目前儲存在 SFI-forest 中的最大頻繁項目集合快速輸出。

## 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 project, 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.

**Keywords:** Data Streams, Data Mining, Single-Pass Algorithm, Maximal Frequent Itemsets

## 2. Introduction and Related Work

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  $I = \{i_1, i_2, \dots, i_n\}$  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  $T = \{x_1, x_2, \dots, x_m\}$ , such that  $T \subseteq I$ . A k-itemset is a set with k items and denoted by  $(x_1, x_2, \dots, x_k)$ . 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  $\text{sup}(X)$ . An itemset X is called frequent if  $\text{sup}(X) \geq \text{minsup}$ , where minsup 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 referred 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 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-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.

### 3. Problem Definition

Let  $\Psi = \{i_1, i_2, \dots, i_n\}$  be a set of literals, called *items*. A *data stream*,  $DS = [W_1, W_2, \dots, W_N]$ , is an infinite sequence of *basic windows*, where each basic window  $W_i$ ,  $\forall i = 1, 2, \dots, N$ , 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  $|W_1| + |W_2| + \dots + |W_N|$ . A transaction  $T$  with  $k$  items is denoted by  $T = (x_1, x_2, \dots, x_k)$ , such that  $T \subseteq \Psi$ . A  $k$ -itemset is an itemset with  $k$  items and denoted by  $(x_1, x_2, \dots, x_k)$ .

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  $|W_j| + |W_{j+1}| + \dots + |W_N|$ , 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  $\varepsilon$  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 project, 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  $DS = [W_1, W_2, \dots, W_N]$ , a minimum support threshold  $s$  in the range of  $[0, 1]$ , and a maximum support error threshold  $\epsilon$  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.

#### 4. The Proposed Method: 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.

##### Algorithm 1 (SFI-forest construction)

**Input:** A data stream,  $DS = [W_1, W_2, \dots, W_N]$ , a user-specified minimum support threshold  $s \in (0, 1)$ , and a user-defined maximum support error threshold  $\epsilon \in (0, s)$ .

**Output:** A SFI-forest generated so far.

```

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

```

**Figure 1. Algorithm of SFI-forest Construction**

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

**Input:** A transaction  $T = (x_1, x_2, \dots, x_m)$  and the current window-id  $j$ ;

**Output:**  $x_i$ -SFI-tree,  $\forall i = 1, 2, \dots, m$ ;

```

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

```

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

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

**Output:** A modified  $x_i$ .SFI-tree,  $\forall i = 1, 2, \dots, m$ ;

```

1:  foreach item  $x_l$  do /*  $l = i+1, i+2, \dots, m$  */
2:    if  $x_l \notin x_i$ .OFI-list then /*  $x_l$ .OFI-list maintenance */
3:      create a new entry of form  $(x_l, 1, j, \text{head-link})$  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  $x_i$ ,  $\forall i = 1, 2, \dots, m$ , 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  $(x_i, 1, j, \text{node-link})$ ;
/* 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  $\epsilon$ , and the current window identifier  $N$ ;

**Output:** A SFI-forest which contains the set of all significant and frequent itemsets.

```

1:  foreach entry  $x_i$  ( $i=1, 2, \dots, d$ )  $\in$  FI-list, where  $d = |\text{FI-list}|$  do
2:    if  $x_i$ .esup <  $\epsilon \cdot x_i$ .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 ( $j = 1, 2, \dots, d; j \neq i$ );
7:      delete the entry  $x_i$  from the FI-list;
8:    end if
9:  end for

```

**Figure 2. Subroutines of SFI-forest construction algorithm**

### 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  $\epsilon$ .

**Output:** A set of all maximal frequent itemsets.

```

1:  MFItemp-list =  $\emptyset$ ; /* MFItemp-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  $\geq s \cdot E$ .CL then
6:      if  $E \notin \text{MFI}_{\text{temp}}\text{-list}$  and  $E$  is not a subset of any other frequent itemsets in MFItemp-list
7:        then
8:          add  $E$  into the MFItemp-list;
9:          remove  $E$ 's subsets from the MFItemp-list;
10:       end if
11:     else /* if  $E$  is not a frequent itemset */
12:       enumerate  $E$  into itemsets with size  $|E|-1$ ;
13:     end if
14:   until todoMFI finds the set of all maximal frequent itemsets with respect to entry  $e$ ;
15: end for

```

**Figure 3. Algorithm description of todoMFI**

## 5. Conclusions

In this project, 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.

## 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 Proceedings 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.