In IEEE SMC 2006

Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window

Hua-Fu Li, Chin-Chuan Ho, Man-Kwan Shan, and Suh-Yin Lee

Abstract— **Online mining of streaming data is one of the most important issues in data mining. In this paper, we proposed an efficient one-pass algorithm, called MFI-TransSW (Mining Frequent Itemsets over a Transaction-sensitive Sliding Window), to mine the set of all frequent itemsets in data streams with a transaction-sensitive sliding window. An effective bit-sequence representation of items is used in the proposed algorithm to reduce the time and memory needed to slide the windows. The experiments show that the proposed algorithm not only attain highly accurate mining results, but also run significant faster and consume less memory than existing algorithms for mining frequent itemsets over recent data streams.**

I. Introduction

A data stream is an ordered sequence of elements that arrives in timely order. Different from data in traditional static datasets, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [10]. It is often refer to as *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.

Data streams can be further classified into *offline* data streams and *online* data streams. Offline data streams are characterized by regular bulk arrivals [15], such as a bulk addition of new transactions as in a data warehouse system. Online data streams are characterized by real-time updated data that come one by one in time, such as an a continuously generated transaction as in a network monitoring system. Bulk data processing is not possible for one streaming data.

Due to the characteristics of data streams, there are some inherent challenges for mining streaming data [3, 8, 10]. 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 a 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* data mining algorithms studied for static datasets are not feasible for mining data streams.

Many previous studies contributed to the efficient mining of frequent itemsets (FI) in streaming data [4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19]. According to the stream processing model [20], the research of mining frequent itemsets in data streams can be divided into three categories: *landmark windows* [15, 12, 19, 11, 13], *sliding windows* [5, 6, 14, 16, 17, 18], and *damped windows* [7, 4], as described briefly as follows. In the landmark window model, knowledge discovery is performed based on the values between a specific timestamp called landmark and the present. In the sliding window model, knowledge discovery is performed over a fixed number of recently generated data elements which is the target of data mining. Two types of sliding widow, i.e., *transaction- sensitive sliding window* and *time-sensitive sliding window*, are used in mining data streams. The basic processing unit of window sliding of first type is an expired transaction while the basic unit of window sliding of second one is a time unit, such as a minute or a hour. In the damped window model, recent sliding windows are more important than previous ones.

In [15], Manku and Motwani developed two single-pass algorithms, Sticky-Sampling and Lossy Counting, to mine frequent items over *offline* data streams with a landmark window. Moreover, Manku and Motwani proposed a Lossy-Counting based three module method BTS (Buffer-Trie-SetGen) for mining the set of frequent itemsets from *offline* data streams. Li et al. proposed prefix tree-based single-pass algorithms, called DSM-FI [12] and DSM-MFI [13], to mine the set of all frequent itemsets and maximal frequent itemsets over the entire history of *offline* data streams.

This work was supported in part by the National Science Council under Grant NSC 94-2213-E-009-012.

Hua-Fu Li is with the Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan 300, R.O.C. (corresponding author to provide phone: 886-3-5717172#56601; fax: 886-3-5721490; e-mail: hfli@csie.nctu.edu.tw).

Chin-Chuan Ho is with the Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: hocc@csie.nctu.edu.tw).

Man-Kwan Shan is with Department of Computer Science, National Chengchi University, Taipei, Taiwan 116, R.O.C. (e-mail: mkshan@cs.nccu.edu.tw).

Suh-Yin Lee is with the Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: sylee@csie.nctu.edu.tw).

Jin and Agrawal [11] proposed an algorithm, called StreamMining, for in-core frequent itemset mining over *online* data streams. Yu et al. [19] discussed the issues of false negative or false positive in mining frequent itemsets from high speed *offline* transactional data streams.

Chang and Lee [5] proposed a BTS-based algorithm, called SWFI-stream, for mining frequent itemsets in *online* data streams with a *transaction-sensitive* sliding windows model. Li et al. [13] proposed a DSM-MFI-based algorithm called DSM-RMFI to find maximal frequent itemsets over *offline* data streams with a *transaction-sensitive* sliding window. Teng et al. [16] proposed a regression-based algorithm, called FTP-DS, to find temporal patterns (frequent inter-transaction itemsets) across *multiple online* data streams in a time-sensitive sliding window. Teng et al. [17] proposed a resource-aware algorithm called RAM-DS, to mine temporal patterns in *multiple online* data streams with a time-sensitive sliding window. Lin et al. [14] proposed an incremental mining algorithm to find the set of frequent itemsets in *offline* data streams with a *time-sensitive* sliding window. Wong and Fu [18] proposed an efficient algorithm to mine *top-k frequent itemsets* in offline data streams with a *transaction-sensitive* sliding window without a user-defined minimum support constraint. Chi et al. [6] proposed a *transaction-sensitive* sliding window based algorithm, called MOMENT, which might be the first to find *frequent closed itemsets* (FCI) from *online* data streams with a transaction- sensitive sliding window. A summary data structure, called CET, is used in the MOMENT algorithm to maintain the information of closed frequent itemsets.

Chang and Lee [4] developed a damped window based algorithm, called estDec, for mining frequent itemsets in *online* streaming data in which each transaction has a weight decreasing with age. In other words, older transactions contribute less toward itemset frequencies, and it is a kind of damped windows model. Giannella et al. [7] proposed a frequent pattern tree (abbreviated as FP-tree) [9] based algorithm, called FP-stream, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique. FP-stream focuses on *offline* data streams.

The purpose of this paper is on *frequent itemsets* mining over *online* data streams with a *transaction-sensitive* sliding window, we mainly address it by comparison with the SWFI-stream algorithm proposed by Chang and Lee [5]. An efficient algorithm, called MFI-TransSW (Mining Frequent Itemsets over Transaction-sensitive Sliding Windows), is proposed to mine frequent itemsets over *online* data streams with a *transaction-sensitive* sliding window. The experiments show that the MFI-TransSW algorithm not only attain highly accurate mining results, but also run significant faster and consume less memory than SWFI-stream algorithm for mining frequent itemsets over recent data streams.

The remainder of this paper is organized as follows. The problem of FI mining in a transaction-sensitive sliding window is defined and the algorithm MFI-TransSW is proposed in Section 2. Experiments are discussed in Section 3. Finally, we conclude the work in Section 4.

II. Mining of Frequent Itemsets over Transaction Sensitive Sliding Windows

A. Problem Definition

Let $\Psi = \{i_1, i_2, \ldots, i_m\}$ be a set of **items**. A **transaction** $T = (tid, x_1x_2\cdots x_n)$, $x_i \in \Psi$, for $1 \le i \le n$, is a set of items, while n is called the **size** of the transaction, and *tid* is the unique identifier of the transaction. An **itemset** is a non-empty set of items. An itemset with size k is called a *k-itemset*. A **transaction data stream** $TDS = T_1, T_2, \ldots, T_N$ is a continuous sequence of transactions, where N is the tid of latest incoming transaction T_N.

A **transaction-sensitive sliding window** (*TransSW*) in the transaction data stream is a window that slides forward for every transaction. The window at each slide has a fixed number, w, of transactions, and w is called the *size* of the window. Hence, the *current transaction-sensitive sliding window* is $TransSW_{N-w+1} = [T_{N-w+1}, T_{N-w+2}, \ldots, T_N]$, where $N-w+1$ is the window id of current *TransSW*. The **support** of an itemset X over *TransSW*, denoted as $\mathbf{sup}(X)^{TransSW}$, is the number of transactions in *TransSW* containing X as a *subset*. An itemset X is called a **frequent itemset** (*FI*) if $\mathbf{sup}(X)^{TransSW} \ge s \cdot w$, where s is a user-defined minimum support threshold (MST) in the range of [0, 1]. The value $s \cdot w$ is called the **frequent threshold** of *TransSW* ($FT^{TransSW}$).

Given a transaction-sensitive sliding window *TransSW*, and a MST s, the problem of online mining of frequent itemsets in recent transaction data streams is to mine the set of all frequent itemsets by *one scan* of the *TransSW*.

Transaction Data Stream	FIs in TransSW$_1$	FIs in TransSW$_2$
$<T_1, (acd)>$ $<T_2, (bce)>$ $<T_3, (abce)>$ $<T_4, (be)>$	(*a*), (*b*), (*c*), (*e*), (*ac*), (*bc*), (*be*), (*ce*), (*bce*)	(*b*), (*c*), (e), (*bc*), (*be*), (*ce*), (*bce*)

A transaction data stream is formed by transactions arriving in series

Figure 1: An example transaction data stream and the frequent itemsets over two consecutive TransSWs

Example 1 Let the first four transactions in a transaction data stream be $<T_1, (acd)>$, $<T_2, (bce)>$, $<T_3, (abce)>$, and $<T_4, (be)>$, where T_1, T_2, T_3, and T_4 are transactions and *a*, *b*, *c*, *d*, and *e* are items. Let the size of sliding window w be 3 and the user-defined minimum support threshold s be 0.6. Hence, the transaction data stream consists of two transaction-sensitive sliding windows, i.e., $TransSW_1 = [T_1, T_2, T_3]$ and $TransSW_2 = [T_2, T_3, T_4]$, where first window $TransSW_1$ contains the transactions T_1, T_2, and T_3, and the second window $TransSW_2$ contains the transactions T_2, T_3, and T_4. The example is shown in Figure 1.

In Figure 1, the frequent itemsets in $TransSW_1$ are (*a*), (*b*), (*c*), (*e*), (*ac*), (*bc*), (*be*), (*ce*), and (*bce*), and the frequent

itemsets in $TransSW_2$ are (*b*), (*c*), (*e*), (*bc*), (*be*), (*ce*), and (*bce*). In this instance, we can find that (*a*), and (*ac*) are frequent itemsets in $TransSW_1$, but not frequent ones in $TransSW_2$.

B. The Proposes Algorithm

1) Bit-Sequence Representation

In the proposed algorithm called *MFI-TransSW* (Mining Frequent Itemsets in a Transaction-sensitive Sliding Window), for each item *X* in the current transaction-sensitive sliding window *TransSW*, a *bit-sequence* with *w* bits, denoted as **Bit**(*X*), is constructed. If an item *X* is in the *i*-th transaction of current *TransSW*, the *i*-th bit of **Bit**(*X*) is set to be 1; otherwise, it is set to be 0. The process is called *bit-sequence transform*.

For example, in Figure 1, the first sliding window $TransSW_1$ consists of three transactions: <T_1, (*acd*) >, <T_2, (*bce*) >, and <T_3, (*abce*) >, but the $TransSW_2$ consists of transactions: <T_2, (*bce*) >, <T_3, (*abce*) >, and <T_4, (*be*) >. Because item *a* appears in the 1st and 3rd transactions of $TransSW_1$, the bit-sequence of *a*, **Bit**(*a*), is 101. Similarly, **Bit**(*b*) = 011, **Bit**(*c*) = 111, **Bit**(*d*) = 100, and **Bit**(*e*) = 011.

Window-id	Transactions	Bit-Sequences of items
$TransSW_1$	<T_1, (*acd*) > <T_2, (*bce*) > <T_3, (*abce*) >	**Bit**(*a*) = 101, **Bit**(*c*) = 111, **Bit**(*d*) = 100, **Bit**(*b*) = 011, **Bit**(*e*) = 011
$TransSW_2$	<T_2, (*bce*) > <T_3, (*abce*) > <T_4, (*be*) >	**Bit**(*a*) = 010, **Bit**(*c*) = 110, **Bit**(*d*) = 000, **Bit**(*b*) = 111, **Bit**(*e*) = 111

Figure 2: Bit-sequences of items in window initialization phase of TransSW

tid	Items	Bit-Sequences in current $TransSW_1$
T_1	(*acd*)	**Bit**(*a*)=100, **Bit**(*c*)=100, **Bit**(*d*)=100
T_2	(*bce*)	**Bit**(*a*)=100,**Bit**(*c*)=110,**Bit**(*d*)=100,**Bit**(*b*)=010, **Bit**(*e*)=010
T_3	(*abce*)	**Bit**(*a*)=101,**Bit**(*c*)=111,**Bit**(*d*)=100, **Bit**(*b*)=011, **Bit**(*e*)=011

Figure 3: Bit-sequences of items after sliding $TransSW_1$ to $TransSW_2$

2) MFI-TransSW Algorithm

The proposed MFI-TransSW algorithm consists of three phases, *window initialization* phase, *window sliding* phase, and *frequent itemsets generation* phase.

a) Window Initialization Phase

The phase is activated while the number of transactions generated so far in a transaction data stream is less than or equal to a user-predefined sliding window size *w*. In this phase, each item in the new incoming transaction is transformed into its bit-sequence representation.

For instance, in Figure 1, the first sliding window $TransSW_1$ contains three transactions: T_1, T_2, and T_3. The bit-sequences of items of $TransSW_1$ in the window initialization phase are shown in Figure 2.

Algorithm MFI-TransSW

Input: *TDS* (a transaction data stream), *s* (a user-defined minimum support threshold in the range of [0, 1]), and *w* (the user-specified sliding window size).

Output: a set of frequent itemsets, *FI-Output*.

```
Begin
  TransSW = NULL;  /* TransSW consists of w transactions */
  Repeat:
    for each incoming transaction T_i in TransSW do
      if TransSW = FULL then
        Do bitwise-shift on bit-sequences of all items in TransSW;
      else
        for each item X in T_i do
          Do bit-sequence transform(X);
        end for
      end if
    end for
    for each bit-sequence Bit(X) in TransSW do
      if sup(X) = 0 then
        Drop X from TransSW;
      end if
    end for
  /* The following is the frequent itemsets generation phase. The phase is
              performed only when requested by users. */
    FI_1 = {frequent 1-itemsets};
    for (k=2; FI_{k-1} ≠ NULL; k++) do
      CI_k = CIGA(FI_{k-1});
      Do bitwise AND to find the supports of CI_k;
      for each candidate c_k ∈ CI_k do
        if sup(c_k)^TransSW ≥ w·s then
          FI_k = {c_k ∈ CI_k | sup(c_k)^TransSW ≥ w·s};
        end if
      end for
    end for
  FI-Output = ∪_k FI_k;
End
```

Figure 4: Algorithm MFI-TransSW

b) Window Sliding Phase

The phase is activated after the sliding window *TransSW* becomes full. A new incoming transaction is appended to the sliding window, and the oldest transaction is removed from the window.

For removing oldest information, an efficient method is used in the proposed algorithm. Based on the bit-sequence representation, MFI-TransSW algorithm uses the *bitwise left shift* operation to remove the aged transaction from the set of items in the current sliding window. After sliding the window, an effective pruning method, called *Item-Prune*, is used to improve the memory usage. The pruning approach is that *an item X in the current transaction-sensitive sliding window is dropped if and only if* $\boldsymbol{sup}(X)^{TransSW} = 0$.

For example, in Figure 1, before the fourth transaction <T_4, (*be*)> is processed, the first transaction T_1 must be removed from the current window using bitwise left shift on the set of items. Hence, **Bit**(*a*) is modified from 101 to 010. Similarly, **Bit**(*c*)=110, **Bit**(*d*)=000, **Bit**(*b*)=110, and **Bit**(*e*)=110. Then, the new transaction <T_4, (*be*)> is processed by bit-sequence transform. The result is shown in Figure 3. Note that item *d* is

dropped since **Bit**(d)=000, i.e., **sup**(d)^{TransSW} = 0.

c) Frequent Itemsets Generation Phase

The phase is performed only when the up-to-date set of frequent itemsets is requested. In this phase, MFI-TransSW algorithm uses a level-wise method to generate the set of candidate itemsets CI_k (candidate itemsets with k items) from the pre-known frequent itemsets FI_{k-1} (frequent itemsets with k-1 items) according to the *Apriori* property [1][1]. The step is called CIGA (Candidate Itemset Generation using Apriori property). Then, the proposed algorithm uses the *bitwise AND* operation to compute the support (the number of bit 1) of these candidates in order to find the frequent k-itemsets FI_k. The candidate-generation-then-testing process is stopped until no new candidates with k+1 items (CI_{k+1}) are generated. The MFI-TransSW algorithm is shown in Figure 4.

For instance, consider the bit-sequences of $TransSW_2$ in Figure 3, and let the minimum support threshold s be 0.6. Hence, an itemset X is *frequent* if **sup**(X)$^{TransSW} \geq 0.6 \cdot 3 = 1.8$. In the following, we discuss the step of frequent itemset mining of $TransSW_2$. The generated patterns are shown in Figure 1.

First, MFI-TransSW algorithm generates three candidate 2-itemsets, (*bc*), (*be*) and (*ce*), by combining frequent 1-itemsets: (*b*), (*c*) and (*e*), where **Bit**(*b*) = 111, i.e., **sup**(*b*) = 3, **Bit**(*c*) = 110, i.e., **sup**(*c*) = 2, and **Bit**(*e*) = 110, i.e., **sup**(*e*) = 2. 1-itemset (*a*) is an *infrequent* itemset, since its **Bit**(*a*) = 010, i.e., **sup**(*a*) = 1. All these candidates are frequent itemsets after using bitwise AND operations to count the supports of these candidates. Because the **Bit**(*bc*) is 110, the support of candidate 2-itemset *bc* are 2, i.e., **sup**(*bc*) = 2. Similarity, **sup**(*be*) = 3, and **sup**(*ce*) = 2. Second, MFI-TransSW generates one candidate 3-itemset (*bce*) according to Apriori property and uses bitwise AND operation to count the **sup**(*bce*) = 2, i.e., **Bit**(*bc*) AND **Bit**(*be*) AND **Bit**(*ce*) = 110. Because no new candidates are generated, the generation-then-test process is stopped. Hence, there are six frequent itemsets, (*b*), (*c*), (*bc*), (*be*), (*ce*), (*bce*), generated by MFI-TransSW algorithm in $TransSW_2$. The process is shown in Figure 5.

Transactions in $TransSW_2$	Bit-Sequences in $TransSW_2$
$<T_2$, (*bce*) >	**Bit**(*a*) = 010
$<T_3$, (*abce*) >	**Bit**(*c*) = 110
$<T_4$, (*be*) >	**Bit**(*b*) = 111
	Bit(*e*) = 111

FI_1 in $TransSW_2$ (s = 0.6)	sup
{(*b*) | **Bit**(*b*) = 111}	3
{(*c*) | **Bit**(*c*) = 110}	2
{(*e*) | **Bit**(*e*) = 111}	3

CI_2 in SW_2
{(*bc*) | **Bit**(*b*) = 111 AND **Bit**(*c*) = 110}
{(*be*) | **Bit**(*b*) = 111 AND **Bit**(*e*) = 111}
{(*ce*) | **Bit**(*c*) = 110 AND **Bit**(*e*) = 111}

FI_2 in $TransSW_2$	sup
{(*bc*) | **Bit**(*bc*) = 110}	2
{(*be*) | **Bit**(*be*) = 111}	3
{(*ce*) | **Bit**(*ce*) = 110}	2

CI_3 in $TransSW_2$
{(*bce*) | **Bit**(*bc*) = 110 AND **Bit**(*be*) = 111 AND **Bit**(*ce*) = 110}

FI_3 in $TransSW_2$	sup
{(*bce*) | **Bit**(*bce*) = 110}	2

Figure 5: Steps of frequent itemsets generation in $TransSW_2$

[1] It is a **downward closure property**, i.e., *if a pattern is frequent, all of its sub-patterns will also be frequent*.

In the next section, we will discuss the problem of online mining of frequent itemsets over *time-sensitive* stream sliding windows and extend the MFI-TransSW algorithm to solve this problem.

III. EXPERIMENTS

In this section, we report the experimental results of the proposed algorithm MFI-TransSW. All the programs are implemented using Microsoft Visual C++ Version 6.0 and performed on a 1.80 GHz Pentium(R) PC machine with 512 MB memory running on Windows 2000. For testing frequent itemset mining over sliding windows, we generate online data streams using IBM synthetic data generator proposed by Agrawal and Srikant [1, 2]. The synthetic data stream, denoted by T5.I4.D1000K, of size 1 million transactions (D1000K) has an average transaction size of 5 items (T5) with average maximal frequent itemset size of 4 items (I4). In all experiments, the transactions of T5.I4D1000K are looked up in sequence to simulate the environment of an online data stream.

A. Performance Evaluation of MFI-TransSW algorithm

Because the the purpose of the paper is on frequent itemsets mining over online data streams with a transaction-sensitive sliding window, we mainly address it by comparison with the SWFI-stream algorithm [5]. The experiments of memory usage are shown in Figures 6, 7, and 8, and the processing times are shown in Figures 9 and 10. The minimum support threshold s and the size of a sliding window w are set to 0.1% and 20,000, respectively. As shown in these experiments, MFI-TransSW significantly outperforms SWFI-stream for both memory consumption and CPU cost.

Figure 6 shows the memory usage of the window initialization phase. As shown in Figure 6, MFI-TransSW algorithm requires only about 2.1 MB in window initialization phase, but the memory requirement of SWFI-stream is increased linearly from 11.2 MB to 109.7 MB. Figure 7 shows the memory usage of the window sliding phase. In this phase, the memory requirement of MFI-TransSW is also approximately 2.1 MB, but that of SWFI-stream is between 109.7 MB to 120.3 MB. Figure 8 gives the memory usage of the frequent itemsets generation phase. In this phase, the memory requirement of MFI-TransSW is between 33.5MB to 39MB. As shown in Figures 6-8, MFI-TransSW algorithm outperforms SWFI-stream for memory consumption.

Figure 9 shows the processing time of window initialization phase under different window sizes from 20,000 (200K) transactions to 100,000 (1,000K) transactions. Figure 10 shows the total time of window sliding time and pattern mining time at each 100K transactions using various window sizes from 200K transactions to 1000K transactions. As shown in Figures 9 and 10, MFI-TransSW algorithm outperforms SWFI-stream for processing time consumption.

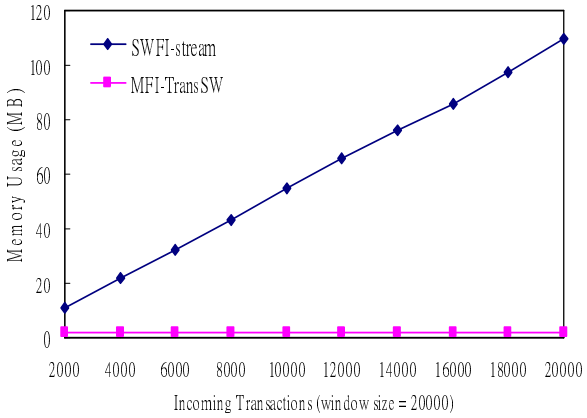

Figure 6. Memory usages in window initialization phases of algorithms SWFI-stream and MFI-TransSW.

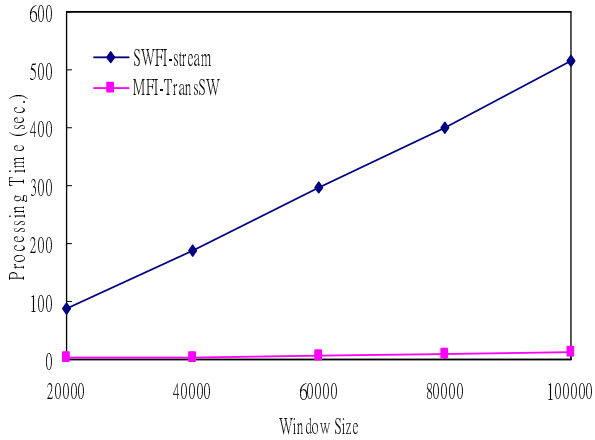

Figure 9. Processing time in window initialization phases of algorithms SWFI-stream and MFI-TransSW under different window sizes.

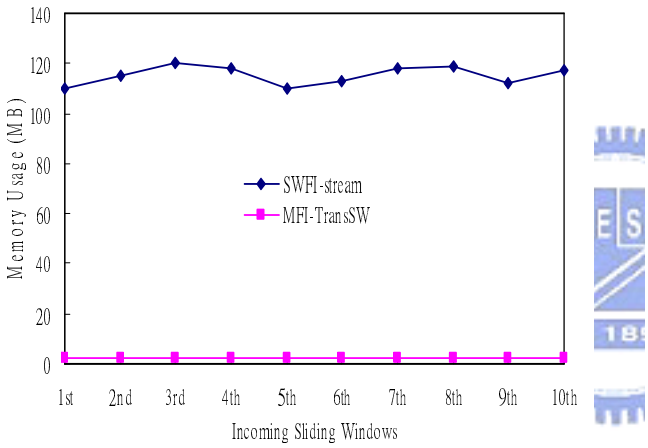

Figure 7. Memory usages in window sliding phases of algorithms SWFI-stream and MFI-TransSW.

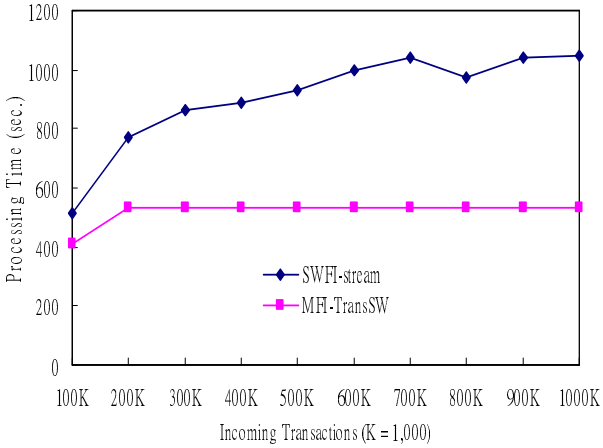

Figure 10. Processing time including window sliding time and pattern generation time of algorithms SWFI-stream and MFI-TransSW under window size 200K transactions.

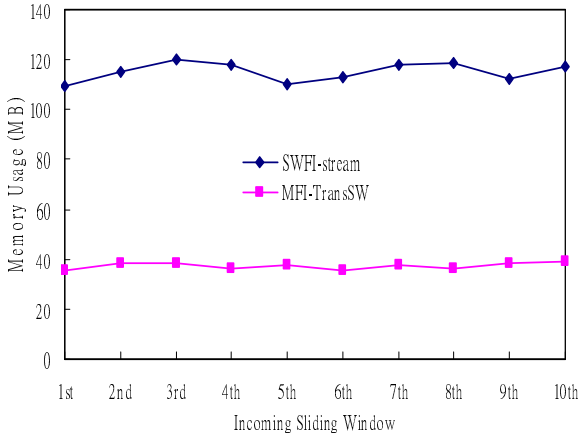

Figure 8. Memory usages in frequent itemset generation phases of algorithms SWFI-stream and MFI-TransSW.

IV. Conclusions

In this paper, we proposed an efficient one-pass algorithm, called MFI-TransSW, for mining frequent itemsets over online data streams with a transaction-sensitive sliding window. Experiments show that the proposed algorithm not only attain highly accurate mining results, but also run significant faster and consume less memory than existing algorithms for mining frequent itemsets over online data streams.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proceedings

of the 1993 International Conference on Management of Data, pp. 207-216, 1993.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499, 1994.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In Proceedings of the 21th ACM SIGMOD-SIGACT-AIGART Symposium on Principles of Database Systems, pp. 1-16, 2002.

[4] J. Chang and W. Lee. Decaying Obsolete Information in Finding Recent Frequent Itemsets over Data Stream. IEICE Transaction on Information and Systems, Vol. E87-D, No. 6, June, 2004.

[5] J. Chang and W. Lee. A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams. Journal of Information Science and Engineering, Vol. 20, No.4, July, 2004.

[6] Y. Chi, H. Wang, P. Yu, and R. Muntz. MOMENT: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In Proceedings of the 4th IEEE International Conference on Data Mining, pp. 59-66, 2004.

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

[8] L. Golab and M. T. Özsu. Issues in Data Stream Management. ACM SIGMOD Record, 32(2): 5-14, June, 2003.

[9] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In Proceedings of the 2000 International Conference on Management of Data, pp. 1-12, 2000.

[10] N. Jiang, and L. Gruenwald. Research Issues in Data Stream Association Rule Mining. In SIGMOD Record, Vol. 35, No. 1, Mar. 2006.

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

[12] H.-F. Li, S.-Y. Lee, and M.-K. Shan, An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams, in Proc. of First International Workshop on Knowledge Discovery in Data Streams (IWKDDS), 2004.

[13] H.-F. Li., S.-Y. Lee, and M.-K. Shan, Online Mining (Recently) Maximal Frequent Itemsets over Data Streams, in Proc. of the 15th IEEE International Workshop on Research Issues on Data Engineering (RIDE), 2005.

[14] C.H. Lin, D.Y. Chiu, Y.H. Wu and A.L.P. Chen. Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In Proceedings of 2005 SIAM International Conference on Data Mining, 2005.

[15] G. S. Manku, and R. Motwani. Approximate Frequency Counts Over Data Streams. In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 346-357, 2002.

[16] W.-G. Teng, M.-S. Chen, and P. S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In Proceedings of the 29th International Conference on Very Large Data Bases, pp. 93-104, 2003.

[17] W.-G. Teng, M.-S. Chen, and P. S. Yu. Using Wavelet-based Resource-Aware Mining to Explore Temporal and Support Count Granularities in Data Streams,. In Proceedings of the 4th SIAM International Conference on Data Mining (SDM-04), April 22-24, 2004.

[18] R.C.W. Wong, and A. Fu. Mining Top-K Itemsets over a Sliding Window Based on Zipfian Distribution. In Proceedings of 2005 SIAM International Conference on Data Mining, 2005.

[19] J.-X. Yu, Z. Chong, H. Lu, and A. Zhou. False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In Proceedings of the 30th International Conference on Very Large Data Bases, pp. 204-215, 2004.

[20] Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 358-369, 2002.