Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 36 (2009) 1466–1477
Expert Systems with Applications
www.elsevier.com/locate/eswa

Mining frequent itemsets over data streams using efficient window sliding techniques

Hua-Fu Li *, Suh-Yin Lee

Department of Computer Science, Kainan University, Taiwan
Department of Computer Science, National Chiao-Tung University, Taiwan

Abstract

Online mining of frequent itemsets over a stream sliding window is one of the most important problems in stream data mining with broad applications. It is also a difficult issue since the streaming data possess some challenging characteristics, such as unknown or unbound size, possibly a very fast arrival rate, inability to backtrack over previously arrived transactions, and a lack of system control over the order in which the data arrive. In this paper, we propose an effective bit-sequence based, one-pass algorithm, called MFI-TransSW (*M*ining *F*requent *I*temsets within a *Trans*action-sensitive *S*liding *W*indow), to mine the set of frequent itemsets from data streams within a transaction-sensitive sliding window which consists of a fixed number of transactions. The proposed MFI-TransSW algorithm consists of three phases: window initialization, window sliding and pattern generation. First, every item of each transaction is encoded in an effective bit-sequence representation in the window initialization phase. The proposed bit-sequence representation of item is used to reduce the time and memory needed to slide the windows in the following phases. Second, MFI-TransSW uses the left bit-shift technique to slide the windows efficiently in the window sliding phase. Finally, the complete set of frequent itemsets within the current sliding window is generated by a level-wise method in the pattern generation phase. Experimental studies show that the proposed algorithm not only attain highly accurate mining results, but also run significant faster and consume less memory than do existing algorithms for mining frequent itemsets over data streams with a sliding window. Furthermore, based on the MFI-TransSW framework, an extended single-pass algorithm, called MFI-TimeSW (*M*ining *F*requent *I*temsets within a *Time*-sensitive *S*liding *W*indow) is presented to mine the set of frequent itemsets efficiently over time-sensitive sliding windows.

Keywords: Data mining; Data streams; Frequent itemsets; Single-pass algorithms; Sliding windows; Bit-sequence representation

1. Introduction

Mining data streams is one of the most challenging problems in data mining. 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, and performance measurement in network monitoring and traffic management. Data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time. Hence, it is also called streaming data.

Due to the characteristics of data streams, there are some inherent challenges for mining streaming data (Babcock, Babu, Datar, Motwani, & Widom, 2002; Golab & Özsu, 2003; Jiang & Gruenwald, 2006). 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. Based on the above issues, a one scan of streaming data

* Corresponding author.
E-mail addresses: hfli@csie.nctu.edu.tw (H.-F. Li), sylee@csie.nctu.edu.tw (S.-Y. Lee).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2007.11.061

and a proper small and compact data structure of the stream data mining techniques are necessary. Hence, previous studies of multiple-pass algorithms for mining static datasets are not feasible for mining data streams.

Recently, some research results of stream data mining have been studied for modeling and computing data streams (Henzinger, Raghavan, & Rajagopalan, 1998), monitoring statistics over streams (Datar, Ginois, Indyk, & Motwani, 2002), and continuous queries over data streams (Babu & Widom, 2001), multidimensional data analysis (Chen, Dong, Han, Wah, & Wang, 2002), classification (Domingos & Hulten, 2000), clustering (Aggarwal, Han, Wang, & Yu, 2003; Bandyopadhyay et al., 2006), synopsis data structure (Gibbons & Matias, 1999), stream data reduction (Littau & Boley, 2006; Yan et al., 2006), mining frequent itemsets (Chang & Lee, 2003; Chang & Lee, 2004; Chi, Wang, Yu, & Muntz, 2006; Giannella, Han, Pei, Yan, & Yu, 2003; Lee, Lin, & Chen, 2005; Li, Lee, & Shan, 2004; Li, Lee, & Shan, 2005a; Manku & Motwani, 2002; Yu, Chong, Lu, Zhang, & Zhou, 2006), change detection and mining (Dong et al., 2003; Li, Lee, & Shan, 2005b), top-k queries computing (Babcock & Olston, 2003), and mining sequential patterns (Chen, Wu, & Zhu, 2005; Ho, Li, Kuo, & Lee, 2006). In this paper, we consider one of the most challenging problems of stream data mining, i.e., *single-pass mining of frequent itemsets over data streams within a sliding window*.

Compared with previous sliding-window based mining techniques of frequent itemsets (Chang & Lee, 2004; Chi et al., 2006; Lee et al., 2005), we propose an efficient one-pass algorithm called MFI-TransSW (*M*ining *F*requent *I*temsets within a *Trans*action-sensitive *S*liding *W*indow), for online, incremental mining of frequent itemsets in data streams within a transaction-sensitive sliding window. A transaction-sensitive sliding window is composed of a fixed number of transactions. An effective bit-sequence representation of item is developed to maintain the sliding order of window and the itemsets frequencies. Comprehensive experiments show that the proposed MFI-TransSW algorithm not only attains highly accurate mining results, but also runs significant faster and consume less memory than do previous well-known algorithms SWFI-stream (Chang & Lee, 2004), moment (Chi et al., 2006), and SWF (Lee et al., 2005) for mining frequent itemsets over sliding windows. Furthermore, based on the MFI-TransSW framework, an efficient single-pass algorithm, called MFI-TimeSW (*M*ining *F*requent *I*temsets within a *Time*-sensitive *S*liding *W*indow), is developed for mining the set of frequent itemsets over data streams within a time-sensitive sliding window. A time-sensitive sliding window consists of a fixed number of time-units.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. The problem definition of mining frequent itemsets in data streams within a transaction-sensitive sliding window is described in Section 3. In Section 4, the MFI-TransSW algorithm is proposed to mine the set of frequent itemsets in a transaction-sensitive sliding window. Moreover, based on MFI-TransSW, the MFI-TimeSW algorithm is also proposed for mining time-sensitive sliding windows in this section. Comprehensive experiments of the proposed algorithms are discussed in Section 5. Finally, we conclude the work in Section 6.

2. Related work

Many previous studies contributed to the efficient mining of frequent itemsets over data streams (Chang & Lee, 2003, 2004; Chi et al., 2006; Giannella et al., 2003; Lee et al., 2005; Li et al., 2004, 2005; Manku & Motwani, 2002; Yu et al., 2006). According to the stream data processing model (Zhu & Shasha, 2002), the research of mining frequent itemsets in data streams can be divided into three categories: *landmark-window based mining* (Li et al., 2004, 2005; Manku & Motwani, 2002; Yu et al., 2006), *damped-window based mining* (Chang & Lee, 2003; Giannella et al., 2003), and *sliding-window based mining* (Chang & Lee, 2004; Chi et al., 2006; Lee et al., 2005).

In a landmark window model, frequent pattern mining is performed based on the values between a specific timestamp called landmark and the present. In Manku and Motwani (2002), developed two single-pass algorithms, Sticky-Sampling and Lossy Counting, to mine frequent items over 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 data streams. A lattice-based in-memory data structure is used in the BTS algorithm to store itemsets, approximate supports of itemsets, and maximum possible errors in the approximate supports. Li et al. proposed the online incremental projected, single-pass algorithms, called DSM-FI (Li et al., 2004) and DSM-MFI (Li et al., 2005), to mine the set of all frequent itemsets and maximal frequent itemsets over the entire history of data streams. Prefix-tree based data structures are developed in the proposed algorithms to store items and their support values, window ids, and nodes links pointing to the root or a certain node. Yu et al. (2006) proposed efficient algorithms based on Chernoff bound for false negative or false positive mining of frequent itemsets from high speed transactional data streams. The algorithms use a running error parameter to prune itemsets and use a reliability parameter to control memory.

In the damped window model, recent windows are more important than previous ones. In other words, older transactions contribute less weight toward itemsets support. Chang and Lee (2003) developed a damped window based algorithm, called estDec, for mining frequent itemsets over streaming data in which each transaction has a weight decreasing with age. Two efficient techniques, delayed-insertion and pruning, are used in estDec to enhance the mining performance. Giannella et al. (2003) proposed the FP-stream algorithm to find the set of frequent itemsets at multiple time granularities by a novel titled-time windows technique. A frequent pattern tree (FP-tree) (Han,

Pei, Yin, & Mao, 2004) is used to maintain the frequent itemsets and their supports stored in titled-time windows.

In a 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 transaction-sensitive sliding window is an expired transaction while the basic unit of window sliding of time-sensitive sliding window is a time unit, such as a minute or 1 h.

Lee et al. (2005) proposed a sliding-window filtering (SWF) algorithm for incremental mining of frequent itemsets within a sliding window. The sliding window is composed of a sequence of partitions. Each partition maintains a number of transactions. SWF uses a filtering threshold in each partition to deal with the candidate itemsets generation. All candidate 2-itemsets are maintained separately. When the window is sliding, the candidate 2-itemsets of the new incoming partition are modified. Subsequently, all possible candidate itemsets are generated from these candidate 2-itemsets. The set of frequent itemsets is generated by scanning the entire window. In the SWF algorithm, to get an up-to-date mining result, all the transactions within the current window should be re-scanned.

Chang and Lee (2004) proposed a BTS-based algorithm, called SWFI-stream, for finding frequent itemsets within a transaction-sensitive sliding window. A prefix tree lattice structure called monitoring lattice is constructed to store the current candidate itemsets and their frequencies. Another in-memory data structure called current transaction list (CTL) is developed to maintain all the transactions in the range of current sliding window. For each incoming transaction, SWFI-stream inserts this transaction into the CTL and enumerates the transaction into a set of itemsets. Furthermore, SWFI-stream inserts these itemsets into the current monitoring lattice. When the current window is sliding, the oldest transaction in the CTL is extracted and every itemsets embedded in the transaction is deleted from the current monitoring lattice. Finally, all the currently frequent itemsets in the monitoring lattice are found by traversing all the paths of the monitoring lattice.

Chi et al. (2006) proposed the first streaming algorithm, called Moment, to mine the set of *closed frequent itemsets* within a transaction-sensitive sliding window. A compact prefix-tree based data structure, called closed enumeration tree (CET), is developed in the Moment algorithm to maintain a dynamically selected set of itemsets which includes four types of nodes: infrequent gateway nodes, unpromising gateway nodes, intermediate nodes and closed nodes. The selected itemsets is composed of a boundary between closed frequent itemsets and the rest of the itemsets. When a new transaction arrives, Moment algorithm traverses the parts of the CET that related to the new transaction. For each related node, Moment updates its support, tid_sum and possibly its node type property. When an oldest transaction is deleted from the current sliding window, Moment also traverses the parts of the CET that is related to the deleted transaction and judges the related node types. Furthermore, the exploration and node type checking are time consuming.

In this paper, we propose two efficient one-pass algorithms, MFI-TransSW and MFI-TimeSW, to mine the set of frequent itemsets online and incrementally within a sliding window by using an effective bit-sequence representation of items. Before we describe our bit-sequence data structure and the proposed algorithms, let us first explain the definitions that are going to be used in this paper.

3. Problem definition

Let $\Psi = \{i_1, i_2, \ldots, i_m\}$ be a set of *items*. A *transaction* $T = (tid, x_1x_2\ldots x_n)$, $x_i \in \Psi$, for $1 \leqslant i \leqslant n$, 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 $TransSW_{Nw+1} = [T_{N-w+1}, T_{N-w+2}, \ldots, T_N]$, where $N - w + 1$ is the window identifier 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 $\mathbf{sup}(X)^{TransSW} \geqslant s \cdot w$, 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*.

Example 1. Let the first four transactions in a transaction data stream be $\langle T_1, (acd)\rangle$, $\langle T_2, (bce)\rangle$, $\langle T_3, (abce)\rangle$, and $\langle T_4, (be)\rangle$, 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 Fig. 1.

In Fig. 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), (bc), (be), (ce), and

A 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), (bc), (be), (ce), (bce)

A transaction data stream is formed by transactions arriving in series

Fig. 1. An example data stream and the frequent itemsets over two consecutive TransSWs.

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

4. MFI-TransSW: online mining of frequent itemsets within a transaction-sensitive sliding window

In this section, we describe our proposed single-pass mining algorithm, called *MFI-TransSW* (*M*ining *F*requent *I*temsets within a *Trans*action-sensitive *S*liding *W*indow) and its bit-sequence representation of items. Compared with other sliding window based mining techniques, we save memory and improve speed by dynamically maintaining all transactions in the current sliding window by using an effective bit-sequence representation of items.

4.1. Bit-sequence representation of items

In MFI-TransSW algorithm, 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 ith transaction of current *TransSW*, the ith 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 Fig. 1, the first sliding window $TransSW_1$ consists of three transactions: $\langle T_1, (acd)\rangle$, $\langle T_2, (bce)\rangle$, and $\langle T_3, (abce)\rangle$, but the $TransSW_2$ consists of transactions: $\langle T_2, (bce)\rangle$, $\langle T_3, (abce)\rangle$, and $\langle T_4, (be)\rangle$. 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**$(e) = 011$.

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

4.2. Window initialization phase of MFI-TransSW

The window initialization 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 of the new incoming transaction is transformed into its bit-sequence representation.

For example, in Fig. 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 Fig. 2.

4.3. Window sliding phase of MFI-TransSW

The window sliding phase is activated after the current sliding window *TransSW* becomes full. A new incoming transaction is appended to the current 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* $\mathbf{sup}(X)^{TransSW} = 0$.

For example, in Fig. 1, before the fourth transaction $\langle T_4, (be)\rangle$ 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 $\langle T_4, (be)\rangle$ is processed by bit-sequence transform. The result is shown in Fig. 3. Note that item d is dropped since **Bit**$(d) = 000$, i.e., $\mathbf{sup}(d)^{TransSW} = 0$.

4.4. Frequent itemsets generation phase of MFI-TransSW

The frequent itemsets generation 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 (Agrawal & Srikant, 1994).[1] The step is called CIGA (*C*andidate *I*temset *G*eneration using *A*priori 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 Fig. 4.

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

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

Fig. 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

Fig. 3. Bit-sequences of items after sliding $TransSW_1$ to $TransSW_2$.

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

First, MFI-TransSW algorithm generates three candidate 2-itemsets, (bc), (be) and (ce), by combining frequent 1-itemsets: (b), (c) and (e), where $\textbf{Bit}(b) = 111$, i.e.,$\textbf{sup}(b) = 3$, $\textbf{Bit}(c) = 110$, i.e., $\textbf{sup}(c) = 2$, and $\textbf{Bit}(e) = 110$, i.e., $\textbf{sup}(e) = 2$. 1-itemset (a) is an *infrequent* itemset, since its $\textbf{Bit}(a) = 010$, i.e., $\textbf{sup}(a) = 1$. 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., $\textbf{sup}(bc) = 2$. Similarity, $\textbf{sup}(be) = 3$, and $\textbf{sup}(ce) = 2$. Second, MFI-TransSW generates one candidate 3-itemset (bce) according to Apriori property and uses bitwise AND operation to count the $\textbf{sup}(bce) = 2$, i.e., **Bit**(bc) AND **Bit**(be) AND $\textbf{Bit}(ce) = 110$. 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 Fig. 5.

In the next section, we will extend the capabilities of MFI-TransSW algorithm to mine the set of frequent itemsets over data streams within a time-sensitive sliding window.

4.5. Mining frequent itemsets in a time-sensitive sliding window

4.5.1. Problem definition

Let $\Psi = \{i_1, i_2, \ldots, i_m\}$ be a set of **items**. 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 transaction identifier of latest incoming transaction T_N. A *transaction* $T = (TUid, Tid, itemset)$, where $TUid$ is the identifier of the time unit, and Tid is the identifier of the transaction.

A *time-sensitive sliding window* (*TimeSW*) in the transaction data stream is a window that slides forward for every *time unit* (*TU*). Each time unit TU_i consists of a variable number, $|TU_i|$, of transactions, and $|TU_i|$ is also called the *size* of the time unit. Hence, the *current time-sensitive sliding window* with w time units is $TimeSW_{N-w+1} = [TU_{N-w+1}, TU_{N-w+2}, \ldots, TU_N]$, where $N - w + 1$ is the id of time unit of current *TimeSW*, and N is the *TUid* of latest time unit TU_N. The window at each slide has a fixed number, w, of time units. The value $w = |TU_{N-w+1}| + |TU_{N-w+2}| + \cdots + |TU_N|$ is called the *size* of the time-sensitive sliding window and denoted as $|TimeSW|$.

The *support* of an itemset X over *TimeSW*, denoted as $\textbf{sup}(X)^{TimeSW}$, is the number of transactions in *TimeSW* containing X as a *subset*. An itemset X is called a *frequent itemset* (*FI*) if $\textbf{sup}(X)^{TimeSW} \geqslant s \cdot |TimeSW|$, where s is a user-defined minimum support threshold (MST) in the range of $[0, 1]$. The value $s \cdot |TimeSW|$ is called the *frequent threshold* of *TimeSW* (FT^{TimeSW}).

Given a time-sensitive sliding window *TimeSW*, 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 *TimeSW*.

4.5.2. The proposed algorithm MFI-TimeSW

Based on the MFI-TransSW framework, we propose an extended single-pass algorithm, called *MFI-TimeSW* (*M*ining *F*requent *I*temsets within a *Time*-sensitive *S*liding *W*indow), to mine the set of frequent itemsets over data streams within a time-sensitive sliding window. The algorithm description of MFI-TimeSW is shown in Fig. 6. The major differences between MFI-TransSW and MFI-

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**
for each item *X* in T_i **do**
Do *bit-sequence transform*(*X*);
end for
if TransSW = FULL **then**
Do *bitwise-shift* on bit-sequences of all items in TransSW;
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; $FI_{k-1} \neq$ NULL; *k*++) **do**
$CI_k = CIGA(FI_{k-1})$;
Do *bitwise AND* to find the supports of CI_k;
for each candidate $c_k \in CI_k$ **do**
if sup$(c_k)^{TransSW} \geq w \cdot s$ **then**
$FI_k = \{c_k \in CI_k \mid \textbf{sup}(c_k)^{TransSW} \geq w \cdot s\}$;
end if
end for
end for
FI-Output = $\cup_k FI_k$;

End

Fig. 4. Algorithm MFI-TransSW.

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

Fig. 5. Steps of frequent itemsets generation in $TransSW_2$.

TimeSW are the unit of data processing, the bit-sequence transformation of a time unit, the number of sliding transactions, and the dynamic frequent threshold of itemsets. These issues are discussed as follows.

Algorithm MFI-TimeSW

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

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

```
Begin
    TimeSW = NULL;   /* TimeSW consists of w time units */
    Repeat:                    /*  N is the id of current time unit*/
      for each new time unit TU_N from TDS do    /*  N  ≥ 1*/
          for each transaction T_i of TU_N do
              for each item X in T_i do
                  Do bit-sequence transform(X);
              end for
          end for
          if TimeSW = FULL then
              Do |TU_{N-w+1}| times of bitwise-shift operation on bit-
              sequences of all items in TimeSW;
          end if

      end for
      for each bit-sequence Bit(X) in TimeSW do
          if sup(X) = 0 then
              Drop X from TimeSW;
          end if
      end for
    N = N +1;

/* 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)^TimeSW ≥ |TimeSW|·s then
                    FI_k = {c_k ∈ CI_k | sup(c_k)^TimeSW ≥ |TimeSW|·s};
                end if
            end for
        end for
    FI-Output = ∪_k FI_k;
End
```

Fig. 6. Algorithm MFI-TimeSW.

(1) *Unit of data processing:* For MFI-TransSW algorithm, the unit of data processing is each incoming transaction, but the unit of data processing of MFI-TimeSW is each new time unit. However, each time unit contains variable number of transactions. Hence, a list of time unit (TU-list) is developed in the MFI-TimeSW algorithm to solve the issue of data processing unit. A TU-list is a list of time unit entries, where each time unit entry records the number of transactions within the time unit, i.e., $\textit{TU-list} = \langle(TUid_{N-w+1}, |TU_{N-w+1}|), (TUid_{N-w+2}, |TU_{N-w+2}|), \ldots, (TUid_N, |TU_N|)\rangle$, where $|\mathrm{TU}_i|$ is the number of transactions within the time unit TU_i.

(2) *Bit-sequence transformation of a time unit*: For the bit-sequence transformation of a time unit, a modified transformation process is proposed in MFI-TimeSW algorithm. For each item X in the current time-sensitive stream sliding window $TimeSW_{N-w+1}$, a *bit-sequence* with $|TimeSW_{N-w+1}|$ bits, denoted as $\mathbf{Bit}(X)^{TimeSW}_{N-w+1}$, is constructed. Similarly, if the item X is in the i-th transaction of $TimeSW_{N-w+1}$, the ith bit of $\mathbf{Bit}(X)^{TimeSW}_{N-w+1}$ is set to be 1. Otherwise, it is set to be 0.

(3) *Number of sliding transactions*: In the window sliding phase of MFI-TimeSW algorithm, after the oldest time unit TU_{N-w+1} is removed from the current sliding

window, a new time unit TU_{N+1} is appended to the window. If the aged time unit TU_{N-w+1} contains d transactions, MFI-TimeSW performs d times of bit-wise left shift operation on the current sliding window. After that, MFI-TimeSW uses that same pruning method *Item-Prune* to improve the memory usage in mining process.

(4) *Dynamic frequent threshold of itemsets:* In the MFI-TransSW algorithm, the constant value $s \cdot w$ is the frequent threshold of itemsets, where s is the user-specified minimum support threshold in the range of $[0, 1]$ and w is the size of sliding window. However, in the MFI-TimeSW algorithm, the value of frequent threshold is $s \cdot |TimeSW|$ is a dynamic value, where $|TimeSW| = |TU_{N-w+1}| + |TU_{N-w+2}| + \cdots + |TU_N|$.

5. Performance evaluation

In this section, we will describe the experimental evaluation of the proposed algorithms: MFI-TransSW and MFI-TimeSW. All the programs are implemented using Microsoft Visual C++ Version 6.0 and performed on a 1.80 GHz Pentium(R) PC machine with 1024 MB memory running on Windows n000. For testing frequent itemset mining over sliding windows, synthetic data streams are generated by using IBM synthetic data generator proposed by Agrawal and Srikant (1994). Two synthetic data streams, denoted by T5.I4.D1000K and T15.I6.D1000K are generated, where T, I and D mean the average transaction length, the average length of the maximal frequent itemset, and the total number of transactions, respectively. To simulate data streams, the transactions in both synthetic data are looked up in sequence and feed them in the buffer. The parameter setting used in the experiments is shown in Table 1.

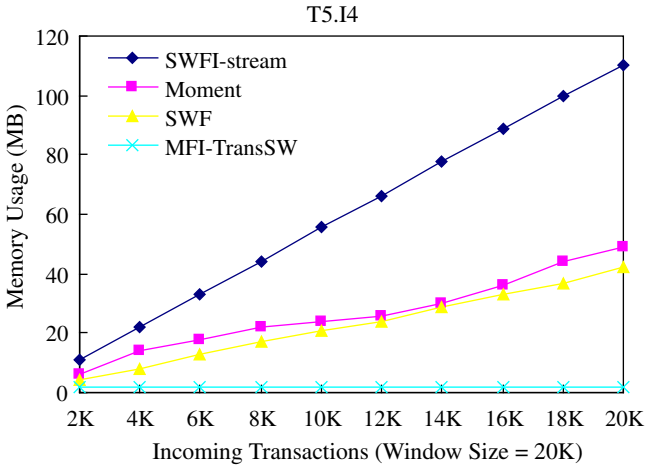
5.1. Performance evaluation of MFI-TransSW algorithm

In this section, we compare our algorithm MFI-TransSW with sliding-window based algorithms, SWF (Lee et al., 2005), Moment (Chi et al., 2006), and SWFI-stream (Chang & Lee, 2004). In the following experiments, the minimum support threshold is 0.1% and the size of the sliding window is 20 K transactions.

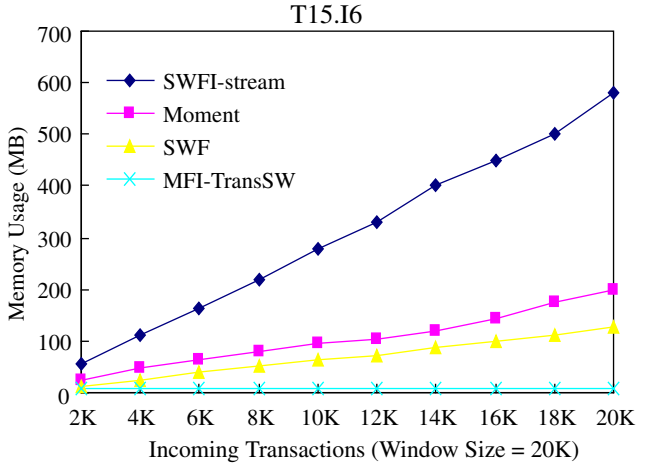
Fig. 7 shows the comparison of memory requirements in the window initialization phase. Fig. 7a shows the memory usage of window initialization on T5.I4 and Fig. 7b gives the memory requirement on T15.I6. From the figures we can see that as window size increases, the memory usage for all algorithms grows. However, the memory requirement of our MFI-TransSW algorithm is significantly less than that of SWF, Moment, and SWFI-stream.

Table 1
Parameters used in the experiments

Parameter	Description	Value
D	Number of transactions in data streams	1000 K
N	Number of distinct items	1 K
I	Average length of maximal frequent itemsets	4, 6
T	Average length of transactions	5, 15
s	Minimum support thresholds	0.1%



(a) T5.I4

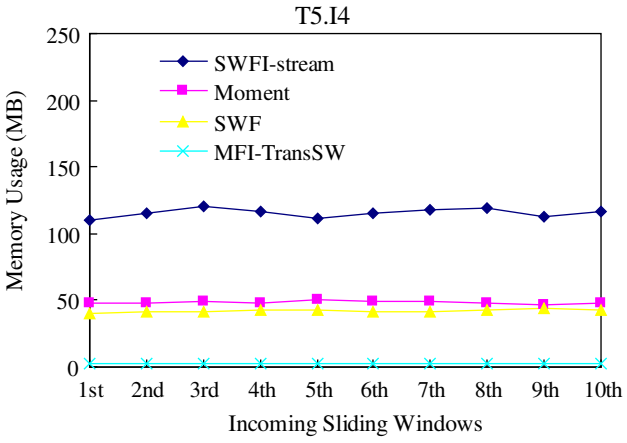


(b) T15.I6

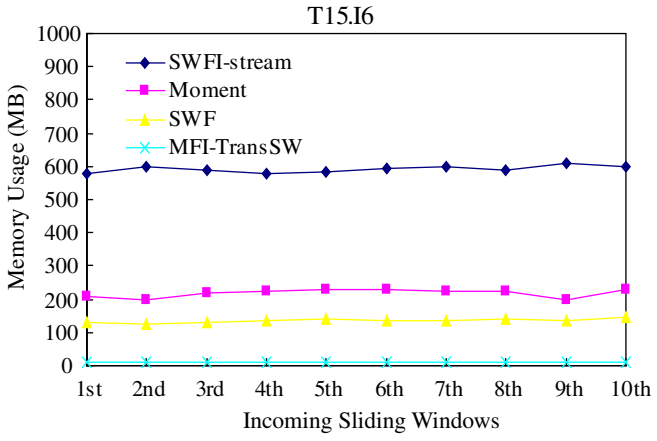
Fig. 7. Comparisons of memory usages in the window initialization phase ($s = 0.1\%$).

Fig. 8 gives the comparison of memory usage in the window sliding phase. Fig. 8a shows the memory requirement of window sliding on T5.I4.D and Fig. 8b gives the memory requirement on T15.I6. As can be seen from the figures, from T5.I4 to T15.I6, the memory usages for algorithms SEFI-stream, Moment and SWF grow dramatically except our algorithm MFI-TransSW. This implies that the item with bit-sequence representation of MFI-TransSW remains approximately the same.

Fig. 9 shows the comparison of memory usage in the frequent itemset mining phase. Fig. 9a shows the memory requirement on T5.I4 and Fig. 9b gives the memory requirement on T15.I6. In this experiment, the up-to-date set of frequent itemsets is requested when a new window

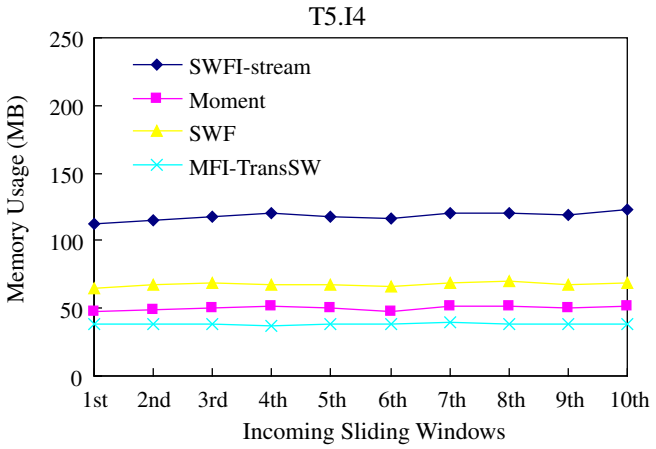


(a) T5.I4

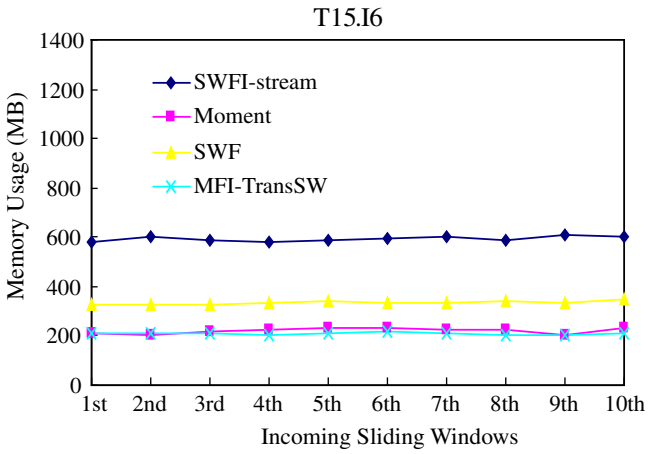


(b) T15.I6

Fig. 8. Comparisons of memory usages in the window sliding phase ($s = 0.1\%$).



(a) T5.I4



(b) T15.I6

Fig. 9. Comparisons of memory usages in the frequent itemset generation phase ($s = 0.1\%$). (a) T5.I4, (b) T15.I6.

is arrived. From the figures we can see that the memory requirement of MFI-TransSW in the frequent itemset mining phase is less than that of SWFI-stream, Moment and SWF.

As shown in the above experiments, the proposed MFI-TransSW algorithm significantly outperforms other sliding-window based data mining algorithms for memory consumption.

Fig. 10 shows the processing time of window initialization phase under various window sizes from 20 K to 100 K transactions. Fig. 10a shows the processing time on T5.I4 and Fig. 10b gives the processing time on T15.I6. From the figures, we can see that as the window size increase, the processing time of window initialization for all algorithms grows. However, the response time of MFI-TransSW is faster than that of SWF, Moment and SWFI-stream by more than an order of magnitude under different window sizes from 20 K to 100 K transactions.

Fig. 11 shows the processing time of window sliding phase under different sizes of sliding window. Fig. 11a shows the processing time on T5.I4 and Fig. 11b gives the processing time on T15.I6. As can be seen from the figures, as the size of window increases, the processing time for all algorithms grows. However, the sliding time of MFI-TransSW is less than that of SWF, Moment and SWFI-stream. This is because that the sliding time of the proposed bit-sequence representation is fast than other algorithms.

Fig. 12 shows the processing time of frequent itemset mining phase under different window sizes. Fig. 12a shows the processing time on T5.I4 and Fig. 12b gives the processing time on T15.I6. From the figures, we can see that as the sliding window size increases, the processing time of pattern discovery for all algorithms increases. Although the processing time of mining frequent itemsets of our MFI-TransSW algorithm is slower than that of Moment and SWFI-stream, the total processing time, which includes window initialization time, window sliding time and frequent itemset mining time, of MFI-TransSW is faster than that of SWF, Moment and SWFI-stream by more than an order of magnitude. In conclusion, the proposed MFI-TransSW algorithm is a time-efficient method for

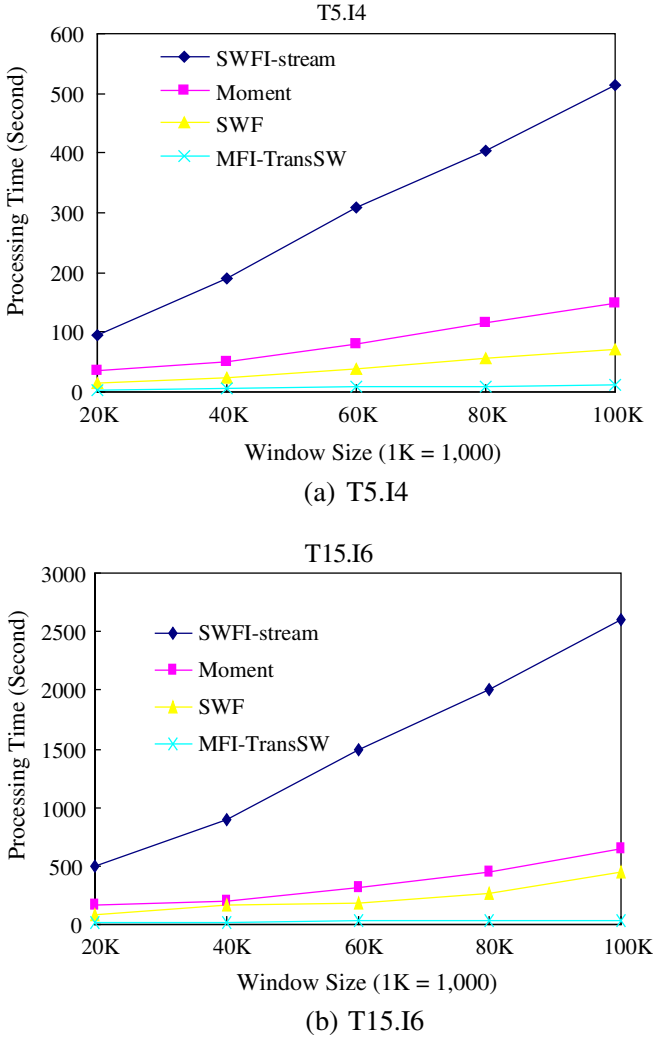


Fig. 10. Processing time of window initialization phase under different window sizes ($s = 0.1\%$).

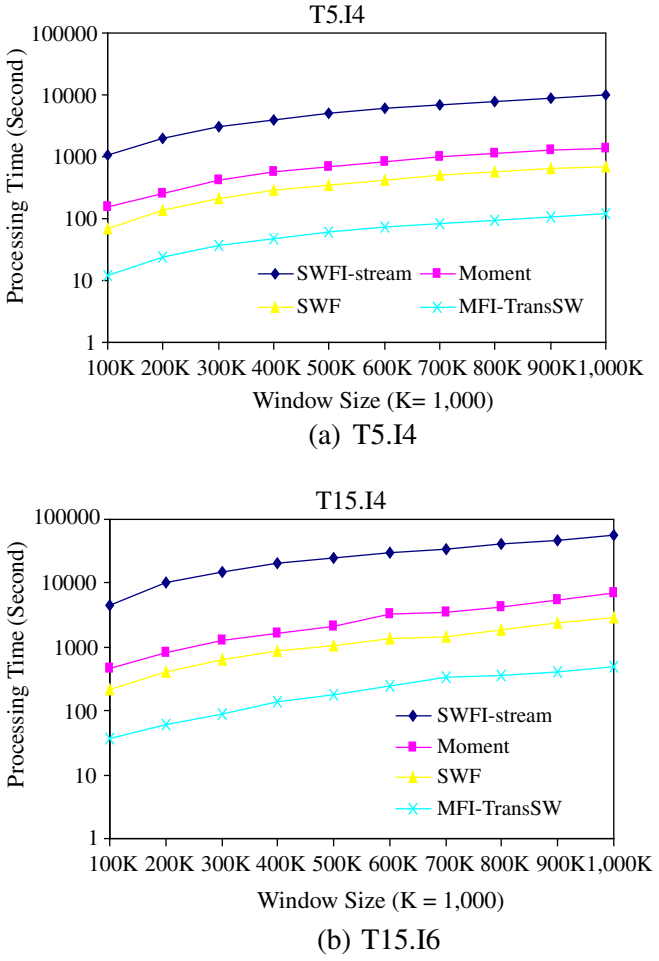


Fig. 11. Processing time of window sliding phase under different window sizes ($s = 0.1\%$).

mining frequent itemsets from data streams within a transaction-sensitive sliding window.

5.2. Performance evaluation of MFI-TimeSW algorithm

In this section, we report the experiments of the proposed MFI-TimeSW algorithm. The minimum support threshold s is 0.1%. The synthetic data stream consists of ten time units from TU_1 to TU_{10}, where $|TU_1| = 200$ K, $|TU_2| = 400$ K, $|TU_3| = 800$ K, $|TU_4| = 1000$ K, $|TU_5| = 1{,}000$ K, $|TU_6| = 200$ K, $|TU_7| = 500$ K, $|TU_8| = 1000$ K, $|TU_9| = 800$ K, and $|TU_{10}| = 800$ K. In order to simulate a time-sensitive sliding window of data streams, the size of the sliding window w is composed of five time units.

Fig. 13 shows the memory usage of phases 1–2 (window initialization phase + window sliding phase) and phases 1–2–3 (window initialization phase + window sliding phase + frequent itemsets generation phase) of MFI-TimeSW algorithm. As shown in Fig. 13, the memory usage of MFI-TimeSW is increased linearly as the window size increased.

Fig. 14 shows the processing time of phases 1–2 (window initialization phase + window sliding phase) and phases 1–2–3 (window initialization phase + window sliding phase + frequent itemsets generation phase) of MFI-TimeSW algorithm. As shown in Fig. 14, the processing time of phases 1 and 2 of MFI-TimeSW is increased linearly as the window size increased.

6. Conclusions

In this paper, we propose an efficient single-pass algorithm, called MFI-TransSW, for mining the set of frequent itemsets over data streams with a transaction-sensitive sliding window. An effective bit-sequence representation of items is developed to enhance the performance of MFI-TransSW. Based on MFI-TransSW algorithm, an efficient single-pass algorithm, called MFI-TimeSW, is presented to find the set of frequent itemsets over time-sensitive sliding windows. Experiments show that the proposed algorithms MFI-TransSW and MFI-TimeSW not

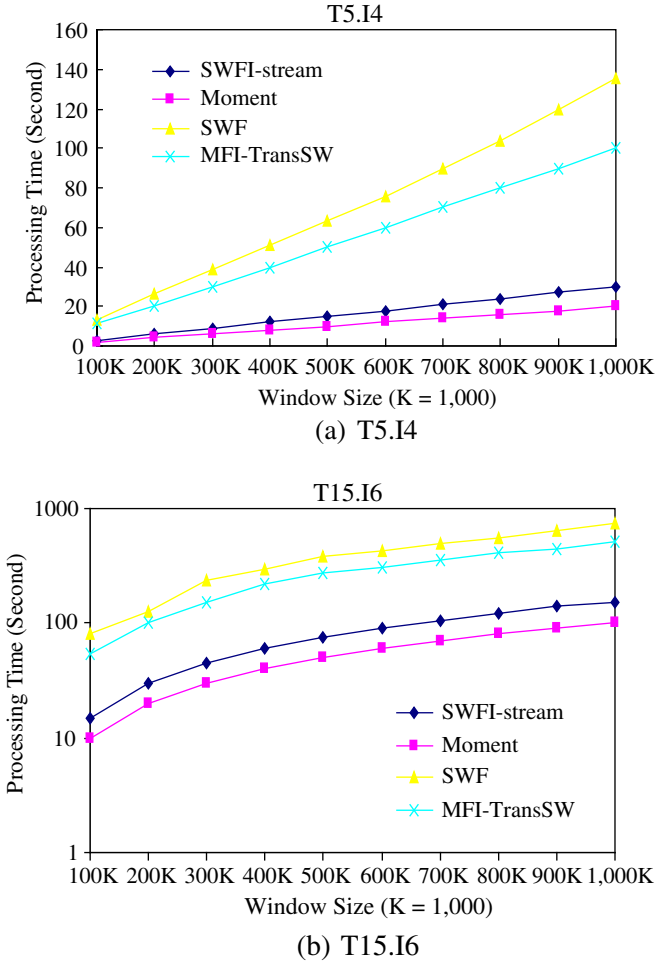


Fig. 12. Processing time of frequent itemset mining phase under different window sizes ($s = 0.1\%$).

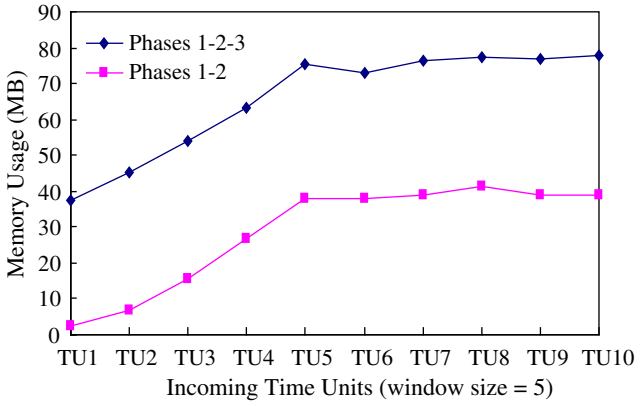


Fig. 13. Memory usages of MFI-TimeSW algorithm in different phases ($s = 0.1\%$, T5.I4).

only attain highly accurate mining results, but also run significant faster and consume less memory than do existing algorithms, such as SWF, Moment and SWFI-stream, for mining frequent itemsets from data streams within a sliding window.

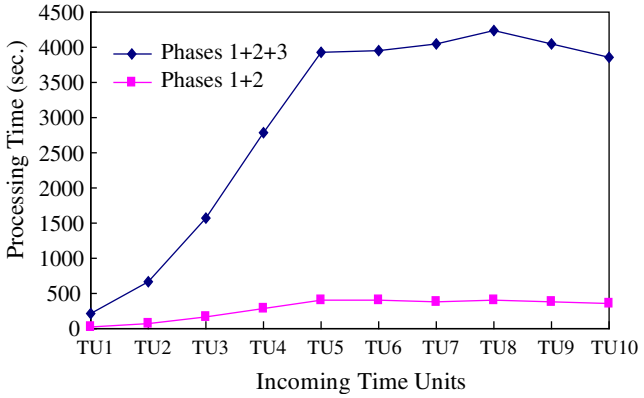


Fig. 14. Processing time of MFI-TimeSW algorithm in different phases ($s = 0.1\%$, T5.I4).

Acknowledgement

The authors are grateful to the anonymous referees whose valuable comments helped to improve the content of this paper. The research is supported in part by the National Science Council, Project No. NSC96-2218-E-424-001, Taiwan, Republic of China.

References

Aggarwal, C. C., Han, J., Wang, J., Yu, P. S. (2003). A framework for clustering evolving data streams. In *Prooceedings of the VLDB* (pp. 81–92).

Agrawal, R., Srikant, R. (1994). Fast algorithms for mining association rules. In *Prooceedings of the VLDB* (pp. 487–499).

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J. (2002). Models and issues in data stream systems. In *Prooceedings of the PODS* (pp. 1–16).

Babcock, B., Olston, C. (2003). Distributed top-k monitoring. In *Prooceedings of the ACM SIGMOD* (pp. 28–39).

Babu, S., & Widom, J. (2001). Continuous queries over data streams. *SIGMOD Record, 30*(3), 109–120.

Bandyopadhyay, S., Giannella, C., Maulik, U., Kargupta, H., Liu, K., & Datta, S. (2006). Clustering distributed data streams in peer-to-peer environments. *Information Sciences, 176*(14), 1952–1985.

Chang, J., Lee, W. (2003). Finding recent frequent itemsets adaptively over online data streams. In *Prooceedings of the ACM SIGKDD* (pp. 487–492).

Chang, J., & Lee, W. (2004). A sliding window method for finding recently frequent itemsets over online data streams. *Journal of Information Science and Engineering, 20*(4), 753–762.

Chen, Y., Dong, G., Han, J., Wah, B. W., Wang, J. (2002). Multi-dimensional regression analysis of time-series data streams. In *Prooceedings of the VLDB* (pp. 323–334).

Chen, G., Wu, X., Zhu, X. (2005). Sequential pattern mining in multiple data streams. In *Prooceedings of the ICDM* (pp. 585–588).

Chi, Y., Wang, H., Yu, P. S., & Muntz, R. R. (2006). Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems, 10*(3), 265–294.

Datar, M., Ginois, A., Indyk, P., Motwani, R. (2002). Maintaining stream statistics over sliding windows. In *Prooceedings of the SODA* (pp. 635–644).

Domingos, P., Hulten, G. (2000). Mining high-speed data streams. In *Prooceedings of the ACM SIGKDD* (pp. 71–80).

Dong, G., Han, J., Lakshmanan, L. V. S., Pei, J., Wang, H., Yu, P. S. (2003). Online mining of changes from data streams: Research problems and preliminary results. In *Prooceedings of the ACM SIGMOD MPDS*.

Giannella, C., Han, J., Pei, J., Yan, X., & Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, & Y. Yesha (Eds.), *Data mining: Next generation challenges and future directions*. AAAI/MIT.

Gibbons, P. B., Matias, Y. (1999). Synopsis data structures for massive data sets. In *Prooceedings of the SODA* (pp. 909–910).

Golab, L., & Özsu, M. T. (2003). Issues in data stream management. *SIGMOD Record, 32*(2), 5–14.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery, 8*(1), 53–87.

Henzinger, M. R., Raghavan, P., Rajagopalan, S. (1998). Computing data streams. Technical report 1998-011, Digital Equipment Corporation. Systems Research Center.

Ho, C.-C., Li, H.-F., Kuo, F.-F., Lee, S.-Y. (2006). Incremental mining of sequential patterns over a stream sliding window. In *Prooceedings of the IWMESD* (pp. 677–681).

Jiang, N., & Gruenwald, L. (2006). Research issues in data stream association rule mining. *SIGMOD Record, 35*(1).

Lee, C.-H., Lin, C.-R., & Chen, M.-S. (2005). Sliding window filtering: An efficient method for incremental mining on a time-variant database. *Information Systems, 30*, 227–244.

Li, H.-F., Lee, S.-Y., Shan, M.-K. (2004). An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Prooceedings of the IWKDDS*.

Li, H.-F., Lee, S.-Y., Shan, M.-K. (2005a). Online mining (recently) maximal frequent itemsets over data streams. In *Prooceedings of the IEEE RIDE*.

Li, H.-F., Lee, S.-Y., & Shan, M.-K. (2005b). Online mining changes of items over continuous append-only and dynamic data streams. *Journal of Universal Computer Science: Special Issue on Knowledge Discovery in Data Streams, 11*(8), 1411–1425.

Littau, D., & Boley, D. (2006). Streaming data reduction using low-memory factored representations. *Information Sciences, 176*(14), 2016–2041.

Manku, G. S., Motwani, R. (2002). Approximate frequency counts over data streams. In *Prooceedings of the VLDB* (pp. 346–357).

Yan, J., Zhang, B., Yan, S., Liu, N., Yang, Q., Cheng, Q., et al. (2006). A scalable supervised algorithm for dimensionality reduction on streaming data. *Information Sciences, 176*(14), 2042–2065.

Yu, J.-X., Chong, Z., Lu, H., Zhang, Z., & Zhou, A. (2006). A false negative approach to mining frequent itemsets from high speed transactional data streams. *Information Sciences, 176*(14), 1986–2015.

Zhu, Y., Shasha, D. (2002). StatStream: Statistical monitoring of thousands of data streams in real time. In *Prooceedings of the VLDB* (pp. 358–369).