Knowl Inf Syst (2008) 17:79–97
DOI 10.1007/s10115-007-0112-4

REGULAR PAPER

DSM-FI: an efficient algorithm for mining frequent itemsets in data streams

Hua-Fu Li · Man-Kwan Shan · Suh-Yin Lee

Received: 9 November 2005 / Revised: 7 October 2006 / Accepted: 1 September 2007 /
Published online: 9 January 2008

Abstract Online mining of data streams is an important data mining problem with broad applications. However, it is also a difficult problem since the streaming data possess some inherent characteristics. In this paper, we propose a new single-pass algorithm, called DSM-FI (data stream mining for frequent itemsets), for online incremental mining of frequent itemsets over a continuous stream of online transactions. According to the proposed algorithm, each transaction of the stream is projected into a set of sub-transactions, and these sub-transactions are inserted into a new in-memory summary data structure, called SFI-forest (summary frequent itemset forest) for maintaining the set of all frequent itemsets embedded in the transaction data stream generated so far. Finally, the set of all frequent itemsets is determined from the current SFI-forest. Theoretical analysis and experimental studies show that the proposed DSM-FI algorithm uses stable memory, makes only one pass over an online transactional data stream, and outperforms the existing algorithms of one-pass mining of frequent itemsets.

Keywords Data mining · Data streams · Frequent itemsets · Single-pass algorithm · Landmark window

1 Introduction

In recent years, database and knowledge discovery communities have focused on a new data model, in which data arrive in the form of *continuous streams*. It is often referred to as *data*

H.-F. Li (✉)
Department of Computer Science, Kainan University, Taoyuan, Taiwan
e-mail: hfli@mail.knu.edu.tw

M.-K. Shan
Department of Computer Science, National Chengchi University, Taipei, Taiwan
e-mail: mkshan@cs.nccu.edu.tw

S.-Y. Lee
Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan
e-mail: sylee@csie.nctu.edu.tw

streams or *streaming data*. Data streams possess some computational characteristics, such as unknown or unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived data elements (only one sequential pass over the data is permitted), and a lack of system control over the order in which that data arrive [3,10]. Many applications generate data streams in real time, such as sensor data generated from sensor networks, transaction flows in retail chains, Web record and click-streams in Web applications, performance measurement in network monitoring and traffic management, and call records in telecommunications.

Online mining of data streams differs from traditional mining of static datasets in the following aspects [10]. First, each data element in streaming data should be examined at most once. Second, the memory usage for mining data streams should be bounded even though new data elements are continuously generated from the stream. Third, each data element in the stream should be processed as fast as possible. Fourth, the analytical results generated by the online mining algorithms should be instantly available when requested by the users. Finally, the frequency errors of outputs generated by the online algorithms should be as small as possible. The online processing model of data streams is shown in Fig. 1.

As described earlier, the *continuous* nature of streaming data makes it essential to use the online algorithms which require only *one scan* over the data streams for knowledge discovery. The *unbounded* characteristic makes it impossible to store all the data into the 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 of streaming data. In other words, one-pass algorithms for mining data streams have to sacrifice the exactness of its analytical results by allowing some tolerable counting errors. Hence, traditional *multiple-pass* techniques studied for mining static datasets are not feasible to mine patterns over streaming data.

1.1 Related work

Frequent itemsets mining is one of the most important research issues in data mining. The problem of frequent itemsets mining of *static datasets* (not *streaming data*) was first introduced

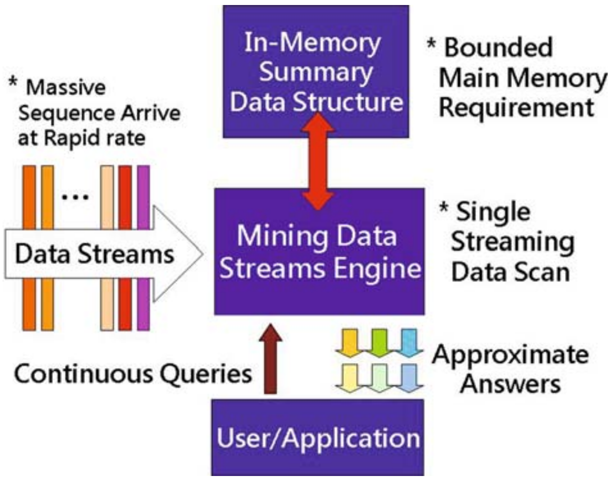
In-Memory Summary Data Structure
* Bounded Main Memory Requirement
* Massive Sequence Arrive at Rapid rate
Data Streams
Mining Data Streams Engine
* Single Streaming Data Scan
Continuous Queries
Approximate Answers
User/Application

Fig. 1 Processing model of data streams

by Agrawal et al. [1] described as follows. Let $\Psi = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. Let database *DB* be a set of transactions, where each transaction *T* contains a set of items, such that $T \subseteq \Psi$. The *size* of database *DB* is the total number of transactions in *DB* and is denoted by $|DB|$. A set of items is referred to as an *itemset*. An itemset *X* with *l* items is denoted by $X = (x_1x_2, \ldots, x_l)$, such that $X \subseteq \Psi$. The *support* of an itemset *X* is the number of transactions in *DB* containing the itemset *X* as a subset, and denoted by *sup*(*X*). An itemset *X* is *frequent* if $sup(X) \geq minsup \cdot |DB|$, where *minsup* is a user-specified minimum support threshold in the range of [0, 1]. Consequently, given a database *DB* and a user-defined minimum support threshold *minsup*, the problem of mining frequent itemsets in *static datasets* is to find the set of all itemsets whose support is no less than $minsup \cdot |DB|$. In this paper, we will focus on the problem of mining frequent itemsets in *data streams*.

Many previous studies contributed to the efficient mining of frequent itemsets in streaming data [4–9,12–17]. According to the stream processing model [18], the research of mining frequent itemsets in data streams can be divided into three categories: *landmark windows* [14], *sliding windows* [5,7,13,15,16], and *damped windows* [4,9]. In the landmark windows model, knowledge discovery is performed based on the values between a specific timestamp called *landmark* and the present. In the sliding windows model, knowledge discovery is performed over a fixed number of recently generated data elements which is the target of data mining. In the damped windows model, recent sliding windows are more important than previous ones. In other words, older transactions contribute less toward itemset frequencies.

In [14], Manku and Motwani developed two single-pass algorithms, sticky-sampling and lossy counting, to mine frequent items over landmark windows. Moreover, Manku and Motwani proposed a lossy-counting based three module method, called *BTS* (Buffer-Trie-SetGen), for mining the set of frequent itemsets (FI) from streaming data. Chang and Lee [5] proposed a BTS-based algorithm for mining frequent itemsets in sliding windows model. Moreover, Chang and Lee [4] also developed another algorithm, called estDec, for mining frequent itemsets in streaming data in which each transaction has a weight decreasing with age. Teng et al. [15] proposed a regression-based algorithm, called FTP-DS, to find frequent itemsets across multiple data streams in a sliding window. Lin et al. [13] proposed an incremental mining algorithm to find the set of frequent itemsets in a time-sensitive sliding window. Giannella et al. [8] proposed a frequent pattern tree (abbreviated as FP-tree [11]) based algorithm, called FP-stream, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique. Yu et al. [17] discussed the issues of false negative or false positive in mining frequent itemsets from high speed transactional data streams. Wong and Fu [16] proposed an efficient algorithm to mine top-*k* frequent itemset in a stream sliding window without a user-defined minimum support constraint. Jin and Agrawal [12] proposed an algorithm, called StreamMining, for in-core frequent itemset mining over data streams. StreamMining is based on the BTS algorithm. Chi et al. [7] proposed an algorithm, called MOMENT, which might be the first to find *closed frequent itemsets* from data streams. A lattice-based summary data structure, called CET, is used in the MOMENT algorithm to maintain the information of closed frequent itemsets.

1.2 Our contributions

Because the focus of the paper is on frequent itemsets mining over data streams with a landmark window, we mainly address it by comparison with the algorithms BTS [14] and StreamMining [12].

In the BTS algorithm, two estimated parameters: *minimum support threshold* s, and *maximum support error threshold* ε, are used, where $0 < \varepsilon \le s < 1$. The incoming data stream is conceptually divided into buckets of width $w = \lceil 1/\varepsilon \rceil$ transactions each, and the current length of the stream is denoted by N transactions. The BTS algorithm is composed of three steps. In the first step, BTS repeatedly reads a *batch* of buckets into main memory. In the second step, it decomposes each transaction within the current bucket into a set of itemsets, and stores these itemsets into a summary data structure D which contains a set of entries of the form $(e, e.freq, e.\Delta)$, where e is an itemset, $e.freq$ is an approximate *freq*uency of the itemset e, and $e.\Delta$ is the maximum possible error in $e.freq$. For each itemset e extracted from the incoming transaction T, BTS performs two operations to maintain the summary data structure D. First, it counts the occurrences of e in the current batch, and updates the value $e.freq$ if the itemset e already exists in the structure D. Second, BTS creates a new entry $(e, e.freq, e.\Delta)$ in D, if the itemset e does not occur in D, but its estimated frequency *e.freq* in the batch is greater than or equal to $|batch| \cdot \varepsilon$, where the value of maximal possible error $e.\Delta$ is set to $\lfloor |batch| \cdot \varepsilon \rfloor$, and $|batch|$ denotes the total number of transactions in the current batch. To bound the space requirement of D, BTS algorithm deletes the updated entry e if $e.freq + e.\Delta \le |batch| \cdot \varepsilon$. Finally, BTS outputs those entries e_i in D, where $e_i.freq \ge (s - \varepsilon) \cdot N$, when a user requests a list of itemsets with the minimum support threshold s and the support error threshold ε.

StreamMining algorithm [12] is an in-core frequent itemset mining algorithm based on the BTS algorithm. StreamMining uses a new approach (derived from the problem of finding a majority element) to reduce the memory requirements for determining the frequent itemsets. Then, StreamMining uses such a reduced set of frequent 2-itemsets and the a priori property to reduce the number of i-itemsets, for $i > 2$, and establishes a bound on false positives.

The motivation of the study is to develop a method that utilizes some space-effective summary data structures (such as FP-tree [11] developed for frequent itemsets mining of a static dataset) to reduce the cost in mining frequent itemsets over data streams. In this paper, an efficient single-pass algorithm, referred to as *Data Stream Mining for Frequent Itemsets* (abbreviated as DSM-FI), is proposed to improve the efficiency of frequent itemset mining in data streams. A new summary data structure called *summary frequent itemset forest* (abbreviated as SFI-forest) is developed for online incremental maintenance of the essential information about the set of all frequent itemsets of data streams generated so far. The proposed algorithm has three important features: a single pass of streaming data for counting the support of itemsets; an extended prefix tree-based, compact pattern representation of summary data structure; and an effective and efficient search and determination mechanism of frequent itemsets. Moreover, the *frequency error guarantees* provided by DSM-FI algorithm is the same as that of BTS algorithm. The error guarantees are stated as follows. First, all itemsets whose true support exceeds $s \cdot N$ are output. Second, no itemsets whose true support is less than $(s - \varepsilon) \cdot N$ is output. Finally, estimated supports of itemsets are less than the true support by at most $\varepsilon \cdot N$ [12]. The comprehensive experiments show that our algorithm is efficient on both sparse and dense datasets. Furthermore, DSM-FI algorithm outperforms the algorithms BTS and StreamMining, by one order of magnitude for discovering the set of all frequent itemsets over the entire history of the data streams.

1.3 Roadmap

The remainder of the paper is organized as follows. Section 2 defines the problem of single-pass mining frequent itemsets in landmark windows over data streams. The proposed DSM-FI algorithm is described in Sect. 3. The extended prefix tree-based summary data structure

SFI-forest is introduced to maintain the essential information about the set of all frequent itemsets of the stream generated so far. Theoretical analysis and experiments are presented in Sect. 4. Section. 5 remarks on future work, and concludes the work.

2 Problem definition

Based on the estimation mechanism of the BTS algorithm, we propose a new, single-pass algorithm to improve the efficiency of mining frequent itemsets over the entire history of data streams when a user-specified minimum support threshold $s \in (0, 1)$, and a maximum support error threshold $\varepsilon \in (0, s)$ are given.

Let $\Psi = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. An *itemset* is a nonempty set of items. A l-itemset, denoted by $(x_1 x_2, \ldots, x_l)$, is an itemset with l items. A transaction T consists of a unique transaction identifier (tid) and a set of items, and denoted by $<$ tid, $(x_1 x_2, \ldots, x_q) >$, where $x_i \in \Psi, \forall i = 1, 2, \ldots, q$. A *basic window* W consists of k transactions. The basic windows are labeled with window identifier *wid*, starting from 1.

Definition 1 A *data stream*, DS= $[W_1, W_2, \ldots, W_N)$, is an infinite sequence of basic windows, where N is the window identifier of the " *latest*" basic window. The *current length* of *DS*, written as DS.CL, is $k \cdot N$, i.e., $|W_1| + |W_2| + \cdots + |W_N|$. The windows arrive in some order (implicitly by arrival time or explicitly by timestamp), and may be seen only once.

Online mining of frequent itemsets in a landmark window of data streams is to mine the set of all frequent itemsets from the transactions between a specified window identifier, called *landmark*, and the current window identifier N. Note that the value of landmark is set to 1 in this paper.

To ensure the completeness of frequent itemsets for data streams, it is necessary to store not only the information related to frequent itemsets, but also that related to infrequent ones. If the information about the currently infrequent itemsets were not stored, such information would be lost. If these itemsets become frequent later on, it would be impossible to figure out their correct support and their relationship with other itemsets [9]. The data stream mining algorithms have to sacrifice the exactness of the analytical results by allowing some tolerable support errors since it is unrealistic to store all the streaming data into the limited main memory. Hence, we define two types of *support* (or *occurrence frequency*) of an itemset, and divide the itemsets embedded in the stream into three categories: *frequent itemsets*, *semi-frequent itemsets*, and *infrequent itemsets*.

Definition 2 The *true support* of an itemset X, denoted by $X.tsup$, is the number of transactions in the data stream containing the itemset X as a subset. The *estimated support* of an itemset X, denoted by $X.esup$, is the estimated true support of X stored in the summary data structure, where $0 < X.esup \leq X.tsup$.

Definition 3 The *current length* (*CL*) of data stream with respect to an itemset X stored in the summary data structure, denoted by $X.CL$, is $(N - j + 1) \cdot k$, i.e., $|W_j| + |W_{j+1}| + \cdots + |W_N|$, where W_j is the first basic window stored in the current summary data structure containing the itemset X.

Definition 4 An itemset X is *frequent* if $X.tsup \geq s \cdot X.CL$. An itemset X is *semi-frequent* if $s \cdot X.CL > X.tsup \geq \varepsilon \cdot X.CL$. An itemset X is *infrequent* if $\varepsilon \cdot X.CL > X.tsup$.

Definition 5 A frequent itemset is *maximal* if it is not a subset of any other frequent itemsets generated so far.

Therefore, given a continuous data stream DS $= [W_1, W_2, \ldots, W_N)$, a user-defined minimum support threshold s in the range of [0, 1], and a user-specified maximum support error threshold ε in the range of $[0, s]$, the problem of online mining of frequent itemsets in a landmark window over data streams is to find the set of all frequent itemsets by one scan of the streaming data.

3 The proposed DSM-FI algorithm

In this section, we describe the proposed algorithm *DSM-FI* (data stream mining for frequent itemsets) for online mining of frequent itemsets in a landmark window of a continuous data stream. The DSM-FI algorithm consists of four steps.

(a) Step 1: the proposed DSM-FI algorithm reads a basic window of transactions from the buffer in main memory, and sorts the items of transaction in *lexicographical order*.
(b) Step 2: DSM-FI algorithm constructs and maintains an in-memory prefix-tree based summary data structure, called SFI-forest (summary frequent itemset forest).
(c) Step 3: DSM-FI algorithm prunes the infrequent information from the current SFI-forest.
(d) Step 4: DSM-FI finds the frequent itemsets from the current SFI-forest.

Steps 1 and 2 are performed in sequence for a new incoming basic window. Step 3 is performed after every basic window has been processed. Finally, step 4 is usually performed periodically or when it is needed. Since the reading of a basic window of transactions from the buffer in main-memory is straightforward, we shall henceforth focus on Steps 2 (discussed in Sect. 3.1), 3 (discussed in Sect. 3.2), and 4 (discussed in Sect. 3.3), and devise new methods for effective construction and maintenance of summary data structure, and efficient determination of frequent itemsets.

Before discussing the proposed DSM-FI algorithm, we use an example to illustrate the construction of the summary data structure SFI-forest.

Example 1 Assume that the current basic window W_j contains six transactions: $<acdf>$, $<abe>$, $<df>$, $<cef>$, $<acdef>$ and $<cef>$, where a, b, c, d, e and f are items in the data stream. The SFI-forest with respect to the first two transactions, $<acdf>$ and $<abe>$, constructed by DSM-FI algorithm is described as follows. Note that each node of the form (id: $id.\ esup$: $id.\ wid$) consists of three fields: *item-id*, *estimated support*, and *window-id*. For example, $(a : 2 : j)$ indicates that, from basic window W_j to current basic window $W_N (1 \leq j \leq N)$, item a appeared twice.

(a) First transaction $<acdf>$: First of all, the proposed DSM-FI algorithm reads the first transaction and performs the *transaction projection* (TP) on the first transaction $<acdf>$, namely, TP($<acdf>$). After performing TP($<acdf>$), DSM-FI generates a set of sub-transactions, $<acdf>$, $<cdf>$, $<df>$, and $<f>$, to record the essential information about the set of potential frequent itemsets of the first transaction. These sub-transactions are also called *itemset-suffix transactions* of item a. Then, DSM-FI inserts these itemset-suffix transactions: $<acdf>$, $<cdf>$, $<df>$, and $<f>$ of item a into the proposed summary data structure SFI-forest. The SFI-forest consists of three parts: *a list of frequent items* (FI-list), *a list of summary frequent trees of frequent items* (SFI-trees), and *a list of opposite frequent items* (OFI-list). Hence, DSM-FI inserts these itemset-suffix transactions into FI-list, [a.SFI-tree, a.OFI-list], [c.SFI-tree, c.OFI-list], [d.SFI-tree, d.OFI-list], and [f.SFI-tree, f.OFI-list], respectively. The result is shown in Fig. 2. Each item in the FI-list has a SFI-tree and an OFI-list. The *head-links* of items

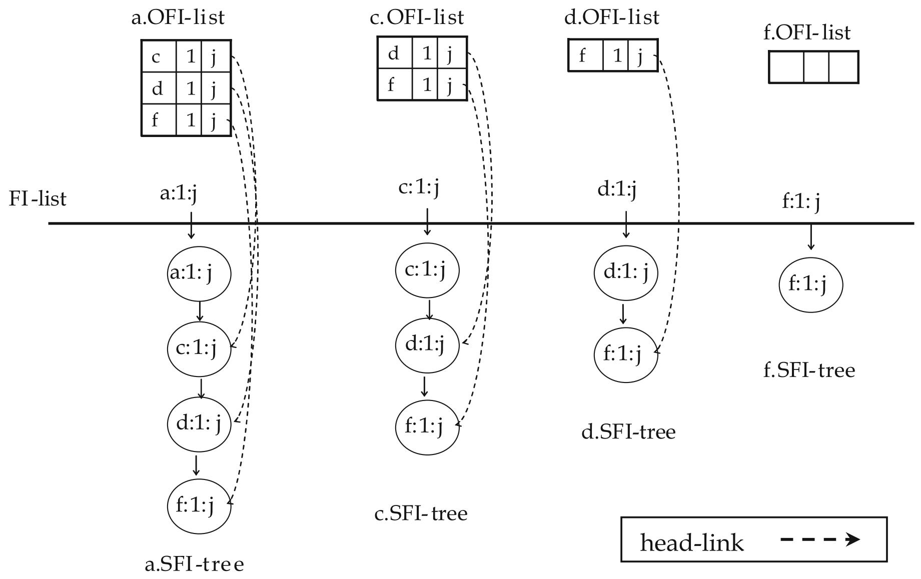

Fig. 2 SFI-forest construction after processing the first transaction $<acdf>$ of window W_j

in each OFI-list is used to record the occurrence order in each transaction. But, in the following steps, the head-links of each OFI-list are omitted for concise presentation.

(b) Second transaction $<abe>$: DSM-FI reads the second transaction and calls the TP ($<abe>$). Next, DSM-FI inserts three item-suffix transactions: $<abe>$, $<be>$, and $<e>$ into the FI-list, [a.SFI-tree, a.OFI-list], [b.SFI-tree, b.OFI-list], and [e.SFI-tree, e.OFI-list], respectively. The result is shown in Fig. 3. After processing all the transactions of window W_j, the SFI-forest generated so far is shown in Fig. 4.

3.1 Effective construction and maintenance of summary data structure

In this section, we describe the method which constructs and maintains the proposed in-memory prefix-tree based summary data structure.

Definition 6 A *summary frequent itemset forest* (*SFI-forest*) is an extended prefix-tree based summary data structure defined below.

1. *SFI-forest* consists of a *FI-list* (a list of Frequent Items) with k items ($e_1, e_2, \ldots, e_i, \ldots, e_k, k \geq 1$), and a set of e_i*.SFI-trees* (summary frequent itemset trees of items).
2. Each entry e in the *FI-list* consists of four fields: $\boldsymbol{e}$, $\boldsymbol{e.esup}$,$\boldsymbol{e.window\text{-}id}$, and $\boldsymbol{e.head\text{-}link}$. The field e registers which item identifier the entry represents, *e.esup* records the number of transactions in the stream so far containing the item e, the value of *e.window-id* assigned to a new entry is the window identifier of current window, and *e.head-link* points to the root node of the e.SFI-tree. Note that each entry in the FI-list is a *root node* of the e.SFI-tree .
3. Each node in the *e.SFI-tree* consists of four fields: $\boldsymbol{e'}$,$\boldsymbol{e'.esup}$, $\boldsymbol{e'.window\text{-}id}$, *and*$\boldsymbol{e'.node\text{-}link}$. The first field e' is the item identifier of the item being inserted. The second field *e'.esup* registers the number of transactions represented by a portion of the path reaching the node with the item-id e'. The value of the third field *e'.window-id*

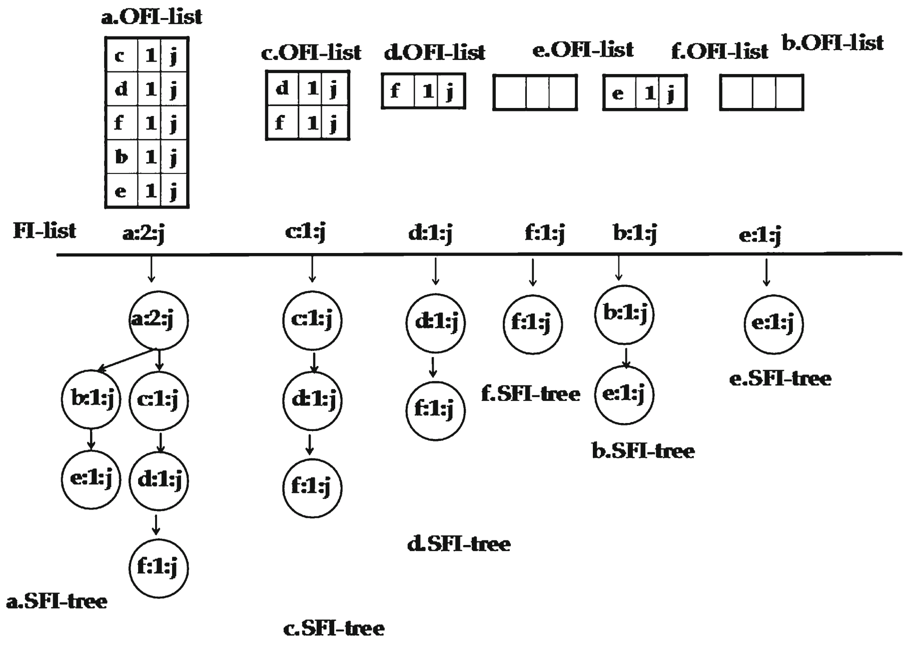

Fig. 3 SFI-forest construction after processing the second transaction <*abe*> of window W_j

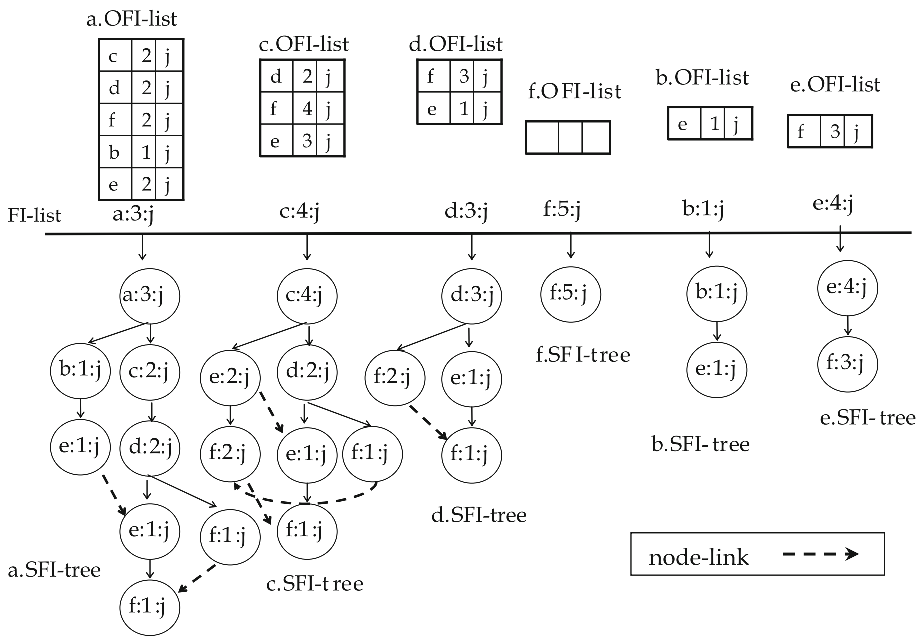

Fig. 4 SFI-forest construction after processing the window W_j

assigned to a new node is the window identifier of the current window. The fourth field *e'.node-link* links up a node with the next node with the same item-id e' in the same SFI-tree or null if there is none.

4. Each *e.SFI-tree* has a specific *OFI-list* (a list of Opposite Frequent Items) with k' items, denoted by $\boldsymbol{e.OFI-list}$. Each item x in the *e.OFI-list* consists of four fields: $\boldsymbol{x}$, $\boldsymbol{x.esup}$, $\boldsymbol{x.window\text{-}id}$, and $\boldsymbol{x.head\text{-}link}$. The *e.OFI-list* is similar to the *FI-list* except that the field *head-link* links to the first node with the item-id in the *e.SFI-tree*. Note that $|e.OFI-list| = |FI\text{-}list|$ in the worst case, where $|FI\text{-}list|$ denotes the total number of entries in the FI-list.

The construction process of SFI-forest is described as follows. First, DSM-FI algorithm reads a transaction T with m items ($m \geq 1$) from the current window W_N for SFI-forest construction. At this time, DSM-FI projects the transaction T into m sub-transactions, and inserts the m sub-transactions into the SFI-forest. The detail of the effective projection is described as follows. A transaction T with m items, i.e., $(e_1e_2, \ldots, e_m)$, should be projected into m sub-transactions; that is, $(e_1e_2, \ldots, e_m)$, $(e_2e_3, \ldots, e_m)$, $\ldots$, $(e_{m-1}e_m)$, and (e_m). These m sub-transactions are called *itemset-suffix transactions*, since the first item of each sub-transaction is an *itemset-suffix* of the original transaction T. This step, called ***transaction projection***(*TP*), is denoted by $TP(T) = \{e_1|T, e_2|T, \ldots, e_i|T, \ldots, e_m|T\}$, where $e_i|T = (e_ie_{i+1}, \ldots, e_m)$, $\forall i = 1, 2, \ldots, m$. The *projecting cost* of a transaction T with m items for constructing the SFI-forest is $O(m^2)$.

After performing the transaction projection of transaction T, two operations of DSM-FI are executed. First, DSM-FI inserts the items $e_1, e_2, \ldots, e_m$ of T into the *FI-list*, and then removes T from the current window W_N. Second, the items of these itemset-suffix transactions are inserted into the e_i. *SFI-trees* ($\forall i, i = 1, 2, \ldots, m$) as branches, and the estimated support of the corresponding e_i. *OFI-lists* are updated. If an itemset share a prefix with an itemset already in the SFI-tree, the new itemset will share a prefix of the branch representing that itemset. In addition, an estimated support counter is associated with each node in the tree. The counter is updated when an itemset-suffix transaction causes the insertion of a new branch. Figure. 5 outlines the algorithms of SFI-forest construction in the DSM-FI algorithm and Fig. 6 shows the subroutines of SFI-forest construction and maintenance.

In the next section, we describe the steps of pruning infrequent information of DSM-FI algorithm.

3.2 Pruning infrequent information from the current SFI-forest

According to the a priori property, only the frequent 1-itemsets are used to construct candidate k-itemsets, where $k \geq 2$. Thus, the set of candidate itemsets containing the infrequent 1-itemsets stored in the summary data structure SFI-forest is pruned. The pruning is usually performed periodically or when it is needed.

Let the maximum support error threshold be ε in the range of $[0, s]$, where s is a user-defined minimum support threshold in the range of $[0, 1]$. The summary data structure pruning mechanism of DSM-FI algorithm is that the item x and its supersets are deleted from SFI-forest if $x.esup < \varepsilon \cdot x.CL$. For each entry ($x$, *x.esup*, *x.window-id*, *x.head-link*) in the FI-list, if its *x.esup* is less than $\varepsilon \cdot x.CL$, it can be regarded as an *infrequent item*. At this time, three operations are performed in sequence. First, the proposed DSM-FI algorithm deletes the x.OFI-list, x.SFI-tree, and the infrequent entry x from the FI-list. Second, DSM-FI removes the infrequent item x of other OFI-lists by traversing the FI-list. Third, DSM-FI deletes the infrequent item x from other SFI-trees, and reconstructs these SFI-trees.

Algorithm 1 (SFI-forest construction)

Input: A data stream, $DS = [W_1, W_2, \ldots, W_N)$ with landmark 1, a user-specified minimum support threshold $s \in (0, 1)$, and a maximum support error threshold $\varepsilon \in (0, s)$.

Output: A SFI-forest generated so far.

```
1:  FI-list = {};   /*initialize the FI-list to empty.*/
2:  foreach window $W_j$ do /* $j = 1, 2, \ldots, N$ */
3:     foreach transaction $T = (x_1x_2\ldots x_m) \in W_j$ ($j = 1, 2, \ldots, N$) do
                                  /* $m \geq 1$ and $j$ is the current window identifier */
4:           foreach item $x_i \in T$ do   /* the maintenance of FI-list */
5:                 if $x_i \notin$ FI-list then
6:                    create a new entry of form ($x_i$, 1, $j$, head-link) into the FI-list;
                     /* the entry form is (item-id, item-id.esup, window-id, head-link)*/
7:                 else /* the entry already exists in the FI-list*/
8:                       $x_i$.esup = $x_i$.esup + 1;
                     /* increment the estimated support of item-id $x_i$ by one*/
9:                 end if
10:            end for
11:       call TP($T$, $j$);
        /* project the transaction with each itemset-suffix $x_i$ for constructing the $x_i$.SFI-tree */
12:    end for
13:    call SFI-forest-pruning(SFI-forest, $\varepsilon$, $N$);   /* Step 3 of DSM-FI algorithm */
14: end for
```

Fig. 5 Algorithm of SFI-forest construction

After pruning all infrequent items from SFI-forest, SFI-forest contains the set of all frequent itemsets and semi-frequent itemsets of the data stream generated so far. Now, we use an example to illustrate the pruning operation of DSM-FI algorithm.

Example 2 Let the maximum support error threshold ε be 0.2. Hence, an itemset X is *infrequent* in Fig. 6 if X.esup $< \varepsilon \cdot X$.CL. Note that $\varepsilon \cdot X.\text{CL} = 0.2 \cdot 6 = 1.2$. After computing the current window W_j, the next step of DSM-FI is to prune all the infrequent items from the current SFI-forest. At this time, DSM-FI deletes the b.SFI-tree, b.OFI-list, and item b itself from the FI-list, since item b is an infrequent item; that is, $b.\text{esup} = 1 < 1.2$. Then, DSM-FI reconstructs the a.OFI-list and a.SFI-tree, because a.OFI-list and a.SFI-tree contains the infrequent item b. The result is shown in Fig. 7.

The next step of DSM-FI is to determine the set of all frequent itemsets from SFI-forest constructed so far. The step is performed only when the analytical results of the data stream is requested. Note that the number of candidate 2-itemsets is a performance bottleneck in the a priori-based frequent itemset mining algorithms [11]. The proposed DSM-FI algorithm can avoid the performance problem, because DSM-FI can generate the set of all frequent 2-itemsets immediately by combining the frequent items in the FI-list with the frequent items in their corresponding OFI-lists.

3.3 Determining frequent itemsets from the summary data structure

Once the SFI-forest is constructed and maintained, we can derive the set of all frequent itemsets by traversing the SFI-forest according to the a priori principle. We propose an

```
Subroutine TP   /* Step 2 of DSM-FI algorithm */
Input: A transaction $T = (x_1x_2\ldots x_m)$ and the current window-id $j$;
Output: $x_i$.SFI-tree, $\forall i = 1, 2, \ldots, m$;

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

Subroutine SFI-tree-maintenance   /* Step 2 of DSM-FI algorithm   */
Input: An itemset-suffix transaction $(x_ix_{i+1}\ldots x_m)$, the current window-id $j$, and $x_i$.SFI-tree, where
    $i$=1, 2, …, $m$;
Output: A modified $x_i$.SFI-tree, where $i$=1, 2, ..., $m$;

1:   foreach item $x_l$ do /* $l = i+1, i+2, \ldots, m$ */
2:        if $x_l \notin x_i$.OFI-list then    /*  $x_i$.OFI-list maintenance   */
3:             create a new entry of form ($x_l$, 1, $j$, head-link) into the $x_i$.OFI-list;
               /* the entry form is (item-id, item-id.esup, item-id.window-id, item-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 $x_i$, $\forall i = 1, 2, \ldots, m$, do /* $x_i$.SFI-tree maintenance */
9:        if SFI-tree has a child node with item-id $y$ such that $y.item\text{-}id = x_i.item\text{-}id$ then
10:            $y.esup = y.esup + 1$; /*increment $y$'s estimated support by one*/
11:        else create a new node of the form ($x_i$, 1, $j$, node-link);
   /* initialize the estimated support of the new node to one, 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-FI algorithm   */
Input: A SFI-forest, a user-specified maximum support error threshold $\varepsilon$, and the current window
identifier $N$;
Output: A SFI-forest which contains the set of all semi-frequent and frequent itemsets.

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

Fig. 6 Subroutines of SFI-forest construction algorithm

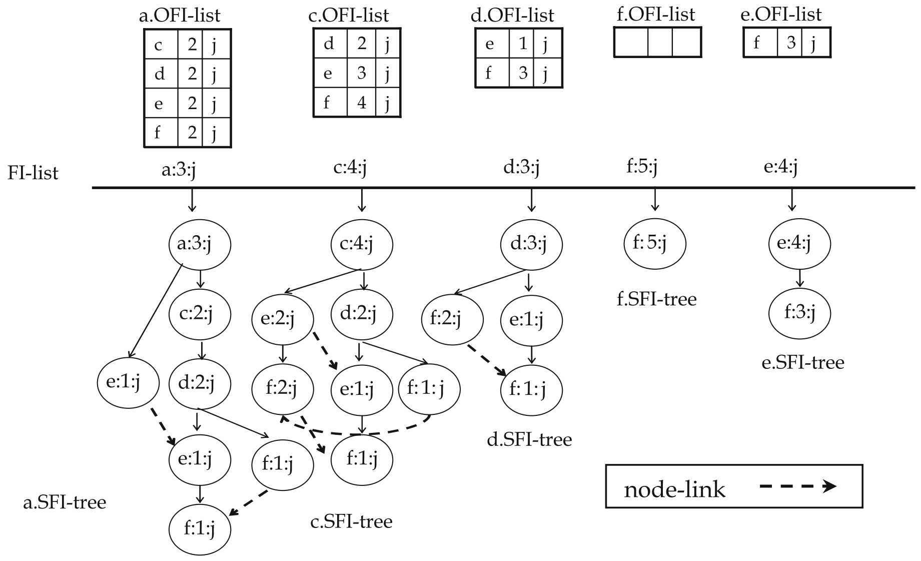

Fig. 7 SFI-forest after pruning all infrequent items

efficient mechanism called *top-down Frequent Itemset Selection* (**todoFIS**), as shown in Fig. 8, for mining frequent itemsets. It is especially useful in mining long frequent itemsets. The method is described as follows.

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

First, DSM-FI starts with a specific frequent item $e_i.o_l (1 \leq l \leq j)$, whose estimated support is smallest, and traverses the paths containing $e_i.o_l$ via node-links of e_i.SFI-tree to count the estimated support of the candidate $(e_i e_i.o_1 e_i.o_2 \ldots e_i.o_j)$. If the estimated support of the candidate is greater than or equal to $(s - \varepsilon) \cdot e_i.CL$, then it is a frequent itemset. All subsets of this frequent itemset are also frequent itemsets according to the a priori property.[1] Hence, the complete set of the frequent itemsets stored in the e_i.SFI-tree can be generated by enumeration of all the combinations of the subsets of frequent $(j+1)$-itemset, $(e_i e_i.o_1 e_i.o_2, \ldots, e_i.o_j)$.

On the other hand, if the estimated support of the candidate $(j+1)$-itemset is less than the threshold $(s-\varepsilon) \cdot e_i.CL$, it is not a frequent itemset. Now, we need to use the same mechanism to test all the subsets of the $(j+1)$-itemset until the candidate 3-itemsets. This is because all frequent 2-itemsets can be generated by combining the item e_i and the frequent items of the e_i.OFI-list. Note that a $(j+1)$-itemset can be decomposed into $C(j+1, j)$ j-itemsets. We decompose one candidate j-itemset from the $(j+1)$-itemset at a time, and use the same scheme described above to count the estimated support of this candidate j-itemset. Finally,

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

```
Algorithm 2 (todoFIS: top-down Frequent Itemset Selection)
Input: A current SFI-forest, the current window identifier $N$, a minimum support threshold $s$, and a
       maximum support error threshold $\varepsilon$.
Output: A set of all frequent itemsets.
1:  MFI_temp-list = ∅;
     /* MFI_temp-list is a temporary list used to store the set of maximal frequent itemsets */
2:  foreach entry $e$ in the current FI-list do
3:          construct a maximal candidate itemset $E$ with size $|E|$   /* $|E| = 1+|e$.OFI-list$|$ */
4:          count $E.esup$ by traversing the $e$.SFI-tree;
5:          if $E.esup \geq (s-\varepsilon)\cdot N$ then
6:             if $E \not\subset$ MFI_temp-list and $E$ is not a subset of any other patterns in the MFI_temp-list
               then
7:                  add $E$ into the MFI_temp-list;
8:                  remove $E$'s subsets from the MFI_temp-list;
9:             end if
10:         else /* if $E$ is not a frequent itemset */
11:              enumerate $E$ into itemsets with size $|E|-1$;
12:         end if
13:    until todoFIS finds the set of all frequent itemsets with respect to entry $e$;
14: end for
```

Fig. 8 Algorithm description of todoFIS

all the maximal frequent itemsets are maintained in a temporal MFI-list, called MFI$_{\text{temp}}$-list, for efficient generation of the set of all frequent itemsets. If such the MFI$_{\text{temp}}$-list is obtained, all the frequent itemsets can be generated efficiently by enumerating the set of all maximal frequent itemsets in the current MFI$_{\text{temp}}$-list without any candidate generation and support counting. Note that if the user request is just to find the set of all *maximal frequent itemsets* generated so far, DSM-FI outputs all maximal frequent itemsets efficiently by scanning the MFI$_{\text{temp}}$-list.

Example 3 Let the minimum support threshold s be 0.5. Therefore, an itemset X is *frequent* in Fig. 7 if X.esup $\geq s \cdot X$.CL. Note that $s \cdot X.\text{CL} = 0.5 \cdot 6 = 3$ in this example. The online mining steps of frequent itemsets of DSM-FI are described as follows.

1. First of all, DSM-FI starts the frequent itemset mining scheme from the first frequent item a (from left to right). At this moment, only item a is a frequent itemset, since the estimated support of items c, d, e, and f in the a.OFI-list are less than $s \cdot a$.CL, where $s \cdot a.\text{CL} = 3$. Now, DSM-FI stores the maximal frequent 1-itemset (a) into the MFI$_{\text{temp}}$-list.
2. Next, DSM-FI starts on the second entry c for frequent itemset mining. DSM-FI generates a candidate maximal 3-itemset (*cef*), and traverses the c.SFI-tree to count its estimated support. As a result, the candidate (*cef*) is a maximal frequent itemset, since its estimated support is 3 and it is not a subset of any other frequent itemsets in the MFI$_{\text{temp}}$-list. Now, DSM-FI stores the maximal frequent itemset (*cef*) into the MFI$_{\text{temp}}$-list.
3. Next, DSM-FI starts on the third entry d and generates a candidate maximal 2-itemset (*df*). DSM-FI stores the itemset (*df*) into the MFI$_{\text{temp}}$-list without traversing d.SFI-tree because (*df*) is a frequent 2-itemset and is not a subset of any other maximal frequent itemsets stored in the MFI$_{\text{temp}}$-list.

4. On the fourth entry f, DSM-FI algorithm generates one frequent 1-itemset (f) directly, since the f.OFI-list is empty. DSM-FI does not store it into the MFI_{temp}-list, because (f) is a subset of a generated maximal frequent itemset (*cef*).
5. Finally, on the fifth entry e, DSM-FI generates a frequent 2-itemset (*ef*) directly. However, the frequent 2-itemset (*ef*) is a subset of a maximal frequent itemset (*cef*) stored in the MFI_{temp}-list. DSM-FI algorithm does not store it into the MFI_{temp}-list.

After processing all the entries in the FI-list, the MFI_{temp}-list generated by DSM-FI algorithm contains the set of current maximal frequent iemsets: $\{(a), (cef), (df)\}$. Therefore, the set of all frequent itemsets can be generated by enumerating the set: $\{(a), (cef), (df)\}$. Consequently, the set of all frequent itemsets in Fig. 7 are $\{(a), (cef), (ce), (cf), (ef), (c), (e), (f), (df), (d)\}$.

3.4 Theoretical analysis

In this section, we discuss the maximal estimated support error of frequent itemsets generated by DSM-FI algorithm, the space upper bound of the prefix-tree-based summary data structure, and the differences between the proposed SFI-forest and the FP-tree.

3.4.1 Maximal estimated support error analysis

In this section, we discuss the maximal estimated support error of all frequent itemsets generated by DSM-FI algorithm. Let $X.\ wid$ is the *window-id* of itemset X stored in the current SFI-forest. Let the window contains k transactions. Let the maximum support error threshold be ε. Let the current *window-id* of the incoming stream be *wid*(N). Now, we have the following theorem of *maximal estimated support error guarantee* of frequent itemsets generated by the proposed algorithm.

Theorem 1 $X.tsup - X.esup \leq \varepsilon \cdot (X.wid - 1) \cdot k.$

Proof We prove by induction. Base case ($X.wid = 1$): $X.tsup = X.esup$. Thus, $X.tsup - X.esup \leq \varepsilon \cdot (X.wid - 1) \cdot k$.

Induction step: Consider an itemset of a form (X, $X.esup$, $X.wid$) that get deleted for some $wid(N) > 1$. The itemset is inserted in the SFI-forest when $wid(N+1)$ is being processed. The itemset X whose *window-id* is $wid(N + 1)$ in the FI-list could possibly have been deleted as late as the time when $X.esup \leq \varepsilon \cdot (wid(N+1) - X.wid + 1) \cdot k$. Therefore, $X.tsup$ of X when that deletion occurred is no more than $\varepsilon \cdot (wid(N+1) - X.wid + 1) \cdot k$. Furthermore, $X.esup$ is the estimated true support of the itemset X since it is inserted. It follows that $X.tsup$ which is the true support of X in the first window containing X though the current window, is at most $X.esup + \varepsilon \cdot (wid(N) - 1) \cdot k$. As a result, we have $X.tsup - X.esup \leq \varepsilon \cdot (X.wid - 1) \cdot k$. □

Because our algorithm is a *false-positive* algorithm, the answers produced by DSM-FI will have the following guarantees as same as that of BTS algorithm [14]:

(a) All itemsets whose true frequency exceeds $s \cdot N$ are output. There are no false negatives.
(b) No itemsets whose true frequency is less than $(s - \varepsilon) \cdot N$ is output.
(c) Estimated frequencies are less than the true frequencies by at most $\varepsilon \cdot N$.

If we want that the error dose not increase linearly with the value of window id, we can modify the line 5 of algorithm todoFIS from "**if** E.esup $\geq (s - \varepsilon) \cdot N$ **then**" to "**if** E.esup $\geq s \cdot N$ **then**". After that DSM-FI algorithm becomes a *false-negative* algorithm.

Note that a *false-positive* approach returns a set of itemsets that includes all frequent itemsets but also some infrequent itemsets. A *false-negative* algorithm returns a set of itemsets that does not include any infrequent itemsets but misses some frequent itemsets.

3.4.2 Space upper bound of prefix tree-based summary data structures

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

Theorem 2 *A prefix tree-based summary data structure has at most $2^{m'}$ nodes for storing the set of all frequent itemsets of data streams, when m' frequent items are given.*

Proof Let m' be the number of frequent items, i.e., 1-itemsets, in the data stream generated so far. Hence, the number of potential frequent itemsets is $\mathrm{C}(m', 1)$ regarding one item, $\mathrm{C}(m', 2)$ regarding two items,. . . , $\mathrm{C}(m', i)$ regarding i items,. . ., and $\mathrm{C}(m', m')$ regarding m' items according to the a priori property. In such a prefix tree-based summary data structure, an itemset is represented by a path and its appearance support is maintained in the last node of the path. Thus, there are $\mathrm{C}(m', 1)$ nodes in the first level, $\mathrm{C}(m', 2)$ nodes in the second level, . . . , $\mathrm{C}(m', i)$ nodes in the ith level, . . ., and $\mathrm{C}(m', m')$ nodes in the m'-th level. There are totally $\mathrm{C}(m', 1) + \mathrm{C}(m', 2) + \cdots + \mathrm{C}(m', i) + \cdots + \mathrm{C}(m', m')$ nodes in the prefix tree-based summary data structure. Consequently, the space upper bound of a prefix-tree based summary data structure is $O(2^{m'})$. □

The construction cost of summary data structure of DSM-FI algorithm is extremely less than that of BTS algorithm although theoretically, their worst case space complexities are same, i.e., $O(2^{m'})$, when m' frequent items are given.

3.4.3 Differences between SFI-forest and FP-tree

The SFI-forest can be regarded as an enhanced version of FP-tree [11], but there are some differences between SFI-forest and FP-tree. First, the construction step of both prefix tree-based structures is different. SFI-forest adopts an online incremental maintaining manner, but FP-tree is not. Second, the method used to construct a FP-tree needs two dataset scans, but DSM-FI scans the data only once. Third, SFI-forest uses the OFI-list to overcome the bottleneck (to generate a huge number of candidate 2-itemsets) of the a priori-based frequent itemset mining algorithms, but FP-growth uses a recursive method.

4 Performance evaluation

All the experiments are performed on a 1 GHz IBM X24 with 384 MB, and the program is written in Microsoft Visual C++ 6.0. To evaluate the performance of algorithm DSM-FI, we conduct the empirical studies based on the synthetic datasets. In Sect. 4.1, we report the scalability study of algorithm DSM-FI. In Sect. 4.2, we compare the memory and execution time requested by DSM-FI with algorithms BTS [14] and StreamMining [12]. The parameters of synthetic data generated by IBM synthetic data generator [2] are described as follows.
IBM synthetic dataset $T10.I5.D1$M and $T30.I20.D1$M. The first synthetic dataset $T10.I5$ has average transaction size T of 10 items and the average size of maximal frequent itemset I is 5-items. It is a *sparse* dataset. In the second dataset $T30.I20$, the average transaction

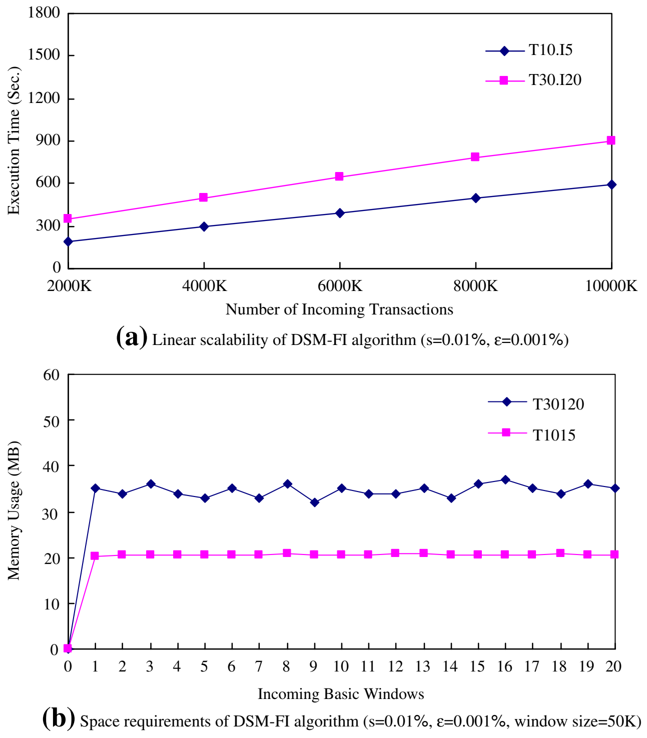

Fig. 9 Resource requirements of DSM-FI algorithm for IBM synthetic datasets: **a** Linear scalability of DSM-FI algorithm (s=0.01%, $\varepsilon = 0.001\%$), **b** Space requirements of DSM-FI algorithm ($s = 0.01\%$, $\varepsilon = 0.001\%$, window size = 50 K)

size T and average size of maximal frequent itemset I are set to 30 and 20, respectively. It is a *dense* dataset. Both synthetic datasets have 1,000,000 transactions. Items were drawn from a universe of 10 K distinct items. In the experiments, the synthetic data stream is broken into data windows with size 50 K (i.e., 50,000 transactions) for simulating the continuous characteristic of streaming data. Hence, there are total 20 windows in these experiments.

4.1 Scalability study of DSM-FI

In this experiment, we examine the two primary factors, *execution time* and *memory usage*, for mining frequent itemsets in a data stream environment, since both should be bounded online as time advances. Therefore, in Fig. 9a, the execution time grows smoothly as the dataset size increases from 2, 000 K to 10, 000 K. The default value of minimum support threshold s is 0.01%. The memory usage in Fig. 9b for both synthetic datasets is stable as time progresses, indicating the scalability and feasibility of algorithm DSM-FI. Notice that, the synthetic data stream used in Fig. 9b is broke into 20 basic windows each of 50 K, i.e., 50,000 transactions.

4.2 Comparisons with algorithms BTS and stream mining

In this experiment, we examine the execution time and memory usage among DSM-FI, BTS and StreamMining using dataset $T30.I20.D1$M. In Fig. 10a, we can see that the execution

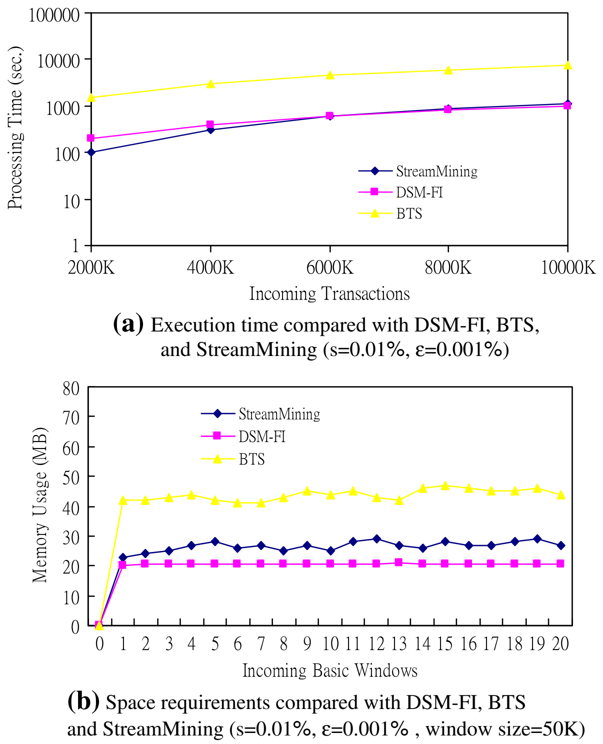

Fig. 10 Comparisons of DSM-FI, BTS and StreamMining: **a** Execution time compared with DSM-FI, BTS, and StreamMining (s=0.01%, ε = 0.001%), **b** Space requirements compared with DSM-FI, BTS and Stream-Mining (s = 0.01%, ε = 0.001%, window size = 50 K)

time incurred by DSM-FI is quite steady and is less than that of BTS. The execution time of StreamMining is less than our proposed algorithm in small datasets (2,000 K and 4,000 K), but is greater than DSM-FI in large datasets (8,000 K and 10,000 K). It shows that DSM-FI performs more efficiently than BTS. In Fig. 10b, the memory usage of DSM-FI is more stable and extremely less than that of BTS and StreamMining. It also shows that DSM-FI algorithm is more suitable for mining frequent itemsets in large-scale data streams.

5 Conclusions and future work

In this paper, we proposed a new, single-pass algorithm, called DSM-FI (data stream mining for frequent itemsets), which mines the set of all frequent itemsets in the landmark model of data streams. In the DSM-FI algorithm, a new in-memory summary data structure, called SFI-forest (summary frequent itemset forest), is constructed for storing the frequent and significant itemsets of the streaming data generated so far. An efficient frequent itemset search mechanism, called todoFIS (top-down Frequent Itemset Selection), is developed to find the set of all frequent itemsets from the current SFI-forest. Experiments with synthetic data streams show that DSM-FI is efficient on both sparse and dense datasets, and demonstrates linear scalability to vary long data streams. Moreover, DSM-FI outperforms the well-known, single-pass algorithms BTS and StreamMining for mining frequent itemsets over the entire history of the streaming data.

There are still many interesting research issues related to the extensions of DSM-FI algorithm, such as mining dynamic data streams, mining top-k frequent itemsets in streaming data, and mining constraint-based frequent itemsets in a landmark window over continuous data streams.

References

1. Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Buneman P, Jajodia S (eds) Proceedings of the 1993 international conference on management of data, Washington, D.C., pp 207–216
2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Bocca J, Jarke M, Zaniolo C (eds) Proceedings of the 20th international conference on lery large data bases, Chile, pp 487–499
3. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Popa L (eds) Proceedings of the 21th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Wisconsin, USA, pp 1–16
4. Chang J, Lee W (2004) Decaying obsolete information in finding recent frequent itemsets over data stream. In:IEICE transactions on information and systems, vol E87-D, no. 6, pp 1588–1592
5. Chang J, Lee W (2004) A sliding window method for finding recently frequent itemsets over online data streams. J Inf Sci Eng 20(4):753–762
6. Cheng J, Ke Y, Ng W (2006) A survey on algorithms for mining frequent itemsets over data streams. Knowl Inf Syst Doi:10.1007/s10115-007-0092-4
7. Chi Y, Wang H, Yu P, Muntz R (2006) Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl Inf Syst 10(3):265–294
8. Dang X, Ng W, Ong K (2007) Online mining of frequent sets in data streams with error guarantee. Knowl Inf Syst, Doi:10.1007/s10115-007-0106-2
9. Giannella C, Han J, Pei J, Yan X, Yu P (2003) Mining frequent patterns in data streams at multiple time granularities. In:Data mining: next generation challenges and future directions, AAAI/MIT, Kargupta H, Joshi A, Sivakumar K, Yesha Y (eds), pp 191–212
10. Golab L, Özsu M (2003) Issues in data stream management. ACM SIGMOD Rec 32(2):5–14
11. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Dis 8(1):53–87
12. Jin R, Agrawal G (2005) An algorithm for in-core frequent itemset mining on streaming data. In: Proceedings of the 5th IEEE international conference on data mining, Houston, TX, USA, pp 210–217
13. Lin C, Chiu D, Wu Y, Chen A (2005) Mining frequent itemsets from data streams with a time-sensitive sliding window. In: Proceedings of 2005 SIAM international conference on data mining, Newport Beach, CA, USA
14. Manku G, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of the 28th international conference on very large data bases, Hong Kong, China, pp 346–357
15. Teng W, Chen M, Yu P (2003) A regression-based temporal pattern mining scheme for data streams. In: Freytag J, Lockemann P, Abiteboul S, Carey M, Selinger P, Heuer A (eds) Proceedings of the 29th international conference on very large data bases, Berlin, Germany, pp 93–104
16. Wong R, Fu A (2006) Mining top-k frequent itemset from data streams. J Data Min Knowl Dis 13(2):193–217
17. Yu J, Chong Z, Lu H, Zhou A (2004) False positive or false negative: mining frequent itemsets from high speed transactional data streams. In: Nascimento M, Özsu M, Kossmann D, Miller R, Blakeley J, Schiefer K (eds) Proceedings of the 30th international conference on very large data bases, Toronto, Canada, pp 204–215
18. Zhu Y, Shasha D (2002) StatStream: statistical monitoring of thousands of data streams in real time. In: Proceedings of the 28th international conference on very large data bases, Hong Kong, China, pp 358–369

Author Biographies

Hua-Fu Li received the Ph.D. degree in Computer Science from National Chiao-Tung University, Taiwan, in 2006, under Prof. Suh-Yin Lee's supervision. He also received the B.S. and the M.S. degrees in Computer Science and Engineering from Tatung Institute of Technology, Taiwan, and in Computer Science from National Chengchi University, Taiwan, in 1999 and 2001, respectively. He is currently an Assistant Professor of Computer Science at Kainan University, Taiwan. His research interests include stream data mining, web data mining, multimedia data mining, and multimedia systems. He has published more than 20 papers in his research areas. He is the recipient of the International Computer Symposium 2004 Best Paper Award.

Man-Kwan Shan received the B.S. degree in Computer Engineering and the M.S. degree in Computer and Information Science both from National Chiao-Tung University, Taiwan, in 1986 and 1988, respectively. From 1988 to 1990, he served as a lecture in the Army Communications and Electronics School. Then, he worked as a lecture at the Computer Center of National Chiao-Tung University, where he supervised the Research and Development Division. He received the Ph.D. degree in Computer Science and Information Engineering from National Chiao-Tung University in 1998. Then he joined the Department of Computer Science at National Chengchi University as an assistant professor. He became an associated professor in 2003. His current research interests include data mining, multimedia systems, and multimedia mining. He has supervised students who were the winner of 2003 National Science Council Excellent M.S. Thesis Award, the winner of 2003 Acer Long Term Award for Excellent Thesis, the winner of 2000, 2003 National Science Council Excellent Undergraduate Research Award.

Suh-Yin Lee received the B.S. degree in electrical engineering from National Chiao-Tung University, Taiwan, in 1972, and the M.S. degree in computer science from University of Washington, U.S.A., in 1975, and the Ph.D. degree in computer science form Institute of Electronics, National Chiao-Tung University. She has been a professor in the Department of Computer Science and Information Engineering at National Chiao-Tung University since 1991, and was the chair of that department in 1991–1993. Her research interests include content-based indexing and retrieval, distributed multimedia information system, mobile computing, and data mining.