International Journal of Innovative Computing, Information and Control
Volume **8**, Number **3(A)**, March **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 1799–1820

A SINGLE-SCAN ALGORITHM FOR MINING SEQUENTIAL PATTERNS FROM DATA STREAMS

Hua-Fu Li[1,*], Chin-Chuan Ho[2], Hsuan-Sheng Chen[2] and Suh-Yin Lee[2]

[1]Department of Information Management
Kainan University
No. 1, Kainan Road, Luzhu Shiang, Taoyuan 338, Taiwan
[*]Corresponding author: hfli@mail.knu.edu.tw

[2]Department of Computer Science
National Chiao-Tung University
No. 1001, University Road, Hsinchu 300, Taiwan
{ hocc; xschen; sylee }@cs.nctu.edu.tw

Received November 2010; revised March 2011

Abstract. *Sequential pattern mining (SPAM) is one of the most interesting research issues of data mining. In this paper, a new research problem of mining data streams for sequential patterns is defined. A data stream is an unbound sequence of data elements arriving at a rapid rate. Based on the characteristics of data streams, the problem complexity of mining data streams for sequential patterns is more difficult than that of mining sequential patterns from large static databases. Therefore, mining sequential patterns from data streams is a challenging research issue of data mining and knowledge discovery. Hence, an efficient single-pass algorithm, called* ***IncSpam*** *(Incremental Sequential pattern mining of streaming itemset-sequences), is proposed for discovering sequential patterns from streaming itemset-sequences over extended sliding window models. In the framework of IncSpam algorithm, a new sliding window model, called* ***CSW-BV*** *(Customer Sliding Window with Bit-Vectors), and an extended lexicographic tree-based data structure, called* ***LexSeq-Tree*** *(Lexicographic Sequence Tree), are developed to reduce the time and memory needed to slide the windows over streaming data and maintain all sequential patterns of current sliding windows. Experimental results show that the proposed method is an efficient single-pass algorithm for mining sequential patterns from streaming data.*

Keywords: Data streams, Data mining, Data stream mining, Sequential pattern mining

1. Introduction.

Mining frequent patterns from data streams is one of the most interesting research issues of data mining and knowledge discovery. A *data stream* (also called *streaming data*) is an unbound sequence of data elements arriving at a rapid rate [10]. Many applications generate data streams in real time including sensor data flows from sensor network, online transaction flows of retail chains, Web record and click streams of Web applications, and call record flows of telecommunication [10,24]. Based on the unique characteristics of data streams, several new performance issues of mining data streams are described as follows [10,24]. First, data elements of data streams continuously arrive at a rapid rate and the number of data is huge. This issue means that the first performance requirement of mining data streams is real time for processing each new incoming element arrived in data streams. Second, once a data element is removed from the in-memory data structure of the proposed approaches, it is unable to backtrack over previously-arrived data elements from streaming data. Hence, the second requirement is only one-pass scan over each element of data streams. Third, the memory requirement for storing all streaming data is unlimited because streaming data is an unbounded sequence of data elements. Hence, the proposed methods use limited memory usage to maintain all essential information about an infinite streaming data. Consequently, the three major performance requirements of mining data streams are *single-pass mining*, *bounded space requirement* and *real-time element processing*. Since the unique characteristics of data streams, mining sequential patterns from data streams is more difficult than mining sequential patterns from a large static dataset.

In this paper, we address the issue of *single-pass* mining sequential patterns from data streams. Based on above descriptions of streaming data mining, previous *multiple-pass* sequential pattern mining methods [1,2,6-9,11,12,14,15,17,18,20-23] cannot feasibly be used for one-pass mining sequential patterns from data streams. Recently, many approaches were proposed for *one-pass* mining of sequential patterns from data streams [4,5,16,19]. However, these methods are focused on mining sequential patterns from streaming *item-sequences*, not from streaming *itemset-sequences*. Based on the descriptions of data streams, the research problem complexity of mining sequential patterns from streaming itemset-sequences is more difficult than that of mining sequential patterns from streaming item-sequences. Mining sequential patterns from streaming itemset-sequences is a challenging research problem of data mining and knowledge discovery. Consequently, we propose in this paper an efficient single-pass algorithm, called **IncSpam** (Incremental Sequential pattern mining of streaming itemset-sequences), for discovering sequential patterns from streaming itemset-sequences over extended sliding window models. In the framework of IncSpam algorithm, a new sliding window model, called **CSW-BV** (Customer Sliding Window with Bit-Vectors), and an extended lexicographic tree-based data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree), are developed to reduce the time and memory needed to slide the windows over streaming data and maintain all sequential patterns of current sliding windows.

1.1. Related works.

Many *multiple-scan* approaches are proposed for mining sequential patterns [1,2,8,11,15,18,20,21] and closed sequential patterns [6,7,14,22] from large static databases. Agrawal et al. [1] introduced the concept of sequential patterns and proposed efficient algorithms for this problem. Do and Kim [8] proposed algorithms with sequential mining techniques for clustering categorical data based on combinations of attribute values. Jea et al. [11] developed efficient hierarchical mining approaches for mining hybrid sequential patterns. Pei et al. [18] proposed an efficient algorithm, called PrefixSpan, to mine sequential patterns by prefix-projected pattern growth. Lin et al. [15] used memory indexing and database partitioning techniques to decrease the time of mining sequential patterns. The assumption is that entire sequence database can be loaded into main memory. An efficient algorithm, called SPAM [2], uses a lexicographic sequence tree to check all possible frequent sequences. Bitmap representation is used in SPAM for speeding up mining process of sequential patterns. Sui et al. [20] developed efficient types' commonality and sequential pattern mining based algorithms for extracting hyponymy relations between Chinese terms. Wang et al. [21] proposed effective sequential pattern mining algorithms and used the discovered Web sequential patterns to establish useful website navigation support systems.

Yan et al. [22] provided an efficient algorithm, called CloSpan, to mine closed sequential patterns in large datasets. Chen et al. [6] proposed efficient algorithms to mine the multiple-level sequential patterns. A concept hierarchy is used to represent the relationship between items. For the flexibility of sequence database, incremental mining of sequential patterns is another important research issue of sequential pattern mining. Lin et al. [14] proposed efficient algorithms for mining sequential patterns in a large database by using implicit merging and efficient counting techniques. Cheng et al. [7] developed efficient algorithms for mining sequential patterns by incremental updates. Although all above algorithms are efficient techniques for mining sequential patterns or closed sequential patterns from large databases, they are not feasible for sequential pattern mining in such streaming environment.

In recent years, mining sequential patterns from data streams has become one of the most important research issues of data mining [5,16,19]. Chen et al. [5] proposed an efficient algorithm for mining sequential patterns across multiple data streams. Marascu and Masseglia [16] proposed an efficient method, called SMDS, to mine sequential patterns from web usage sequences. In [19], the authors proposed a tree structure which offers a region technique to store sequential patterns from a data stream. However, these algorithms are *only* proposed for mining sequential patterns from streaming *item-sequences.* Consequently, the problem of mining sequential patterns from streaming *itemset-sequences* is discussed in this paper. We proposed an efficient one-pass algorithm, called **IncSpam** (Incremental Sequential pattern mining of streaming itemset-sequences), for mining sequential patterns over a stream of itemset-sequences. Based on our knowledge, the proposed algorithm IncSpam is the *first single-pass method* for this interesting research issue of sequential pattern mining.

1.2. **Our contributions.** The contributions of this work are summarized as follows.

- A new research problem of data mining for mining sequential patterns from a stream of itemset-sequences is defined in this paper.
- An efficient single-pass algorithm, called **IncSpam** (Incremental Sequential pattern mining of streaming itemset-sequences), is proposed for discovering a set of sequential patterns from continuous data streams with itemset-sequences.
- Based on the framework of IncSpam algorithm, a new sliding window model, called **CSW-BV** (Customer Sliding Window with Bit-Vectors), and an extended lexicographic tree-based data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree), are developed to reduce the time and memory needed to slide the windows over streaming data and maintain all sequential patterns of current sliding windows.
- Experimental results show that the proposed algorithm IncSpam is an efficient online mining technique for finding a set of sequential patterns over a stream of itemset-sequences.
- Based on our best knowledge, the proposed method IncSpam is the first single-pass algorithm for mining sequential patterns from streaming itemset-sequences.

1.3. **Roadmap.** The remainder of the paper is organized as follows. Section 2 defines the problem of single-pass mining of sequential patterns from streaming itemset-sequences. The proposed algorithm IncSpam algorithm is proposed in Section 3 for mining sequential patterns from streaming itemset-sequences. Experimental results of the proposed IncSpam algorithm are discussed in Section 4. Finally, we conclude the work in Section 5.

2. Problem Statement.

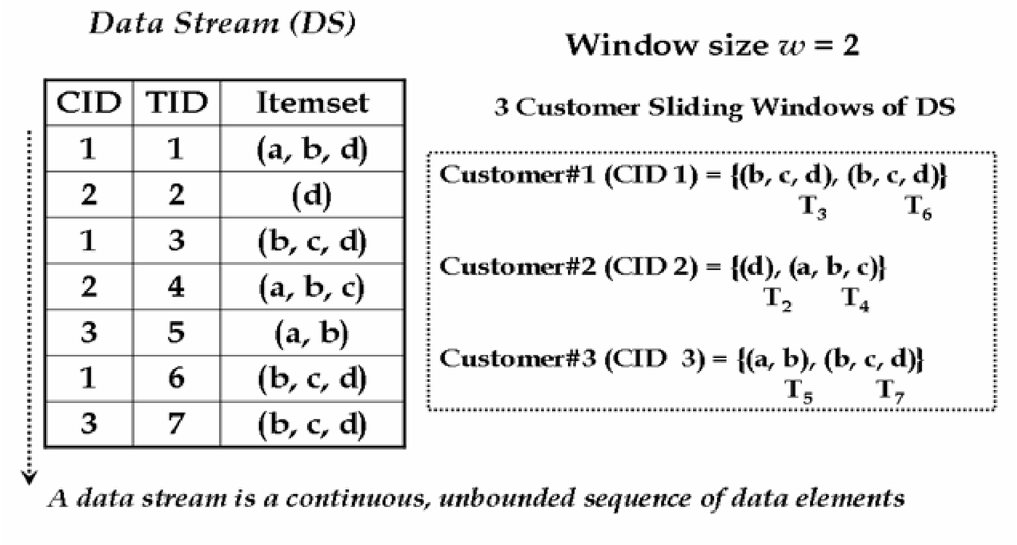
Let $\psi = \{i_1, i_2, \cdots, i_M\}$ be a set of literals, called **items**. An **itemset** $X = (e_1, e_2, \cdots, e_p)$ is a non-empty set of p items such that $X \subseteq \psi$ and $p \geq 1$. A **transaction** $T = (TID, X)$ consists of an itemset X and a unique transaction identifier *TID.* An **itemset-sequence** (or **sequence** in short) $s = \langle T_1 T_2 \cdots T_q \rangle$ is an ordered list of q transactions, where $q \geq 1$. In other words, a sequence $s = \langle Y_1 Y_2 \cdots Y_q \rangle$ is an ordered list of q itemsets from Y_1 to Y_q. Without loss of generality, we assume that the items in an element are in lexicographic order. Let $C = \{c_1, c_2, \cdots, c_N\}$ be a set of all **customers**. Each customer has a unique identifier *CID* and an itemset-sequence. The **length** of an itemset-sequence s, written as *s.length*, is the total number of items in all the itemsets of s. A sequence s is a k-sequence if $s.length = k$. A sequence $s = \langle a_1, a_2, \cdots, a_p\rangle$ is a **subsequence** of a sequence $s' = \langle b_1, b_2, \cdots, b_q\rangle$ if there exist integers $i_1 < i_2 < \cdots < i_p$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i1}, \cdots$, and $a_p \subseteq b_{ip}$. Note that itemset-sequence is also called **customer sequence** in this paper.

A **data stream** $DS = \{D_1, D_2, \cdots, D_n\}$ is a continuous, unbounded sequence of data elements, where n is the identifier of new incoming data element D_n. A **data element** is a set of (*CID*, *TID*, *itemset*) pairs, in which each customer identifier is distinct with others. A **customer sliding window** CSW_{CID} of size w contains w most recent data elements for the customer with identifier *CID*. The **window size** w of CSW_{CID} is the number of data elements in it.

A set of current customer sliding windows of data streams is called the **current database**, denoted by *CDB*. The **support** of customer sequence s, denoted as *s.sup*, is the number of customer sequences containing s divided by the total number of customer sequences in *CDB*. A sequence s is a **frequent sequence**, also called a **sequential pattern**, if $s.sup \geq minsup$, where *minsup* is the user specified minimum support threshold in the range of $[0, 1]$.

Definition 2.1. ***(Problem Definition of Single-Scan Mining of Sequential Patterns from Data Streams)*** *Given a minimum support threshold minsup and the size of customer sliding window w and the data stream DS, the problem of single-scan mining of sequential patterns from streaming itemset-sequences is to discover the set of all frequent sequences from current database CDB, which includes most recent w data elements in each customer sliding window.*

Example 2.1. *An example data stream DS with seven customer sequences* $T_1, T_2, \cdots, T_7$ *is given in Figure 1, where a, b, c and d are items. In this instance, we assume that the window size of each CSW is 2, that is, the most recent 2 transactions of each customer are recorded in customer sliding windows with each customer identifier CID as the right side of Figure 1. From this figure, we can find that the* CSW_1 *is composed of* T_3 *and* T_6*, although there are three transactions with CID#1, i.e.,* T_1, T_3 *and* T_6. *Moreover,* CSW_2 *and* CSW_3 *are* $\{(d), (a, b, c)\}$ *and* $\{(a, b), (b, c, d)\}$*, respectively.*


Data Stream (DS)

CID	TID	Itemset
1	1	(a, b, d)
2	2	(d)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)
1	6	(b, c, d)
3	7	(b, c, d)

A data stream is a continuous, unbounded sequence of data elements

Window size $w = 2$

3 Customer Sliding Windows of DS

Customer#1 (CID 1) = {(b, c, d), (b, c, d)}
T_3 T_6

Customer#2 (CID 2) = {(d), (a, b, c)}
T_2 T_4

Customer#3 (CID 3) = {(a, b), (b, c, d)}
T_5 T_7

FIGURE 1. An example data stream DS and current CSW with size $w = 2$

3. On Mining of Sequential Patterns from Streaming Itemset-Sequences.

In this section, an efficient single-pass algorithm, called **IncSpam** (Incremental Sequential pattern mining of Streaming itemset-sequences), is proposed for discovering a set of sequential patterns from a stream of itemset-sequences over sliding windows. Based on the framework of IncSpam, a new sliding window model, called **CSW-BV** (Customer Sliding Window with Bit-Vectors), and an extended lexicographic tree-based data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree), are developed to reduce the time and memory needed to slide the windows over streaming itemset-sequences and maintain all sequential patterns of current customer sliding windows. The proposed algorithm IncSpam is composed of three major phases, *window initialization phase, window sliding phase and sequential pattern generation phase*, for mining sequential patterns from streaming item-sequences over sliding windows. Given a customer sliding window CSW of size w, the window initialization phase is activated while the number of transactions generated so far is less than the window size w. After the window CSW is full, the second phase is activated. Furthermore, sequential pattern generation is performed periodically in the window sliding phase. Consequently, we shall focus on the definitions of CSW-BV and LexSeq-Tree and devise efficient algorithms for the building and maintenance of CSW-BV and LexSeq-Tree in this section.

In the window initialization phase of IncSpam algorithm, an effective customer sliding window model, called **CSW-BV** (Customer Sliding Window with Bit-Vectors), is developed for storing customer sequences of each customer. In the CSW-BV model, each transaction of customer-sequence is transformed into a bit-vector based sequence representation. Constructing and sliding of the CSW-BV model is discussed in Section 3.1. Furthermore, an extended lexicographic-tree based data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree), is constructed for maintaining all sequential patterns from current database.

In window sliding phase of IncSpam algorithm, two operations are performed. First, CSW-BV model is updated. It is because of the oldest sequence information is dropped and new incoming sequence information is inserted into the CSW-BV model. After that, oldest sequence information is also deleted from and new incoming customer sequences are inserted into the current LexSeq-Tree.

In the last phase, i.e., sequential pattern generation phase, of IncSpam algorithm, a set of sequential patterns are generated by traversing the current LexSeq-Tree by using an efficient method. Furthermore, a weight mechanism is also used in the proposed IncSpam algorithm to judge the importance of customer sequences and ensure the correctness of the customer sliding window model.

3.1. Window initialization phase: building of the CSW-BV and LexSeq-Tree.

In the window initialization phase of IncSpam algorithm, an effective customer sliding window model, called **CSW-BV** (Customer Sliding Window with Bit-Vectors, as discussed in Section 3.1.1), is developed for storing customer sequences of each customer. Moreover, an extended lexicographic-tree based data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree, as discussed in Section 3.1.2), is constructed for maintaining all sequential patterns from current database. However, in the issue of sliding window-based mining of frequent itemsets from data streams, the lastest w transactions of data streams are maintained in a window for frequent pattern discovery [3,5,24]. But, the technique is not feasible for mining sequential patterns from streaming itemset-sequences. Hence, a new sliding window model, called *customer sliding window* (CSW), is proposed in this work for mining sequential patterns from streaming itemset sequences. Based on the definition of customer sequence, each customer has several transactions generated from data streams. For maintaining the transactional information of each customer, IncSpam uses the *customer sliding window model* (CSW-model) to maintain the latest w transactions for each customer from data streams. Consequently, each customer has a list of most recent w transactions in the CSW-model as example in Figure 1.

3.1.1. *Customer sliding window with bit-vectors (CSW-BV).* In the framework of IncSpam algorithm, the bit-vector techniques are used in the CSW-model to maintain the customer transactional information of sliding windows from data streams. The extended CSW-model is called *customer sliding window with bit-vectors* (CSW-BV) in this paper. Bit-vector representation of all items with latest two transactions for each customer of Example 2.1 is given in Figure 2. In this figure, all bit-vectors of customers are recorded as a unique data structure for each customer. The unique data structure is called *CSW-BV*.

Item	Customer#1	Customer#2	Customer#3
a	00	01	10
b	11	11	11
c	11	01	01
d	11	00	01

FIGURE 2. Bit-vectors of all items with latest two transactions for each customer

In the CSW-BV model, only the latest N transactions of each customer sequence s are maintained in the extended sliding window model, where N is a user-defined window size. Each bit-vector of item x contains N bits to represent the occurrences of x in the latest N transactions. If an item x is in the i-th sequence of CSW, the i-th bit of item x is set to be 1; otherwise, it is set to be 0. An example CSW-BV model of Example 2.1 is shown in Figure 3. In the CSW-BV model, each customer has a customer identifier (CID) and is associated with a list of bit vectors of all items. For example, customer with CID = 2 has a list of bit-vectors with four items a, b, c and d, i.e., $\mathbf{BIT}(2, a) = [0, 1]$, $\mathbf{BIT}(2, b) = [2, 1]$, $\mathbf{BIT}(2, c) = [0, 1]$ and $\mathbf{BIT}(2, d) = [0, 0]$, where the first parameter with number 2 is the CID with customer#2.

CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)
1	6	(b, c, d)
3	7	(b, c, d)

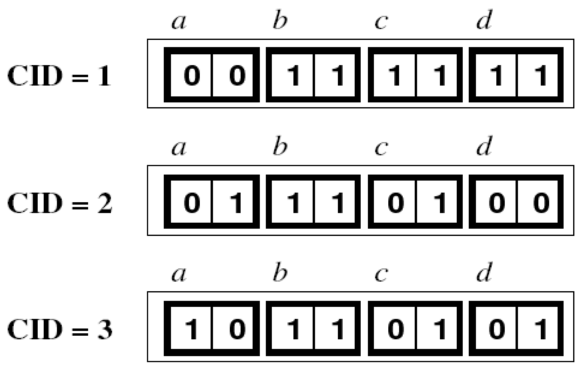


FIGURE 3. CSW-BV of Example 2.1 in the window initialization phase of IncSpam algorithm

3.1.2. *Lexicographic sequence tree (LexSeq-Tree).* In the framework of IncSpam algorithm, a lexicographic tree-based summary data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree), is developed for maintaining all customer sequences of current database. Before introducing the proposed data structure LexSeq-Tree, another effective mechanism, called ***Seqence-index Set*** (*SeqIdx-Set*), is used to improve the performance of LexSeq-Tree construction process. *SeqIdx-Set* can be used to deal with the problem of huge number of generated candidate itemset-sequences from data streams. For each itemset-sequence, *SeqIdx-Set* records the first positions of this sequence in all customer-sequences within current database.

Definition 3.1. *(**Sequence-index Set: SeqIdx-Set**) For a sequence ρ, the first occurring position in another customer sequence s of ρ is recorded as $\boldsymbol{\rho}$**-pos**$_{\boldsymbol{s}}$. If ρ is not in s, the value of ρ-pos$_s$ is 0. The collection of these ρ-pos values in the order of customer id (CID) is called SeqIdx-Set $\boldsymbol{\rho}$**-idx**. For convenience, ρ-pos$_s$ can be represented as $\boldsymbol{\rho}$**-idx**$[\boldsymbol{s}]$.*

For example, in Figure 3, there are four ρ-*idxs*: $\langle(a)\rangle$-*idx* $= [0, 2, 1]$, $\langle(b)\rangle$-*idx* $= [1, 1, 1]$, $\langle(c)\rangle$-*idx* $= [1, 2, 2]$ and $\langle(d)\rangle$-*idx* $= [1, 0, 2]$. Each integer in the array represents the first position of ρ in each customer sliding window. Note that $\langle(d)\rangle$-*idx* $= [1, 0, 2]$ indicates that the 1-sequence (d) is contained in the first element of customer#1 and the second element of customer#3, where each element is an itemset.

After processing CSW-BVs and all SeqIdx-Sets, the proposed IncSpam algorithm constructs an effective data structure, called **LexSeq-Tree** (Lexicographic Sequence Tree, see Definition 3.3), based on these CSW-BVs and SeqIdx-Sets. Before we define the proposed data structure LexSeq-Tree, a notation of lexicographic ordering is introduced.

Definition 3.2. *(**lexicographic ordering:** $\leq$) Given two sequences s_a and s_b, if s_a is a subsequence of s_b, we say that sequences s_a and s_b have lexicographic ordering and denoted as $s_a \leq s_b$.*

Next, the proposed data structure **LexSeq-Tree** is defined and an example of partial LexSeq-Tree is given in Figure 4.

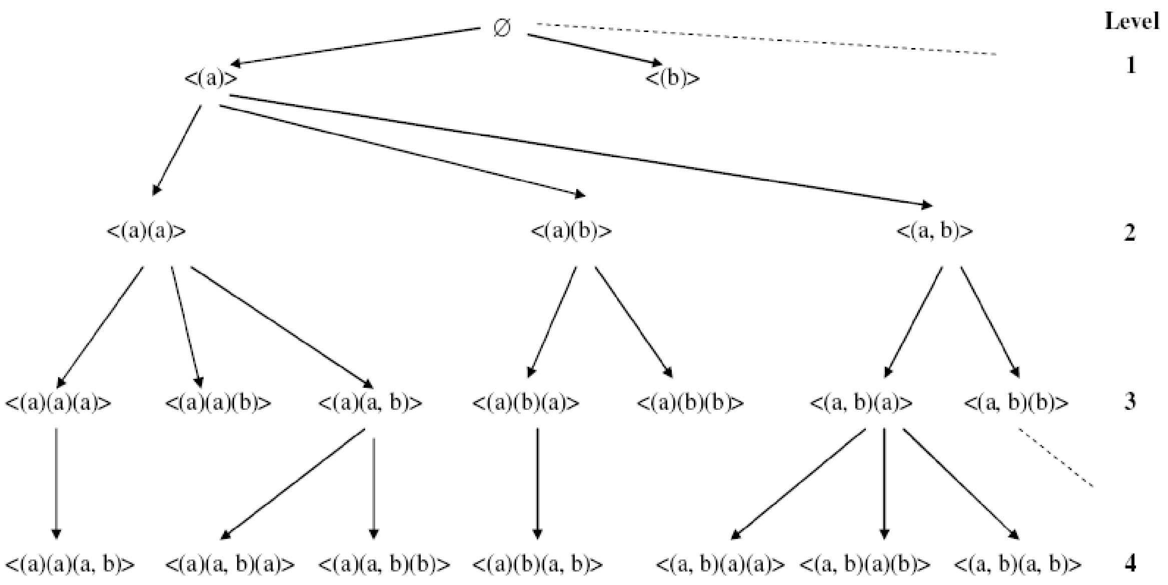


FIGURE 4. An example of partial LexSeq-Tree

Definition 3.3. *(**Lexicographic Sequence Tree: LexSeq-Tree**) **LexSeq-Tree** is an extended lexicographic tree-based summary data structure and defined as follows.*

1. The root node of LexSeq-Tree is labeled with a dummy symbol ϕ, as shown in Figure 4.

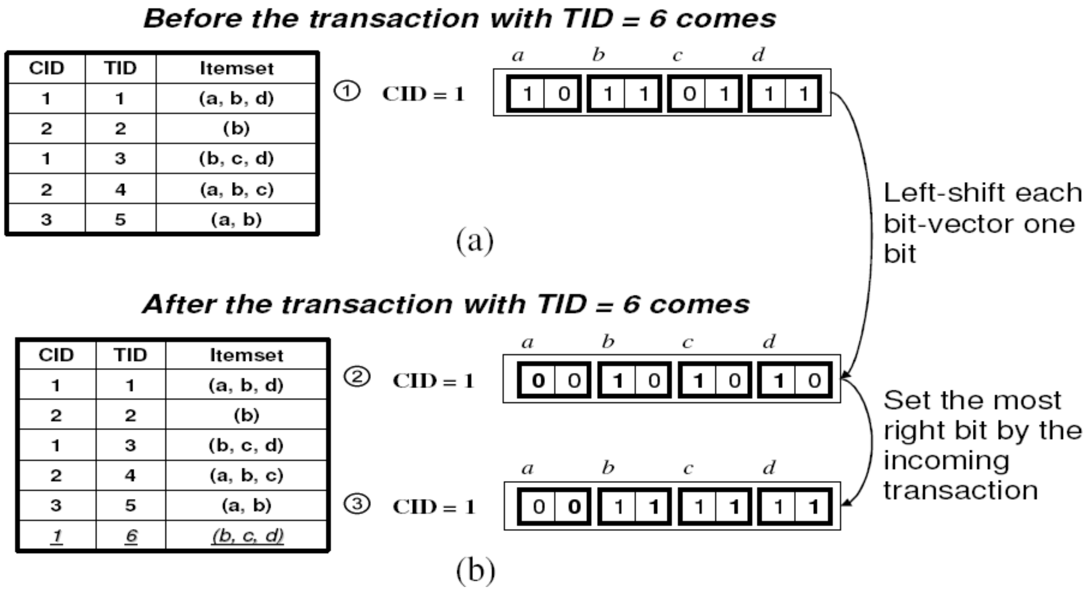
2. *Two kinds of nodes are stored in the LexSeq-Tree, where each node is an itemset-sequence. First type of tree node is a* ***sequence-extended sequence*** *(se-seq) and second type of node is an* ***itemset-extended sequence*** *(ie-seq).*
3. *A* ***sequence-extended sequence*** *(se-seq) is a sequence where it is extended by adding one itemset to the end of its parent's node in the LexSeq-Tree. For example, in Figure 4, two 3-sequences $\langle(a)(a)(a)\rangle$ and $\langle(a)(a)(b)\rangle$ are generated by adding one itemset, i.e., (a), to the end of the 2-sequence $\langle(a)(a)\rangle$ of the level 2 of the LexSeq-Tree.*
4. *An* ***itemset-extended sequence*** *(ie-seq) is a sequence by adding an item to the last element of its parent node in the LexSeq-Tree. For example, in Figure 4, the 3-sequence $\langle(a),(a,b)\rangle$ is generated by adding one item (b) to the end of the 2-sequence $\langle(a),(a)\rangle$ of the level 2 of the LexSeq-Tree.*

3.2. **Window sliding phase: maintenance of the CSW-BV and ρ-idx.** After constructing the CSW-BVs and LexSeq-Tree in the window initialization phase, we describe the maintenance process of CSW-BVs and ρ-idx in the window sliding phase of IncSpam algorithm in this section. When a new transaction arrives in current database, the customer identifier CID is checked to find out that which CSW-BV has to be modified. If the number of transactions of the customer with CID is greater than the size of CSW, the window sliding phase of IncSpam is performed. The process of CSW sliding is described as follows. First, each bit-vector is performed left-shift operation on one bit to eliminate the dropped transaction information and modified the most-right bit by the information of the new incoming transaction. Then, if the item is recorded in the incoming transaction, the most right bit of its bit-vector is set to be value one; otherwise, the most right bit of this bit-vector is set to be value zero.

An example of the window sliding process is given in Figure 5. In Figure 5(a), current database is composed of three customers, i.e., CID = 1, CID = 2 and CID = 3, and five transactions, i.e., T_1, T_2, T_3, T_4 and T_5. In this figure, we can find that the CSW-BV of customer#1 (CID = 1) gives the status of CSW-BV$_{C1}$ = $\{(a, [1, 0]), (b, [1, 1]), (c, [0, 1]), (d, [1, 1])\}$ model before the new transaction T_6 is inserted into current database. Other CSW-BV models of customers #2, #3 and #4 are omitted in this figure. In Figure 5(b), the second CSW-BV gives the result after performing left-shift operation on each bit-vector one bit. The step is preformed before processing T_6. Therefore, CSW-BV$_{C1}$ is modified from $\{(a,[1,0]),(b,[1,1]),(c,[0,1]),(d,[1,1])\}$ to $\{(a,[0,0]),(b,[1,0]),(c,[1,0]),(d,[1,0])\}$. Finally, the third CSW-BV of Figure 5(b) gives the final result after setting the most right bit by processing the incoming transaction T_6. Consequently, CSW-BV$_{C1}$ is modified from $\{(a,[0,0]),(b,[1,0]),(c,[1,0]),(d,[1,0])\}$ to $\{(a,[0,0]),(b,[1,1]),(c,[1,1]),(d,[1,1])\}$.

The ρ-idx of each item, i.e., 1-sequence, is maintained according to these CSW-BVs. When a window sliding occurs for a CSW-BV of the customer c, the value of each ρ-idx[c] is decreased by one. After that, if the value of ρ-idx[c] is zero, the bit-vector of ρ is checked to find out the new first occurring position. If ρ does not exist in this customer-sequence anymore, the ρ-idx[c] is set to zero. For example, in the CSW-BV$_{C1}$ of Figure 5(a), $\langle(a)\rangle$-idx[1] is 1. After sliding the window, $\langle(a)\rangle$-idx[1] is decreased to 0 in the second CSW-BV$_{C1}$. Furthermore, in the third CSW-BV$_{C1}$ as shown in Figure 5(b), we can find that $\langle(a)\rangle$-idx[1] is 0. This is because that the 1-sequence, $\langle(a)\rangle$, does not appeared in the customer-sequences of customer#1 in the current CSW.

In the proposed data structure LexSeq-Tree of IncSpam algorithm, each node ρ uses ρ-idx to compute the support value of sequence ρ. The support computing method consists of two steps: *support computing in S-step* and *support computing in I-step.* That means the step of generating sequence-extended sequence from LexSeq-Tree construction is called



CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)

CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)
1	6	(b, c, d)

Before the transaction with TID = 6 comes
① CID = 1
a b c d
1 0 1 1 0 1 1 1
(a)
Left-shift each bit-vector one bit
After the transaction with TID = 6 comes
② CID = 1
0 0 1 0 1 0 1 0
③ CID = 1
0 0 1 1 1 1 1 1
Set the most right bit by the incoming transaction
(b)

FIGURE 5. Examples of window sliding in CSW-BVC1: (a) CSW-BVC1 before the transaction T6 arrives (b) CSW-BVC1 after the transaction T6 arrives

S-step (discussed in Section 3.2.1). The step of generating itemset-extended sequence from LexSeq-Tree construction is called **I-step** (discussed in Section 3.2.2).

3.2.1. *Support computing in S-step.* In this section, the method used to count the support in *S-step* is discussed. Assume that there are one sequence α and an appended 1-itemset (β). By performing *S-step*, an S-extended sequence γ is generated by concentrating the sequence α and 1-itemset (β). Next, the process of generating γ-idx and computing the support of the S-extended sequence γ by using α-idx and β-idx is described as follows. First, α-idx[c] and β-idx[c] are checked for each customer c. If the value of α-idx[c] or the value of β-idx[c] is zero, the value of γ-idx[c] is modified to be zero. The condition means that the S-extended sequence γ can not exist in the customer-sequence of customer c. If the value of α-idx[c] and the value of β-idx[c] are both not zero, the sequence γ is contained in the customer sequence of customer c with high probability. After that, the corresponding position values have to be checked. Furthermore, two cases of α-idx[c] and β-idx[c] have to be discussed as follows.

Case (a) ***if*** $\boldsymbol{\alpha}$***-idx***$[\boldsymbol{c}] < \boldsymbol{\beta}$***-idx***$[\boldsymbol{c}]$: In this case, the sequence α appears before β and sequence γ exists in the customer-sequence of customer c. Hence, the value of γ-idx[c] is set to be the same value of β-idx[c].

Case (b) ***if*** $\boldsymbol{\alpha}$***-idx***$[\boldsymbol{c}] \geq \boldsymbol{\beta}$***-idx***$[\boldsymbol{c}]$: In this case, the CSW-BV of customer c is checked to verify that whether the sequence γ exists or not. Note that the bit-vector of item x in the CSW-BV of customer c is denoted as CSW-BV$_c(x)$. First, a left-shifting operation is performed on CSW-BV$_c(\beta)$ by α-idx[c] bits. If the result of bit-vector after left-shifting is a non-zero bit-vector, γ is contained in the customer-sequence of sequence c. Furthermore, we assume that the position of the first non-zero bit in this result is h. Therefore, the first position of the sequence γ, i.e., γ-idx[c], is set to α-idx[c] + h. Otherwise, γ is not contained in the customer-sequence of sequence c. In this case, the value of γ-idx[c] is set to zero. Consequently, the support value of γ can be computed by counting the number of non-zero positions.

3.2.2. *Support computing in I-step.* In this section, the method used to count the support in *I-step* is discussed. Assume that there are one sequence α and an appended 1-itemset (β). By performing *I-step*, an I-extended sequence γ is generated by concentrating the sequence α and 1-itemset (β). The process of generating γ-idx and computing the support of I-extended sequence γ by using α-idx and β-idx is described as follows.

As the descriptions in S-step of Section 3.2.1, α-idx[c] and β-idx[c] for each customer c are checked for generating γ-idx and computing the support of I-extended sequence γ. If either the value of α-idx[c] or β-idx[c] is zero, the value of γ-idx[c] is set to zero. If the value of α-idx[c] and the value of β-idx[c] are both not zero, the corresponding position values have to be checked. Furthermore, two cases of α-idx[c] and β-idx[c] in *I-step* need to be discussed as follows.

Case (a) ***if*** $\boldsymbol{\alpha}$***-idx***$[\boldsymbol{c}] = \boldsymbol{\beta}$***-idx***$[\boldsymbol{c}]$: In this case, the last itemset of α and the itemset β are in the same position of customer-sequence of customer c. Based on the definition of itemset-extended sequence, sequence γ exists and the value of γ-idx[c] is equal to the value of β-idx[c].

Case (b) ***if*** $\boldsymbol{\alpha}$***-idx***$[\boldsymbol{c}] \neq \boldsymbol{\beta}$***-idx***$[\boldsymbol{c}]$: In this case, the CSW-BV of customer c is checked as follows. Assume that itemset X is the last itemset of sequence α, a bit-vector of X is generated by performing bitwise-AND operation on CSW-BV$_c(x_1)$, CSW-BV$_c(x_2)$, $\cdots$, and CSW-BV$_c(x_k)$, where x_i is an item of X, where $\forall i$, $i = 1, 2, \cdots, k$. After that, the bit-vector of X is left-shifted (α-idx[c] − 1) bits. Furthermore, if the result of bit-vector after performing bitwise-AND operation is a non-zero bit-vector, γ is contained in the customer-sequence of sequence c. At this time, we assume that the position of the first non-zero bit in the result of bit-vector is h. Hence, the value of the first position of γ, i.e., γ-idx[c], is (α-idx[c] − 1) + h. Otherwise, if the result of bit-vector after performing bitwise-AND operation is a zero bit-vector, γ is not contained in the customer-sequence of customer c and the value of γ-idx[c] is zero.

3.2.3. *Weight of customer-sequence for mining sequential patterns from data streams.* In the framework of IncSpam algorithm, each customer usually maintains a CSW to keep the latest w transactions. However, some customers may have no transactions in recent transactional data streams. Hence, these customer-sequences with out-of-date transactions would generate a *false-positive* problem in our algorithm. It is because the supports of some sequential patterns generated by the proposed algorithm are overly counted. Hence, an effective method is developed and used in our IncSpam algorithm for this issue.

Figure 6 gives an example of these transactions in a data stream. In this figure, we can find that the customer-sequences with these transactions are less important than that of others. Hence, a weight mechanism is used to judge the importance of customer sequences. In this weight mechanism, each customer-sequence s has it own weight w_s, where $0 \leq w_s \leq 1$. Each weight w_s is decayed when the new incoming transactions of data streams do not contain the sequence s. Furthermore, the value of weight w_s is set to be one when a new transaction of customer-sequence s comes.

In the proposed method used in our IncSpam algorithm, a decay function $w_s = 1 \times d^p$ is used to compute the weights of customer-sequences, where d is a user-defined *decay-rate* and p is a *decay-period* of the customer-sequence. Note that decay-rate d is used to decide that how fast a customer-sequence is decayed and decay-period p is the number of transactions between the new incoming transaction and the latest transaction within sequence s. Therefore, decay-period p can be rewritten as $p = (\textit{incoming transaction TID} - \textit{the latest transaction TID of sequence } s)$. Moreover, the decay-rate d is defined as $d = b^{(1/h)}$ $(b < 1, h \geq 1, b^{-1} \leq d < 1)$, where *Decay-base* b is the number of weight

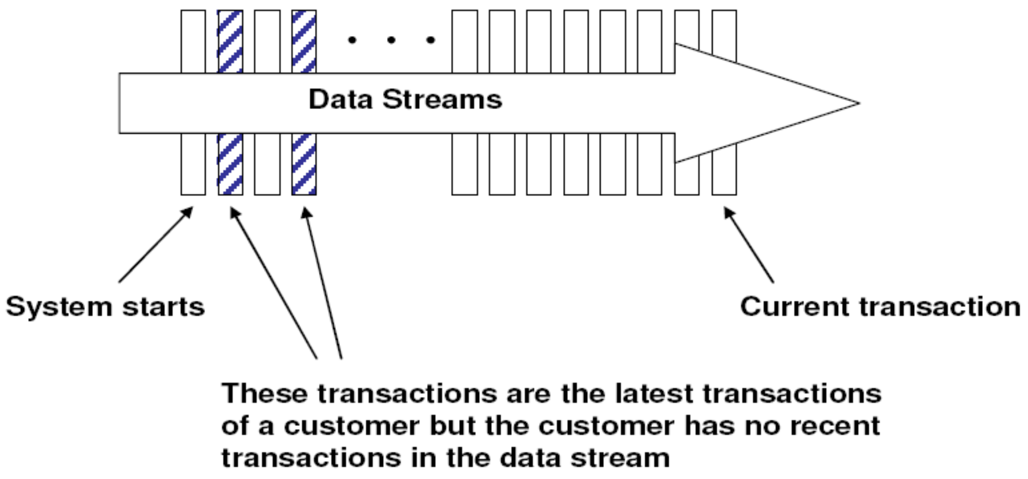
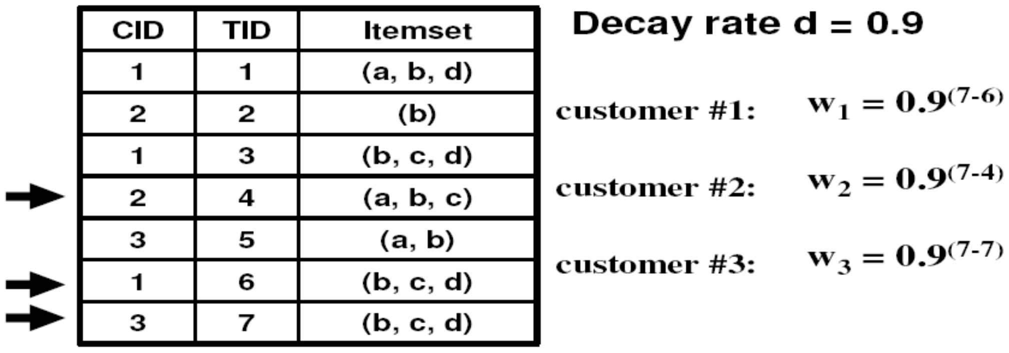

Data Streams
System starts
Current transaction
These transactions are the latest transactions of a customer but the customer has no recent transactions in the data stream

FIGURE 6. No recent transactions generated by an example customer in data streams

reduction per decay-unit, and the *Decay-base-life* h is the number of decay-units that makes the current weight be $1/b$.

An example of calculating the weights of customers is given in Figure 7. Assume that a new incoming transaction is T_7 and the user-specific decay rate d is 0.9. Note that the latest transaction of each customer is pointed by an arrow. Let us take customer #1 as an illustrating instance. The latest transaction of customer #1 is the transaction T_6. Hence, the weight of customer-sequence of customer #1, i.e., w_1, is $0.9^{(7-6)} = 0.9$.



	CID	TID	Itemset
	1	1	(a, b, d)
	2	2	(b)
	1	3	(b, c, d)
➡	2	4	(a, b, c)
	3	5	(a, b)
➡	1	6	(b, c, d)
➡	3	7	(b, c, d)

Decay rate d = 0.9

customer #1: $w_1 = 0.9^{(7-6)}$

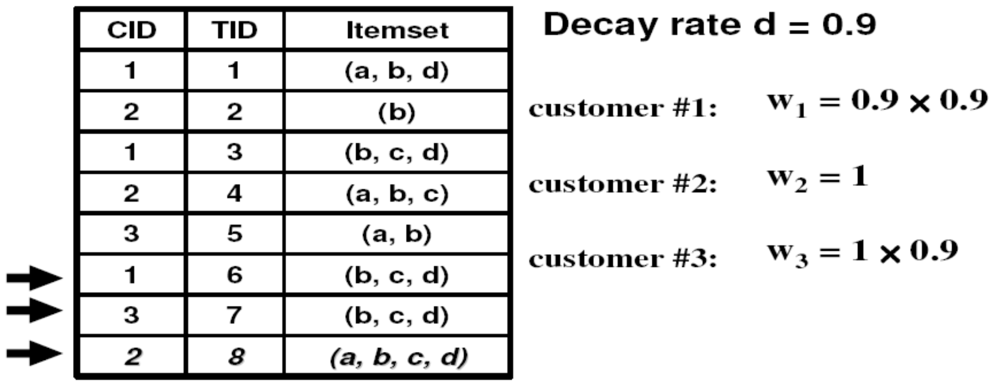
customer #2: $w_2 = 0.9^{(7-4)}$

customer #3: $w_3 = 0.9^{(7-7)}$

FIGURE 7. Example of calculating the weights of customers

We do not need to calculate decay-period when a new transaction comes in the proposed algorithm. The weight of the customer that the new incoming transaction belongs to is set to one. Other transactions are decayed by the user-specific decay-rate d. An example after processing a new transaction with TID = 8, i.e., T_8, is given Figure 8. In this figure, the weight of customer #2 is set to 1, i.e., $w_2 = 1$, and the others are decayed by 0.9. The support of a customer-sequence ρ is not the number of non-zero positions in the ρ-idx. It is counted by adding the weights of customer-sequences with ρ. Note that the updating method of weights is efficient in incremental mining of sequential patterns from streaming itemset-sequences. Updating support value of an existing node in the LexSeq-Tree can also use the same mechasism. The process of support updating of LexSeq-Tree is introduced in the next section.

3.3. **Our proposed algorithm: IncSpam.** In this section, we introduce the proposed IncSpam algorithm for mining sequential patterns from streaming itemset-sequences.



CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)
1	6	(b, c, d)
3	7	(b, c, d)
2	*8*	*(a, b, c, d)*

Decay rate d = 0.9

customer #1: $w_1 = 0.9 \times 0.9$

customer #2: $w_2 = 1$

customer #3: $w_3 = 1 \times 0.9$

Figure 8. Status for weight of customer-sequence after processing a new incoming transaction T_8

Main functions of the proposed algorithm are given in Figure 9. First, the CSW-BV of each customer is processed and modified (from lines 1 to 4 of Figure 9). All generated frequent items are maintained in a temporal pool *TPool* for constructing the proposed LexSeq-Tree. After modifying all CSW-BVs, function *LexSeq-Tree-Maintain* is performed. Based on our approach IncSpam, the proposed data structure LexSeq-Tree is dynamically maintained by processing each incoming transaction from streaming itemset-sequences while CSW is sliding. Assume that the new transaction ω comes and ω belongs to the customer c. The effect of ω upon the LexSeq-Tree T is modified to a changed LexSeq-Tree T'. Hence, three cases are discussed as follows.

```
Algorithm IncSpam: Mining Sequential Patterns from Streaming Itemset-Sequences.
Input Data: DS: data stream, d: decay-rate, w: window size, minsup: a minimum
            support threshold;
Output Data: A generated LexSeq-Tree of current database;
1:    foreach transaction $T$ of data stream $DS$ do
2:        find out which customer $c$ is contained in the incoming transaction $T$;
3:        update the CSW-BV of customer $c$ by using the information of the
          transactions;
4:        update the weight of each customer by decaying the decay-rate $d$;
5:        insert all frequent items into a temporal pool TPool;
6:        LexSeq-Tree-Maintain(c, F);
7:    endfor
```

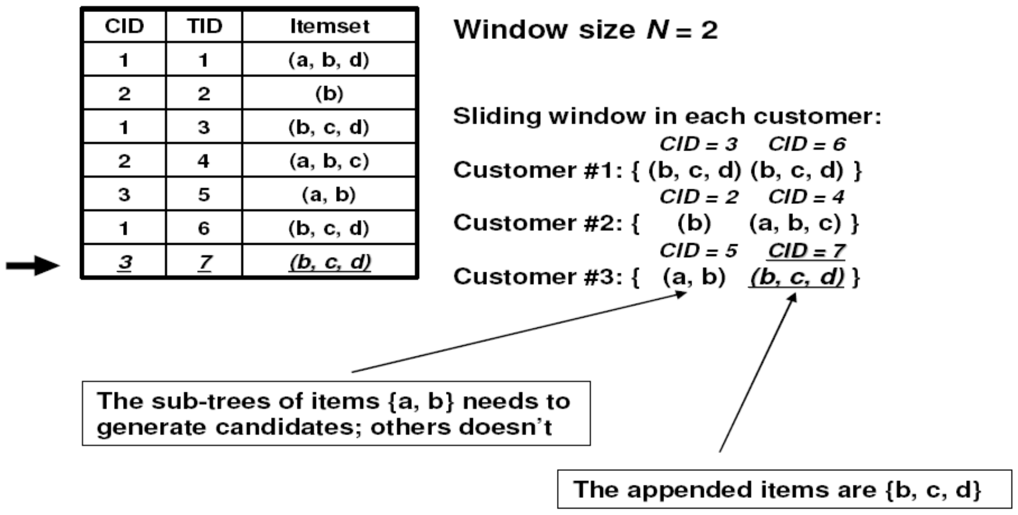
Figure 9. Main functions of the proposed IncSpam algorithm

Case (a): *A pattern is a node in both LexSeq-Trees T and T'.* In this case, only its ρ-idx and support are updated.

Case (b): *A pattern which is not a node in T but it is a node of T'.* A new pattern is generated from the new incoming transaction. Based on the Apriori property of sequential pattern mining [1], prefix of the new pattern is also a frequent pattern. Hence, we only need to generate candidates from the leaf nodes of T. Two ways are used in the proposed algorithm IncSpam to reduce the number of generated candidates. First, we only consider the items in the incoming transaction to append on the leaf nodes. It is because that the new patterns must contain these items in the end. Second, the generated candidates must begin with the items in the customer-sequence of customer c since the incoming transaction only belongs to the specified customer c. Figure 10 gives an example after sliding the CSW-BV for customer #3, where the incoming transaction is T_7, i.e., TID = 7.

Case (c): *A pattern is a node of T but not a node in T'.* In this case, the pattern becomes an infrequent pattern because of window CSW sliding. Hence, we directly delete the node and its sub-trees from the current LexSeq-Tree.

Pseudo codes of function LexSeq-Tree-Maintain are given in Figure 11. In the function LexSeq-Tree-Maintain, two cases and their corresponding procedures, called *LST-Generate* (LexSeq-Tree Generate) and *LST-Update* (LexSeq-Tree Update), are performed, respectively. First, function LST-Generate is executed when the new item is a new node of current LexSeq-Tree. Function LST-Generate uses S-step and I-step to generate all possible children based on the principles mentioned above for each node. If the child does not exist in the LexSeq-Tree, the function LST-Generate generates a new node for this child. Otherwise, it only updates the index set and support of this child node. At this time, function LST-Update checks each node to update its index set and support value.



CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)
2	4	(a, b, c)
3	5	(a, b)
1	6	(b, c, d)
3	*7*	*(b, c, d)*

Window size $N = 2$
Sliding window in each customer:
CID = 3 CID = 6
Customer #1: { (b, c, d) (b, c, d) }
CID = 2 CID = 4
Customer #2: { (b) (a, b, c) }
CID = 5 CID = 7
Customer #3: { (a, b) (b, c, d) }
The sub-trees of items {a, b} needs to generate candidates; others doesn't
The appended items are {b, c, d}

FIGURE 10. Reducing the generated candidates after sliding the CSW-BV of customer#3

```
Function LexSeq-Tree-Maintain (c, F): c: customer, F: a set of frequent items;
1:   foreach tree node n that's representing item $i \subseteq F$ do
2:       if item i does not exist in the customer-sequences of customer c then
3:           perform LST-Generate(c, n);
4:       else /* item i exists in the customer-sequence of customer c */
5:           perform LST-Update(c, n);
6:       endif
7:   endfor
```

FIGURE 11. Pseudo code of function *LexSeq-Tree-Maintain*

Algorithms *LST-Generate* and *LST-Update* are given in Figures 12 and 13, respectively. Whenever a new incoming transaction arrives, only one customer-sequence is modified. Each node of the current LexSeq-Tree only needs to update one position value of its index set. The summation of the weight of the customer-sequences with no modification only needs to be decayed by the used-defined decayed rate. Hence, we do not have to sum

```
Function LST-Generate ($c, n$)
1:  foreach existing child node $n'$ of $n$ do
2:      perform Support-Update ($c, n'$);
3:      if $n'.sup < minsup$ then
4:          drop node $n'$ and its sub-tree;
5:      endif
6:  endfor
7:  generate candidates of node $n$ by using S-step and I-step;
8:  foreach generated candidate $x$ of $n$ do
9:      count the support of $x$;
10:     if $x.sup \geq minsup$ then
11:         $x$ is a child node of $n$;
12:     endif
13: endfor
14: foreach child $n'$ of $n$ do
15:     perform LST-Generate ($c, n'$);
16: endfor
```

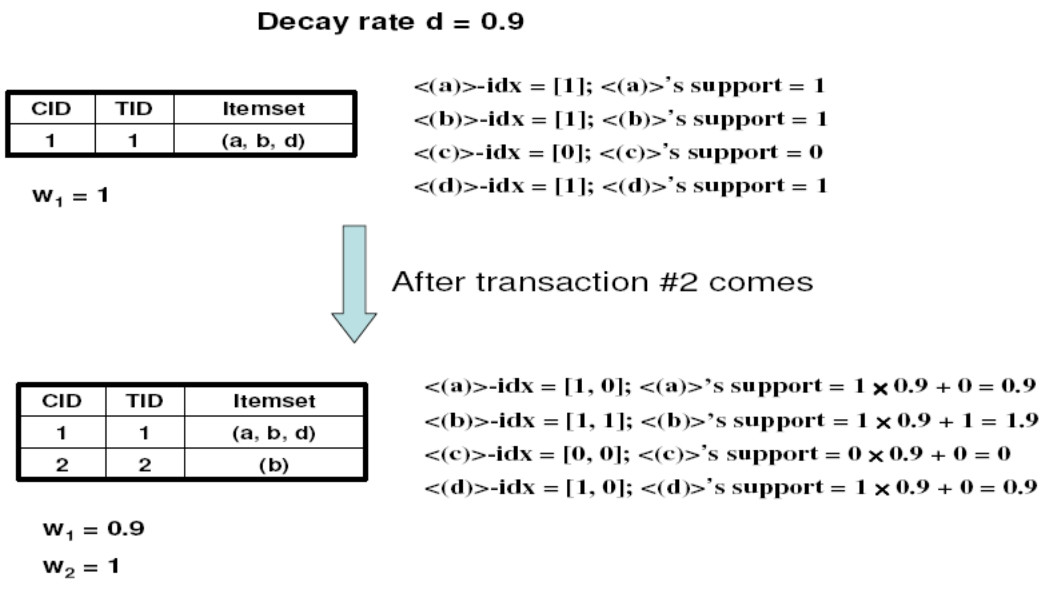
Figure 12. Pseudo code of function *ST-Generate*

```
Function LST-Update ($c, n$)
1:  foreach existing child node $n'$ of $n$ do
2:      perform Support-Update ($c, n'$);
3:      if $n'.sup < minsup$ then
4:          eliminate node $n'$ and its sub-tree;
5:      endif
6:  endfor
7:  foreach child $n'$ of $n$ do
8:      perform LST-Update ($c, n'$);
9:  endfor
```

Figure 13. Pseudo code of function *ST-Update*

```
Function Support-Update ($c, n$)
1:  if customer-sequence $c$ is a new sequence then
2:      decay the support of $n$;
3:      if the sequence of $n \subseteq c$ then $n.sup = n.sup + 1$;
4:      endif
5:  else
6:      if $\rho$-idx[c] == 0 then /* assume the sequence in $n$ is $\rho$ */
7:          decay the support of $n$ by a user-defined decayed rate;
8:          if the sequence of $n \subseteq c$ then $n.sup = n.sup + 1$;
9:          endif
10:     else
11:         $n.sup = n.sup$ – previous weight of $c$;
12:         if the sequence of $n \subseteq c$ then $n.sup = n.sup + 1$;
13:         endif
14:     endif
15: endif
```

Figure 14. Pseudo code of function *Support-Update*



CID	TID	Itemset
1	1	(a, b, d)

$w_1 = 1$

<(a)>-idx = [1]; <(a)>'s support = 1
<(b)>-idx = [1]; <(b)>'s support = 1
<(c)>-idx = [0]; <(c)>'s support = 0
<(d)>-idx = [1]; <(d)>'s support = 1

CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)

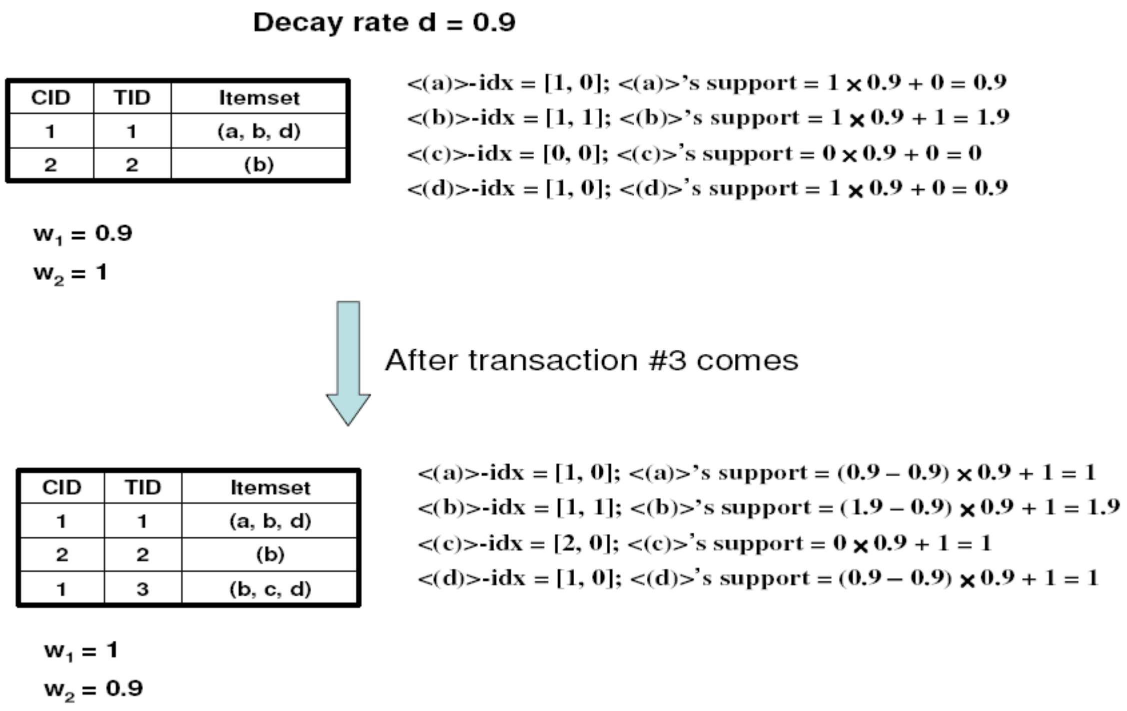
$w_1 = 0.9$
$w_2 = 1$

<(a)>-idx = [1, 0]; <(a)>'s support = 1 × 0.9 + 0 = 0.9
<(b)>-idx = [1, 1]; <(b)>'s support = 1 × 0.9 + 1 = 1.9
<(c)>-idx = [0, 0]; <(c)>'s support = 0 × 0.9 + 0 = 0
<(d)>-idx = [1, 0]; <(d)>'s support = 1 × 0.9 + 0 = 0.9

FIGURE 15. Example of update support in case (a)

up all the weights one by one. The pseudo code of function *Support-Update* is given in Figure 14. Furthermore, three cases of updating support are discussed as follows.

Case (a): *The incoming transaction belongs to a new customer.* In this case, the original support is decayed by a decay-rate for a sequence ρ. Hence, we check whether ρ exists in the new customer- sequence or not. If ρ does exist, we add the decayed support by one, i.e., $\rho.sup + 1$. Otherwise, the decayed support is not updated. An example of case (a) is given in Figure 15.



CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)

$w_1 = 0.9$
$w_2 = 1$

<(a)>-idx = [1, 0]; <(a)>'s support = 1 × 0.9 + 0 = 0.9
<(b)>-idx = [1, 1]; <(b)>'s support = 1 × 0.9 + 1 = 1.9
<(c)>-idx = [0, 0]; <(c)>'s support = 0 × 0.9 + 0 = 0
<(d)>-idx = [1, 0]; <(d)>'s support = 1 × 0.9 + 0 = 0.9

CID	TID	Itemset
1	1	(a, b, d)
2	2	(b)
1	3	(b, c, d)

$w_1 = 1$
$w_2 = 0.9$

<(a)>-idx = [1, 0]; <(a)>'s support = (0.9 – 0.9) × 0.9 + 1 = 1
<(b)>-idx = [1, 1]; <(b)>'s support = (1.9 – 0.9) × 0.9 + 1 = 1.9
<(c)>-idx = [2, 0]; <(c)>'s support = 0 × 0.9 + 1 = 1
<(d)>-idx = [1, 0]; <(d)>'s support = (0.9 – 0.9) × 0.9 + 1 = 1

FIGURE 16. Example of update support in case (b)

Case (b): *The incoming transaction belongs to an existing customer.* In this case, the previous position value of this customer in the ρ-idx has to be checked for a sequence ρ.

Assume that the modified customer-sequence is s. If the value of previous ρ-idx$[s]$ is zero, the original support is decayed by a user-specific decay-rate. Furthermore, the value is increased by one or zero by the existence of ρ in sequence s. If the value of the previous ρ-idx$[s]$ is not a value zero, the original support value subtracts the previous weight of sequence s. After that, the value is decayed by a decay-rate. Finally, we add the support value by one or zero by the same consideration as discussed in the case (a). An example for the case (b) is shown in Figure 16.

4. **Performance Evaluation.**

In this section, several experiments of the proposed single-pass algorithm IncSpam are discussed. These experiments are executed on a desktop with a 2.16GHz CPU and 2GB memory. The proposed algorithm IncSpam is implemented in C++ STL and compiled with gcc-4.0.3 on Linux 9.0. Moreover, the original synthetic data used in these experiments is generated by the IBM synthetic data generator [1]. We modified the synthetic data generator to simulate the environment of streaming itemset-sequences. Parameters of synthetic data used in these experiments are listed in Table 1. The performance measurements include memory usage and average time of window sliding. Experiment of memory usage is evaluated by a system tool to observe the real memory variation. Note that all the experiments are performed with a default decay-rate $d = 0.999$.

Table 1. Parameters of the synthetic data

Experimental Parameters	**Value**
Average Number of transactions per customer (C)	30
Average Number of items per transaction (T)	$2 \sim 3$
Number of Different Items (N)	1000
Default decay-rate (d)	0.999

4.1. **Varying different minimum supports for experiments of memory usage and execution time of IncSpam algorithm.**

In the first experiment of our IncSpam algorithm, an absolute user-defined minimum support threshold *minsup* is used for evaluating experiments of memory usage and average execution time. If the number of customers that support a sequence ρ is greater than or equal to *minsup*, ρ is a sequential pattern based on our problem definition, where minimum support threshold *minsup* is changed from 3 to 10. In these experiments, the total number of customer is 1,000, and the window size w of each customer is 10 as default experimental parameters.

First experiment is used to evaluate the memory usage of IncSpam algorithm under different minimum support thresholds. Figure 17 shows the memory usage of IncSpam algorithm while varying minimum support constraints from 0.3% to 1%. From this figure, we can find that the memory requirement of our algorithm is about 200MB. This is reasonable for mining sequential patterns from streaming itemset-sequences. Although the memory usage is a little higher when the minimum support constraint *minsup* is small, the next experiment as shown in Figure 18 will prove that IncSpam algorithm is a memory efficient method used in a streaming data environment. Figure 18 shows the experimental results for evaluating the relationship between maximum number of tree nodes from 1K to 10K and the memory requirement. From Figure 18, we can see that the relationship between maximum number of tree nodes and memory usage is a linear relationship. The linear growth means that the memory usage of IncSpam grows up only when the number of sequential patterns increases. Furthermore, our proposed algorithm does not use additional data structures when minimum support threshold becomes small.

Consequently, these experiments show that IncSpam algorithm is efficient in memory usage constraint for mining sequential patterns from streaming itemset-sequences. Third experiment of IncSapm algorithm is used to evaluate the average window sliding time of IncSpam under different minimum support thresholds. Figure 19 gives the average window sliding time of IncSpam algorithm from minimum support thresholds form 0.3% to 1%. The result shows that average sliding time of our IncSpam algorithm is below 1 second. This is because IncSpam algorithm uses CSW-BV and the characteristics of incremental mining techniques to speed up the processing time of each new incoming transaction. Consequently, the experiment of average sliding window as given in Figure 19 also shows the efficiency of our proposed algorithm.

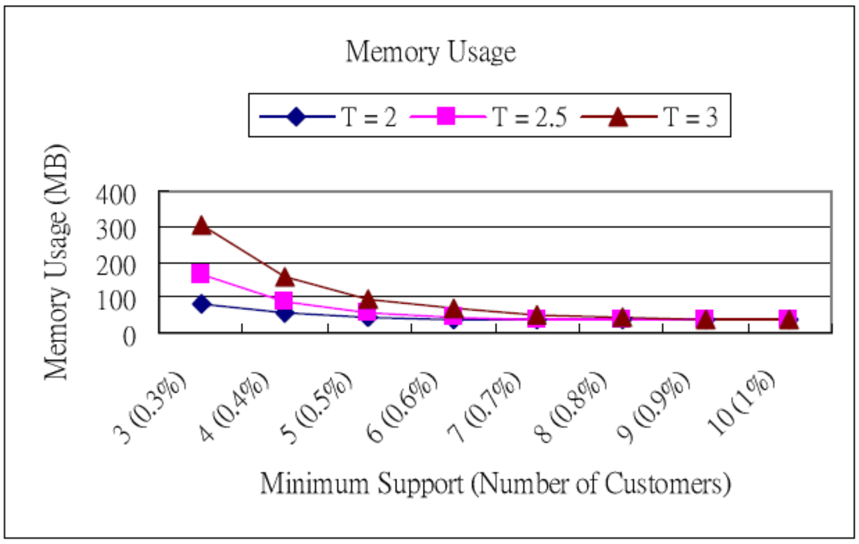


FIGURE 17. Memory usage of IncSpam under different minimum support thresholds from 0.3% to 1%

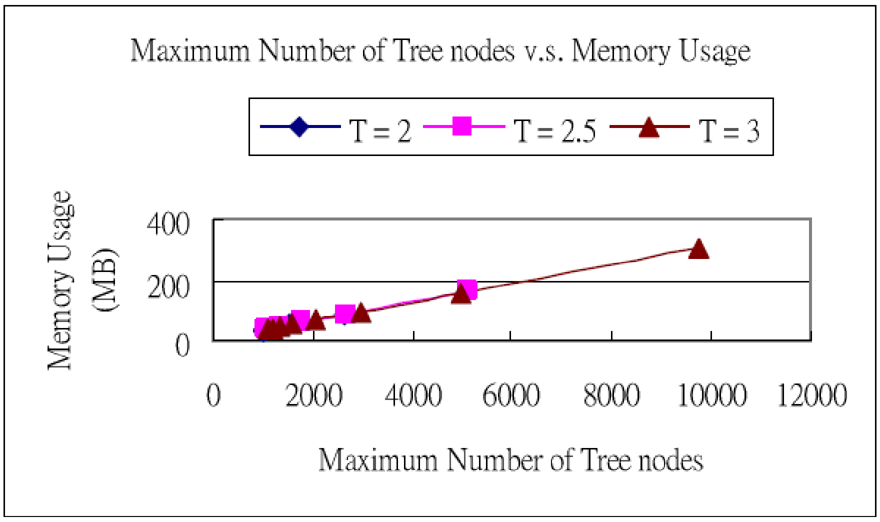


FIGURE 18. Performance relationship of IncSpam algorithm between the maximum number of tree nodes (from 1K to 10K) and the memory usage

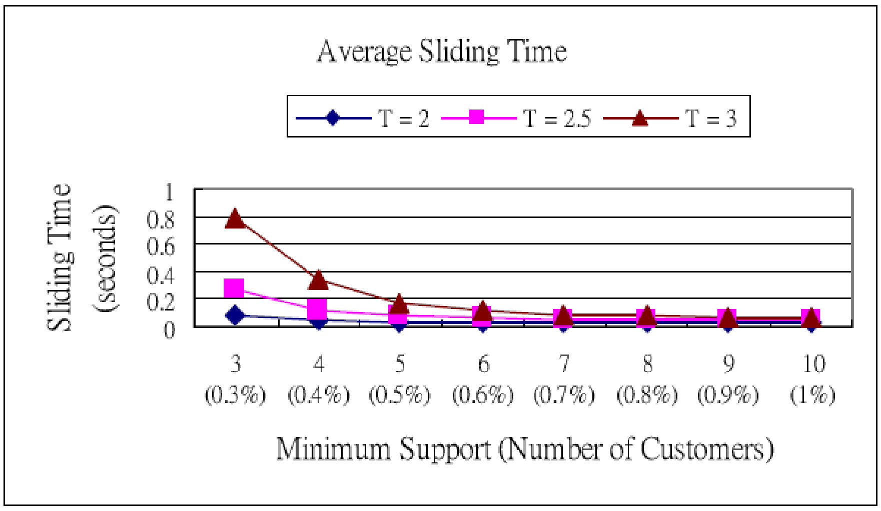


FIGURE 19. Average time of window sliding with different minimum support thresholds (from 0.3% to 1%)

4.2. **Varying different sliding window sizes for experiments of memory usage and execution time of IncSpam algorithm.** In this section, the performance of the IncSpam algorithm is evaluated by the experiments of memory requirement and execution time under different sizes of sliding windows from 10 transactions per customer to 25 transactions per customer. The size of sliding window w is used to control the number of transactions maintained by each customer. In these experiments, window size w ranges from 10 transactions to 25 transactions and the user-defined minimum support threshold $minsup$ is fixed to 10, i.e., ten customers. The experimental result of memory requirement of IncSpam algorithm under different window size from 10 transactions per customer to 25 transactions per customer is given in Figure 20. Another experiment of execution time of IncSpam under different window size is also evaluated in this section and the result is shown in Figure 21. From both figures, we can find that when the number of transactions maintained by each customer increases, the corresponding memory usage and average sliding time also grows up. Hence, the performance of IncSpam under different sliding window is also for mining sequential patterns from streaming itemset-sequences. Based on IBM synthetic dataset and several real-world applications, sliding window maintaining about 15 transactions for each customer is a reasonable choice. Consequently, our proposed IncSpam algorithm can be applied in real-world applications and is very efficient in memory usage requirement and real-time transaction processing.

4.3. **Varying different number of customers for experiments of memory usage and execution time of IncSpam algorithm.** In this section, the performance of the IncSpam algorithm is evaluated by the experiments of memory requirement and execution time under different number of customers from 1K to 5K. Based on the framework of IncSpam algorithm, a new customer can be inserted into the data structure LexSeq-Tree dynamically. In those experiments of Section 4.2, we fix the number of customers for observing performance conveniently. The memory usage and average sliding time of IncSpam algorithm for different number of customers are tested in these experiments. In these experiments, the user defined minimum support threshold $minsup$ is also 10 customers, i.e., $minsup$ =10.

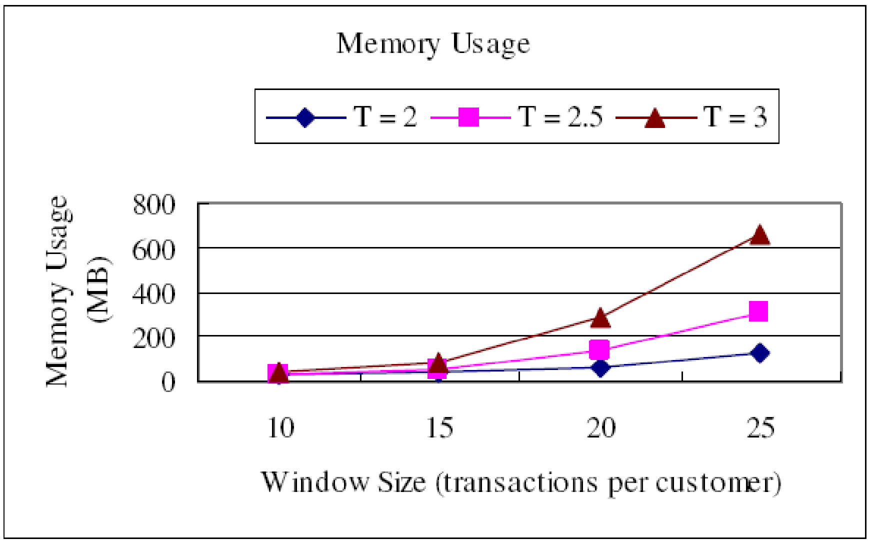


Figure 20. Memory usage of IncSpam algorithm with different size of sliding window (from 10 transactions per customer to 25 transactions per customer)

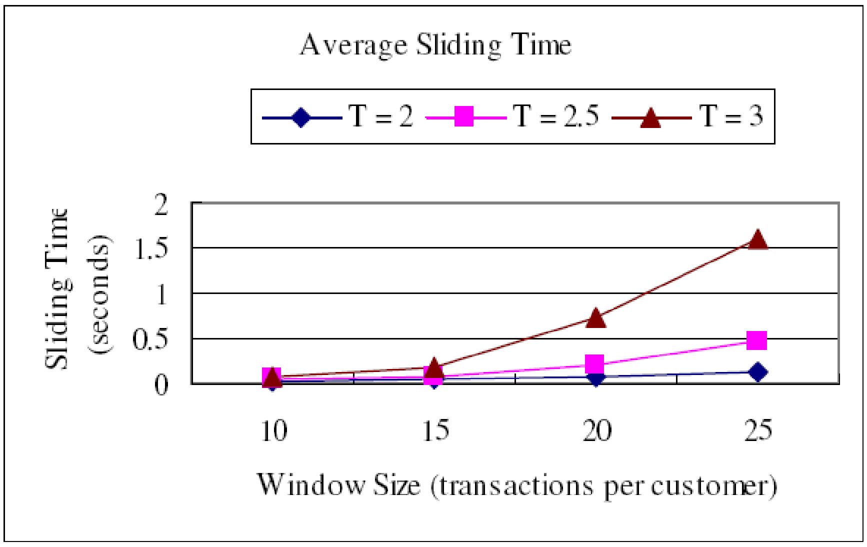


Figure 21. Average time of window sliding with different window size (from 10 transactions per customer to 25 transactions per customer)

Figure 22 shows the results of memory usage experiment of the IncSpam algorithm with different number of customers from 1K to 5K. From this figure, we can find that the performance relationship between memory requirement and the number of customers is also linear growth. Therefore, we can say that the proposed IncSpam algorithm can efficiently handle a great amount of customers with reasonable memory requirement. In the last experiment of this section, execution time performance of IncSpam is evaluated in Figure 23. In Figure 23, we can observe that the average sliding time of IncSpam is also in linear growth relationship under different number of customers from 1K to 5K. Consequently, we can find that our algorithm is an efficient method for mining sequential patterns from streaming itemset-sequences based on these experiments discussed in Section 4.

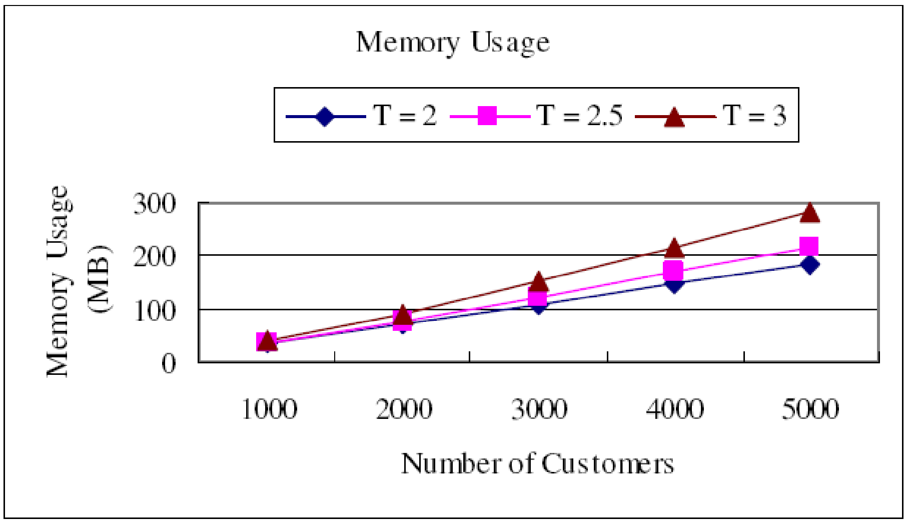


FIGURE 22. Memory usage of IncSpam algorithm with different number of customers from 1K to 5K

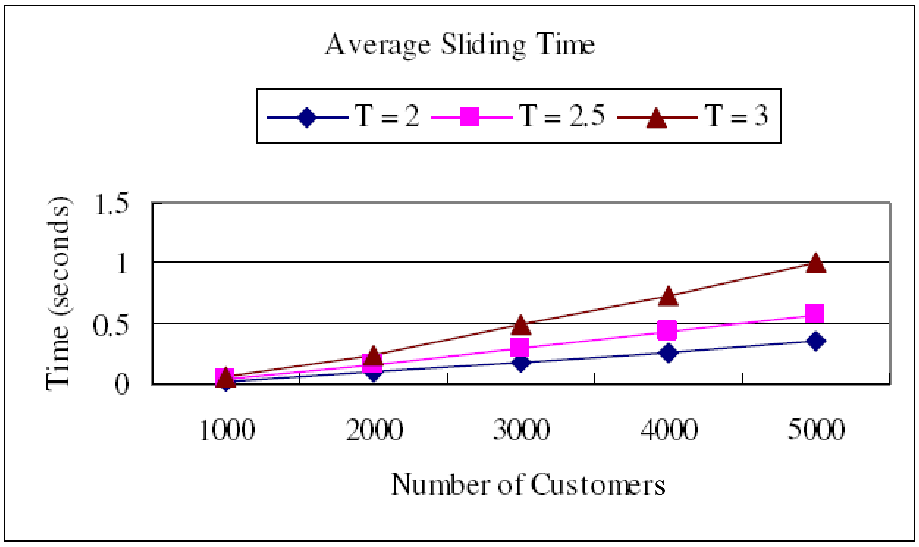


FIGURE 23. Average sliding time of IncSpam with different number of customers from 1K to 5K

5. **Conclusions and Future Works.** In this paper, a new research issue of data mining, i.e., *mining sequential patterns from streaming itemset-sequences*, is defined. We propose an efficient single-scan mining algorithm, called *IncSpam*, for discovering sequential patterns from streaming itemset-sequences. Based on our best knowledge, our algorithm IncSpam is the *first one-pass approach* for mining sequential patterns from streaming itemset-sequences, not just streaming item-sequences or static datasets. In the framework of IncSpam, an effective bit-sequence representation of stream sliding window model *CSW-BV* and an extended lexicographic tree-based data structure *LexSeq-Tree* are developed to reduce the memory requirement of the online maintenance of sequential patterns generated from data streams. Experimental results show that our algorithm IncSpam is an efficient method for mining sequential patterns from steaming itemset-sequences

over customer sliding windows. Future works of this study include mining top-K sequential patterns, closed sequential patterns and maximal sequential patterns from streaming itemset-sequences.

Acknowledgment. This work is partially supported by National Science Council of Taiwan with research grant no. NSC 99-2221-E-424-005-. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] R. Agrawal and R. Srikant, Mining sequential patterns, *Proc. of the 7th International Conference on Data Engineering*, pp.3-14, 1995.

[2] J. Ayres, J. Gehrke, T. Yiu and J. Flannick, Sequential pattern mining using a bitmap representation, *Proc. of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pp.429-435, 2002.

[3] J. H. Chang and W. S. Lee, Decaying obsolete information in finding recent frequent itemsets over data stream, *IEICE Transactions on Information and System*, vol.E87-D, no.6, pp.1588-1592, 2004.

[4] J. H. Chang and W. S. Lee, Efficient mining method for retrieving sequential patterns over online data streams, *Journal of Information Science*, vol.31, no.5, pp.420-432, 2005.

[5] G. Chen, X. Wu and X. Zhu, Sequential pattern mining in multiple data streams, *Proc.of the 5th IEEE International Conference on Data Mining*, pp.585-588, 2005.

[6] A. Chen and H. Ye, Multiple-level sequential pattern discovery from customer transaction databases, *International Journal of Computational Intelligence*, vol.2, no.1, pp.64-75, 2004.

[7] H. Cheng, X. Yan and J. Han, IncSpan: Incremental mining of sequential patterns in large database, *Proc. of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, pp.97-121, 2004.

[8] H.-J. Do and J.-Y. Kim, Clustering categorical data based on combinations of attribute values, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(A), pp.4393-4406, 2009.

[9] M. EI-Sayed, E. A. Rundensteiner and C. Ruiz, FS-Miner: Efficient and incremental mining of frequent sequence patterns in web logs, *Proc. of the 6th ACM International Workshop on Web Information and Data Management*, pp.128-135, 2004.

[10] L. Golab and M. T. Özsu, Issues in data stream management, *ACM SIGMOD Record*, vol.32, no.2, pp.5-14, 2003.

[11] K.-F. Jea, K.-C. Lin and I.-E. Liao, Mining hybrid sequential patterns by hierarchical mining technique, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2351-2367, 2009.

[12] B. Kao, M. Zhang, C.-L. Yip, D. W. Cheung and U. Fayyad, Efficient algorithms for mining and incremental update of maximal frequent sequences, *Journal of Data Mining and Knowledge Discovery*, vol.10, no.2, pp.87-116, 2005.

[13] H.-F. Li, M.-K. Shan and S.-Y. Lee, DSM-FI: An efficient algorithm for mining frequent itemsets in data streams, *Knowledge and Information Systems*, vol.17, no.1, pp.79-97, 2008.

[14] M.-Y. Lin and S.-Y. Lee, Incremental update on sequential patterns in large databases by implicit merging and efficient counting, *Information Systems*, vol.29, no.5, pp.385-404, 2004.

[15] M.-Y. Lin and S.-Y. Lee, Fast discovery of sequential patterns through memory indexing and database partitioning, *Journal of Information Sciences and Engineering*, vol.21, no.1, pp.109-128, 2005.

[16] A. Marascu and F. Masseglia, Mining sequential patterns from temporal streaming data, *Proc. of the 1st ECML/PKDD Workshop on Mining Complex Data*, 2005.

[17] S. Parthasarathy, M. J. Zaki, M. Ogihara and S. Dwarkadas, Incremental and interactive sequence mining, *Proc. of the 8th International Conference on Information and Knowledge Management*, pp.251-258, 1999.

[18] J. Pei, J. Han, B. Mortazavi-Asl and H. Pinto, PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth, *Proc. of the 17th International Conference on Data Engineering*, pp.215-224, 2001.

[19] C. Raissi, P. Poncelet and M. Teisseire, SPEED: Mining maximal sequential patterns over data streams, *Proc. of the 3rd International IEEE Conference Intelligent Systems*, pp.546-552, 2006.

[20] Z. Sui, Ya. Liu and Y. Hu, Extracting hyponymy relation between Chinese terms based on term types' commonality and sequential patterns, *ICIC Express Letters*, vol.3, no.4(B), pp.1233-1238, 2009.

[21] Y. Wang, W. Zhang, C. Zhang, H. Ling and F. Zhao, Establishing website navigation support systems by mining sequential patterns, *ICIC Express Letters*, vol.4, no.2, pp.395-400, 2010.

[22] X. Yan, J. Han and R. Afshar, CloSpan: Mining closed sequential patterns in large datasets, *Proc. of 2003 SIAM International Conference on Data Mining*, pp.166-177, 2003.

[23] M. Zhang, B. Kao and C. L. Yip, A comparison study on algorithms for incremental update of frequent sequences, *Proc. of the 2002 IEEE International Conference on Data Mining*, pp.554-561, 2002.

[24] Y. Zhu and D. Shasha, Statstream: Statistical monitoring of thousands of data stream in real time, *Proc. of the 28th International Conference on Very Large Data Bases*, pp.358-369, 2002.