Sādhanā Vol. 41, No. 12, December 2016, pp. 1369–1380
DOI 10.1007/s12046-016-0567-7

A sequential tree approach for incremental sequential pattern mining

RAJESH KUMAR BOGHEY[1,*] and SHAILENDRA SINGH[2]

[1]Department of Computer Science and Engineering, Technocrats Institute of Technology (Excellence), Bhopal 462021, India
[2]Department of Computer Engineering and Application, National Institute of Technical Teachers Training and Research, Bhopal 462002, India
e-mail: rajeshboghey@gmail.com; ssingh@nitttrbpl.ac.in

MS received 21 October 2015; revised 4 May 2016; accepted 26 May 2016

Abstract. "Sequential pattern mining" is a prominent and significant method to explore the knowledge and innovation from the large database. Common sequential pattern mining algorithms handle static databases. Pragmatically, looking into the functional and actual execution, the database grows exponentially thereby leading to the necessity and requirement of such innovation, research, and development culminating into the designing of mining algorithm. Once the database is updated, the previous mining result will be incorrect, and we need to restart and trigger the entire mining process for the new updated sequential database. To overcome and avoid the process of rescanning of the entire database, this unique system of incremental mining of sequential pattern is available. The previous approaches, system, and techniques are a priori-based frameworks but mine patterns is an advanced and sophisticated technique giving the desired solution. We propose and incorporate an algorithm called STISPM for incremental mining of sequential patterns using the sequence tree space structure. STISPM uses the depth-first approach along with backward tracking and the dynamic lookahead pruning strategy that removes infrequent and irregular patterns. The process and approach from the root node to any leaf node depict a sequential pattern in the database. The structural characteristic of the sequence tree makes it convenient and appropriate for incremental sequential pattern mining. The sequence tree also stores all the sequential patterns with its count and statistics, so whenever the support system is withdrawn or changed, our algorithm using frequent sequence tree as the storage structure can find and detect all the sequential patterns without mining the database once again.

Keywords. Data mining; STISPM; sequential tree; incremental mining; backward tracking.

1. Introduction

Sequential pattern mining is an important and significant breakthrough innovation in data mining. The task of sequential pattern mining is to find all sequential patterns in the sequence database. It is widely used mainly in transactional data analysis, web log analysis, customer purchase behavior analysis and its pattern, weather prediction, and among others. Sequential pattern mining was first introduced by Agrawal and Srikant [1] as "AprioriAll."

Sequential Pattern Mining is bonded together and related to association rule mining and they are at par with an exception of time factor, which links the events in sequential pattern mining. Sequential patterns indicate the link and relationship between transactions, while association rule depicts and denotes intratransaction relationships. Association rule mining delivers and performs such mining results that are about the items brought together frequently and also those items must be the part of the same transaction. Sequential pattern mining precisely performs and maintains the results in the chronology that are purchased by the same customer at different times in various transactions executed by him or her [2].

Earlier, many techniques, research work, and systems for mining Sequential Pattern for transactions were proposed, most of which were based on the A priori-based mining algorithm. The shortcomings of a priori-based mining algorithms of sequential patterns are that multiple scanning of the database and a large number of candidate sequences are needed to be stored. In order to minimize the large set of candidate sequences, Han *et al* [3] proposed a pattern-growth mining method based on projection, called FreeSpan. FreeSpan used the advantages of projected database to minimize the search space.

Common sequence mining algorithms manage and maintain a static database, which means the data in the

*For correspondence

database will not change. Practically speaking, in real-world applications and executions, new transactions are generally incorporated into databases on their rising or increasing pattern. In this case, the originally desired and expected frequent sequences may become invalid and new frequent sequences may appear in the resulting and subsequent updated databases [4–7]. Thus, designing an efficient and effective algorithm that can maintain and manage sequential pattern as the database keeps growing is critical. In view of this, the concept of incremental mining of sequential patterns is proposed and made available for innovation and desired results. The concept of incremental mining algorithms uses the intermediate data collected and accumulated from the previous mining process. It achieves the knowledge and pattern extraction by implementing the latest revised and updated part of the database and thereafter refreshes the older database, as and when required.

In this paper, a sequential tree approach for incremental sequential pattern mining is presented to find out and locate the sequences that are frequent in the updated and revised database. The proposed approach and mechanism uses tree space structure with the depth-first and backward tracking approach. Also, the proposed method gets rid of such patterns that are infrequent using dynamic lookahead pruning method. The path from the root node to any leaf node denotes a sequential pattern in the database. The structural characteristic and features of sequence tree make it appropriate for incremental sequential pattern mining.

The remaining part of the paper is organized in the following order. Section 2 reviews the related algorithms. Section 3 describes the concepts of the structure of sequence tree space. Section 4 describes the sequential pattern and incremental mining. Section 5 illustrates a unique algorithm for sequential patterns mining and theoretical foundation for our approach. Section 6 provides an example to illustrate the proposed algorithm. Section 7 elaborates the performance evolutions and test results. Finally, Section 8 presents the conclusions.

2. Related work

In 1995, Agrawal and Srikant [1], through their research, introduced the mining sequential patterns for the first time. Since then there have been many types of research and innovations in this field and several efficient methods have been initiated and developed to explore and search the frequent sequence in a set of data sequence that includes a series of transactions, happening or occurring chronologically. There are two main approaches or methodologies in sequential pattern mining: mining with a static database called sequential pattern mining and mining on an updated database called incremental mining.

2.1 *Algorithms for discovering sequential patterns*

AprioriAll: Agarwal and Srikant proposed the AprioriAll [1] algorithm, which splits sequential pattern mining into three phases: (1) itemset, (2) transformation, and (3) sequence.

The itemset phase uses AprioriAll to find all frequent itemsets. The second step of transformation phase transforms the database with each transaction being substituted by the set of all frequent itemsets in that transaction. In the third and final phase, AprioriAll multiple passes over the database to generate candidates and to count the support of candidates.

Generalized Sequential Patterns (GSP): Srikant and Agarwal introduced GSP [8] algorithms that compose several passes over the database and discover frequent *k*-sequences at *k*th database scan. At each pass, every data sequence is checked to update the support counts of the candidate contained in this sequence. Initially, each item is a candidate 1-sequence for the first pass. Frequent 1-sequences are determined after checking and monitoring all the data sequences in the database. In the subsequent passes, frequent (*k*-1) sequences are self-joined to produce candidate sequences. Thereafter, the supports of these candidate sequences are counted by examining all data sequences, and then those candidates having a minimum support become the frequent sequences. This process stops when there is no more candidate sequence.

Sequential Pattern mIning with Regular expressIon constraints (SPIRIT): Garofalakis *et al* [9] proposed a family of algorithms for sequential pattern mining with regular expression constraints. Its general idea is to use some relaxed constraint that has excellent characteristics to reduce or prune. There are many versions of the algorithm that differ in the volume and magnitude to which the constraints are enforced to prune the search space of patterns during the computation. The main distinguishing factor among the schemes is the degree to which the regular expression constraints are enforced to prune the search space.

Despite this, all the algorithms discussed above have to re-mine the database. After the database is appended with new data sequences, it is required repeat mining for maintaining the rules discovered earlier and also for discovering and exploring new rules. Re-mining requires more time than the earlier mining process because the appending increases the size of the database. Next, we explore some approaches for updating patterns without re-mining.

2.2 *Approaches for incremental mining of sequential pattern and association rule*

The incremental mining algorithms [4, 10] generally plan to use intermediate information collected during the earlier mining process. Satisfactory research on incremental mining has ongoing since several years. We introduce some previous work here, which helps in discovering and exploring the sequential pattern.

Update with Early Pruning (UWEP): UWEP [11] made use of a dynamic lookahead strategy in the incremented database to refresh the frequent item sets and interactively mined the association rules. It also follows the approaches of FUP2 [4] and partition update [12] algorithm. The major benefits of UWEP [11] were to refresh or scan the old database not more than once and the incremented database once and it evaluates the least number of candidates to find out the new frequent patterns. In the first scan of the database, a Tidlist is formed for each item in the database and uses Tidlist to find out the support of supersets of that item. Also, it removes the item sets that are likely to get infrequent as early as possible, since it uses a dynamic lookahead strategy. This lookahead pruning leads to the reduction of candidates to a minimum in the new incremented database. Also, UWEP (Ayan *et al* 1999) encourages such candidates that are in the old and incremented database. Thus the majority of candidate sets are removed just because of their absence in the incremented set of transactions and this is done without refreshing the existing database.

Fast Sequential Pattern Update Algorithm (FASTUP): Lin and Lee [5] introduced incremental sequential pattern mining. FASTUP is an enhanced GSP [8] taking into account the previous mining result, before generating and validating candidates, using the generating pruning method. The main idea is that FASTUP, by means of the previous result, takes advantage of information about sequence thresholds to generate candidates; it can, therefore, avoid generating some sequences, depending on their support. Item constraint: An item constraint specifies a subset of items that should or should not be present in the patterns.

Incremental Mining of Sequential Patterns (IncSpan): Cheng *et al* [13] developed a method for incremental mining sequential patterns and explores buffering semi-frequent patterns in sequential patterns mining, which is a statistics-based approach. A sequence is semi-frequent if its support is less than min sup but no less than μ*minimum support and a sequence is infrequent if its support is less than μ*minimum support. The SFS forms a kind of boundary (or "buffer zone") between the frequent subsequences and infrequent subsequences. IncSpan uses two optimization techniques: one is reverse pattern matching and the other is shared projection.

Progressive mIning of Sequential pAtterns (Pisa): PISA [14] discovered sequential patterns in defined time period of interest (POI). POI is a sliding window, whose length is a user-specified time interval, continuously advancing as the time goes by. The basic idea behind Pisa is to maintain progressive sequential tree to keep the information from one POI to another. Pisa utilizes a progressive sequential tree to efficiently maintain the latest data sequences, discover the complete set of up-to-date sequential patterns, and delete obsolete data and patterns accordingly. By changing Start time and End time of the POI, Pisa can easily deal with a static database and an incremental database as well.

An incremental mining algorithm for maintaining sequential patterns using pre-large sequences: Hong *et al* [15] developed a novel and an efficient incremental mining algorithm capable of updating sequential patterns based on the concept of pre-large sequences. A pre-large sequence is not truly large, but approximately large. A lower support threshold and an upper support threshold are used to realize this concept. Pre-large sequences act like buffers and are used to reduce the movement of sequences directly from large to small and vice versa during the incremental mining process. The proposed algorithm does not require rescanning original databases until the accumulative amount of newly added customer sequences exceeds a safety bound, which depends on database size. The safety bound also increases monotonically along with the increase in database size.

Pei *et al* later proposed PrefixSpan [16] for dealing with the problem of FreeSpan [3], that is, the length of original database sequences could not be shortened during the mining process. Their principle is to check the frequency of patterns from the prefix subsequences. For those prefixes that pass the frequency threshold, the suffix sub-sequences of each original sequence would be inserted into their projected database. Each projected database would then generate its frequent patterns recursively. The advantage of partitioning the search space into the projected database is that each projected database would contain only the required mining information with respect to the prefix. Along with the growth of the frequent patterns, the projected database will shrink, making Prefix Span perform better than FreeSpan for dense databases. PrefixSpan also outperforms FreeSpan in general. Since the generated projected databases occupy a lot of memory space, when the database or the number of items is huge, the memory space may be insufficient to store the projected databases.

Sequential PAttern Mining (SPAM): Ayres *et al* [17] proposed SPAM algorithm for finding all frequent sequences within a transactional database. The algorithm is especially efficient when the sequential patterns in the database are very long. A depth-first search strategy is used to generate candidate sequences, and various pruning mechanisms are implemented to reduce the search space. The transactional data are stored using a vertical bitmap representation, which allows for efficient support counting as well as significant bitmap compression.

Sequential PAttern Discovery using Equivalence classes (SPADE): Zaki [18] proposed an efficient sequential pattern mining algorithm based on vertical format. It utilizes combinatorial properties to decompose the original problem into smaller sub-problems. SPADE use a lattice-theoretic approach to decompose the original search space (lattice) into smaller pieces (sub-lattices), which can be processed independently in main memory. All sequences are discovered in three database scans, or only a single scan with some Pre-processed information.

Incremental Sequence Mining (ISM): Parthasarathy *et al* proposed ISM (Parthasarathy *et al* 1999) algorithm is actually an extension of SPADE [18]. ISM uses concepts of negative border and an efficient memory management scheme that create an Increment Sequence Lattice (ISL). It exploits its properties to prune the search space for potential new sequences. The ISL consists of all elements in the negative border and the frequent set and is initially constructed using SPADE [18]. In the ISL, each node of the ISL contains the support for the given sequence. The algorithm consists of two phases. Phase 1 is for updating the supports of elements in negative border and frequent sequence and Phase 2 is for adding to the negative border and frequent sequence beyond what was done in phase.

Frequent Pattern Projected Sequential Pattern Mining (FreeSpan): Han *et al* [3] developed FreeSpan algorithm to mine sequential patterns by a database projection technique. FreeSpan first finds the frequent items after scanning the database once. The sequence database is then projected, according to the frequent items, into several smaller intermediate databases. Finally, all sequential patterns are found by recursively growing subsequence fragments in each database.

Han *et al* [19] proposed the Frequent-Pattern-tree (FP-tree) structure for efficiently mining association rules without generation of candidate itemsets. The FP-tree was used to compress a database into a tree structure, which stored only large items. It was condensed and complete for finding all the frequent patterns. The construction process was executed tuple by tuple, from the first transaction to the last. After that, a recursive mining procedure called FP-Growth was executed to derive frequent patterns from the FP-tree. They demonstrated that the approach could have a better performance than Apriori. The FP-tree mining approach belongs to batch mining; that is, all transactions must be processed in a batch way.

Lin *et al* [20] attempt to further modify the FUFP-tree [21] algorithm for incremental mining based on the pre-large concept [15]. Based on two support thresholds, the proposed approach can effectively handle cases in which item sets are small in an original database but large in newly inserted transactions. The proposed algorithm does not require rescanning the original databases to construct the FUFP tree until a number of new transactions have been processed. The number is determined from the two support thresholds and the size of the database.

3. The structure of sequence tree space

Tree space structure is an approach to representing the database of sequential pattern. This sequence tree not only stores the frequent sequences of the original database, but also stores the non-frequent sequences of the original database, and the support of each sequence is also stored in the sequence tree. The construction of tree space structure begins from the root node and the number of edges is equal to the number of items in the database and the weight of the edges is equal to the number of data item values, i.e., X_1, X_2, X_3, …, X_n. In the next step, we take X_2, X_3, X_4, …, X_n and use the concept of backtracking to find out the possible sequence pattern and so on for every initial edge. Thus, we give the definition of the sequence tree as follows:-

Definition 3.1: The root node of the sequence tree is an empty node. In addition to the root node, each node in the sequence tree contains two attributes: One stores sequence in the database and the other stores the support of the sequence. The path from the root node to any leaf node represents the largest sequence in the database. The support of any node is not smaller than the support of its child nodes (table 3).

We give an example to illustrate the sequence tree space. A Sequence Database D for table 1 is shown in table 2. The sequence tree of the sequence database D is shown in figure 1. From the sequence tree shown in figure 1 we can see that the set of all sequences in the sequence database D with its support is {(10):4, (20):4, (30):3, (10 20):0, (10)(20):4, (10 30):0, (10)(30):3, (20 30):1, (20)(30):2, (10 20 30):0, (10 20)(30):0, (10)(20)(30):2, (10)(20 30):1}. The sequence tree stores all the frequent sequences and nonfrequent sequences with its support in the original database. So, we can find all the sequential patterns in the sequence database through traversing the sequence tree. The quoting of sequence tree space in STISPM makes the algorithm take full advantage of the results of previous mining. When the database is updated, STISPM can find all the sequential patterns in the database through traversing the sequence tree without mining the

Table 1. Database of customer item purchased.

Transaction Id	Transaction time	Items bought
C1	Jan 10, 2012	10
C1	Jan 20, 2012	20
C1	Jan 25, 2012	30
C2	Jan 15, 2012	10
C2	Jan 20, 2012	20,30
C3	Jan 25, 2012	10
C3	Jan 28, 2012	20
C3	Jan 30, 2012	30
C4	Jan 20, 2012	10
C4	Jan 25, 2012	20

Table 2. Sequence database representation.

Transaction Id (TID)	Item sequence
C1	((10) (20) (30))
C2	((10) (20, 30))
C3	((10) (20) (30))
C4	((10) (20))

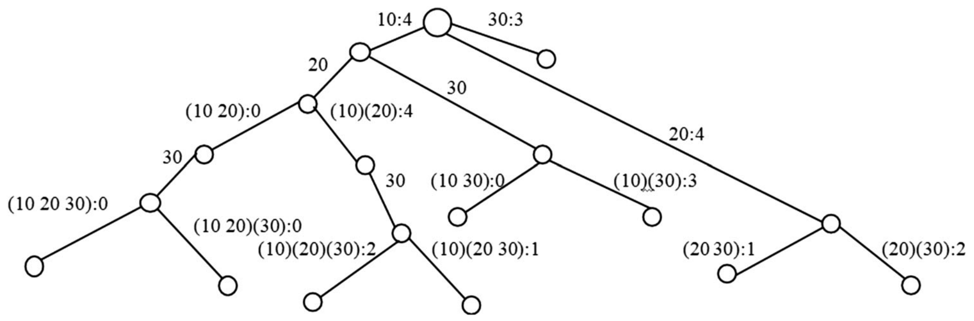

Figure 1. Sequence tree space structure.

database once again. We propose the algorithm that uses the sequence tree, called Tree_D. We use Tree_Generate (D) to construct the sequence tree for the original database, and when the database is updated, the Tree_d tree is generated again using Tree_Generate (d).

4. Problem formulation

4.1 *Mining of sequential patterns*

Let 'DB' be a set of customer transactions, where each transaction T consists of Transaction-id, Transaction time, and a set of items involved in the transaction. Let $I = \{i_1, i_2 \ldots i_m\}$ be a set of literals called items. An itemset is a nonempty set of items. A sequence s is a set of itemsets, ordered according to their time stamp, denoted by $<s_1 \text{-} s_2 \cdots s_n>$, where s_i is an itemset. The size of sequence s, written as |s|, is the total number of items in all the elements in s. Sequence s is a k-sequence if $|s| = k$.

For example, <(5)(7)(8)>, <(1,2)(4)>, and <(3)(5,9)> are all 3-sequences. A sequence $s = <s_1 s_2 \cdots s_n>$ is a subsequence of another sequence $s' = <s_1' s_2' \ldots s_m'>$ if there exists $1 \leq i_1 < i_2 < \ldots < i_n \leq i_m$ such that $s_1 \subseteq s'_{i1}, s_2 \subseteq s'_{i2, \ldots,}$ and $s_n \subseteq s'_{in.}$ Sequence s' contains sequence s if s is a subsequence of s'. For example, <(1)(2,5)> is a subsequence of <(1)(2,3,5)(6)>.

Every sequence in the sequence database 'DB' is referred to as a data sequence. Each data sequence is associated with a transaction-id (cid). The number of data sequences in 'DB' is represented by |DB|. The support of sequence s; denoted by s.sup, is the number of data sequences that contains divided by the total number of data sequences in DB. The minsup is the user specified minimum support threshold. A sequence s is a frequent sequence, or sequential pattern, if s.sup $\geq$ minsup.

4.2 *Incremental mining of sequential pattern*

The database used in mining for knowledge discovery is dynamic in nature. The database is updated with new transactions after the pattern mining process. This kind of update is called incremental update. The knowledge discovered from these databases is also dynamic. The process to find out the new set of all sequential patterns after database update is known as incremental mining of sequential pattern. The objective of incremental mining is to avoid re-learning of rules from the old data and apply knowledge that has already been discovered.

Let D be the original database and *minsup* the minimum support.

Let d be the increment database where new transactions or new customers are added to D.

Assume that each transaction on d is sorted by customer-id and transaction time.

$D + d = D \cup d$ is the updated database that contains all sequences from D and d.

Let L_D^K be the set of K frequent sequences in D.

The Problem of incremental mining of sequential patterns is to find K frequent sequences in D + d, noted L_{D+d}^K, with respect to the same minimum support.

5. The proposed algorithm

However, a number of algorithms have been proposed for the maintenance of sequential patterns over the static database using sequence tree space representation of the database. But not much work has been done on the incremental mining of sequential patterns which process the database in the form of sequence tree space structure .The majority of methods for incremental mining of sequential pattern require multiple scans of transactions in the original database and generate a huge number of candidate sequences to discover sequential patterns for the updated database. In this paper, we propose an efficient incremental mining algorithm for computing the sequential patterns without candidate generation after a nontrivial number of new transactions are added to the original database. Assume that the minimum support remains the same (table 3).

5.1 *An overview*

The proposed algorithm is a sequential tree approach for incremental sequential pattern mining (STISPM) that

Table 3. Notations used in the paper.

Notation	Definition
TID	Transaction identification number
D	Set of old transaction
d	Set of new transaction
D + d	Set of transaction after update
N	Length of maximal sequences in L_{D+d}
minsup	Minimum support threshold
$Supp_D(X)$	Support of X in D
$Supp_d(X)$	Support of X in d
C_D^K	Candidate K-sequences in D.
C_d^K	Candidate K-sequences in d.
Tree_D	Sequence tree of set of old transaction in Database D
Tree_d	Sequence tree of set of new transaction in Database d
L_d^K	Frequent K-sequences in d and D + d
L_D^K	Frequent K-sequences in D
T_d^K	Frequent K-sequences in d
L_{D+d}^K	Frequent K-sequences in D + d
L_{D+d}	Set of frequent sequences in D + d
L_D	Set of frequent sequences in D
P_set	Frequent sequences in D which are not frequent in d.
R_set	Frequent sequences in d which are not frequent in D.
S_{D+d}^m	Subset of m-frequent sequences in D + d
L_{D+d}^m	Maximal frequent sequences in D + d

follows the approaches of UWEP [11] and AprioriAll [1]. The advantages of STISPM are that it does not scan the existing database and it does not generate candidate sequences in order to determine the new set of frequent sequences. Moreover, it prunes sequences that are not frequent from the set of generated sequences as early as possible by a dynamic lookahead pruning strategy [11]. Every time we find those sequences that are frequent in added transactions but not frequent in the whole database and remove all its superset from the existing frequent sequences. In this way, many sequences that are not frequent in the updated database are pruned in early stages. However, discovering the frequent sequences is a nontrivial issue, where the efficiency of an algorithm is very high because this algorithm does not generate candidate sequences. A sequence is called k-sequences if the number of items in the sequences is k. We denote L^K as the set of frequent K-sequences. Candidate sequences are those that are potentially frequent sequences. Let C^K denote the set of candidate K-sequences. The various notations used in the algorithm are shown in table 3.

5.2 *The STISPM algorithm*

This algorithm STISPM is presented in figure 2. Inputs to the algorithm are Tree_D, Tree_d, minsup and N. The output of the algorithm is L_{D+d}^m, the set of frequent sequences in updated database.

```
STISPM (Tree_D, Tree_d, minsup, N)
1   K=1
2   While K< = N
3   $T_d^K$= All K- sequences in Tree_d with support ≥ minsup
4   $L_D^K$= All K- sequences in Tree_D with support ≥ minsup
5   P_set = $L_D^K - T_d^K$
6   If P_set ≠ϕ then
7   Prune (P_set)
8   End if
9   R_set = $T_d^K - L_D^K$
10  $L_d^K = T_d^K$– R_Set
11  $L_{D+d} = L_{D+d} \cup L_d^K$
12  If R_set ≠ ϕ then
13  Prune_R (R_set)
14  End if
15  K = K+1
16  End while
17  K= N (Length of Maximal sequences in $L_{D+d}$)
18  For K > 1
19  For each K-sequences do
20  delete all subsequences from $L_{D+d}$
21  End for
22  End for
```

Figure 2. Algorithm STISPM.

The algorithm can be described in the following steps

Step I:	For K = 1, find all the frequent K-sequences from Tree_d and add them in $T_d{}^K$ shows by Line 3 of figure 2.
Step II:	Find all the frequent K-sequences from Tree_D and add them in L_D^K shows by Line 4 of figure 2.
Step III:	Find P_set (collection of those K-sequences which are frequent in D but not frequent in d) then call Prune procedure (see figure 3 discussed in detail later), that will remove those K-sequences and their supersets, which are frequent in D but not frequent in d and D + d. Lines 5–8 of figure 2 shows this step.
Step IV:	Find R_set (collection of those K-sequences, which are frequent in d but not frequent in D). Generate a set of sequences L_d^K, which are frequent in D and d with the help of R_Set and T_d^K.After that Call Prune_R procedure (see figure 4 discussed in detail later), to find the frequentness of the K-sequences in the whole database (D + d), which are frequent in d but not frequent in D, if it is frequent in (D + d) then add the K-sequences in the set of frequent sequences of D + d (L_{D+d}) and also add it in the set of frequent K-sequences that are frequent in both D and d (L_d^K). Lines 9–14 of figure 2 show this step.
Step V:	Increment K, then obtained in the previous steps, shown by line 15 of figure 2.
Step VI:	Repeat Steps II–VI till K = N.

Step VII: Find the length (n) of maximal sequence in the (L_{D+d}). Delete all sub-sequences of length (1 to *n*) from (L_{D+d}). Shown by lines 17–22 of figure 2.

The procedure Prune (see figure 3) deals with the frequent K-sequences in D but not frequent in d. For sequences X, find its count in (D + d) by taking the sum of the count in Tree_D and Tree_d and check whether it is frequent in the whole database (D + d). If it is frequent, then append all supersets of X that are in Tree_D to the P_set so that the frequency of these supersets in whole database D + d can be checked here itself. Add X to L_{D+d}, then we remove the X from the P_set. Lines 3–7 of figure 3 show these steps. If the sequence X is not frequent in D + d, then remove X and its supersets from Tree_D because supersets of X cannot be frequent if there the subset is not frequent and also remove X and its superset from P_set, lines 8–10 of figure 3 shows these steps.

The procedure Prune_R (see figure 4) deals with the frequent K-sequences in d but not frequent in D. For sequences X, find its count in (D + d) by taking sum of count in Tree_D and Tree_d and check whether it is frequent in whole database (D + d). If sequence X is frequent in D + d, then add sequence X in the set of frequent sequences of D + d (L_{D+d}). Lines 3–8 of figure 4 show these steps.

The procedure Tree_Generate (see figure 5) generate tree of sequence pattern along with count by processing of database. Count shows how many times concern pattern present in different transactions of database. Lines 1–7 show these steps.

The lemmas on which this algorithm is based are as follows.Their proofs can be found [4, 22–24]. Basically these lemmas are related to the association rule but we have used in Sequence mining.

Lemma 1 Given a set of infrequent sequence X in the database D, then any superset of X will also be infrequent in D.

Assume X is an infrequent sequence in the updated database, by Lemma 1, any superset of X will also be infrequent. As compared to the previously designed algorithms, STISPM functions differently in the sense that it prunes all the supersets of frequent sets in D before they are converted to be infrequent. As stated in the previous algorithms, a K-sequence is checked only in the Kth iteration, but STISPM do not wait for the Kth iteration, it takes out the superset of infrequent sets in L_D that are infrequent in L_{D+d}.

Lemma 2 Let X be a sequence, then $X \in L_{D+d}$ only if either $X \in L_D$ or $X \in L_d$

Corollary 1 Let X is a sequence.If X is infrequent in both D and d, then X cannot be frequent in D +d.

```
Prune(P_set)
1    While P_set ≠ ϕ do
2    X = first element of P_set
3    Find sum of count of X in Tree_D and Tree_d for evaluation of support X in D+d
4    find $Supp_{D+d}(X)$
5    If $Supp_{D+d}$ (X)>=minsup then
6    add X to $L_{D+d}$
7    remove X from P_set
8    Else
9    remove X and its supersets from Tree_D
10   remove X and its supersets from P_set
11   End if
12   End while
```

Figure 3. Prune procedure.

```
Prune_R (R_set)
1    While R_set ≠ ϕ do
2    X = first element of R_set
3    Find sum of count of x in Tree_D and Tree_d for evaluation of support X in D+d
4    find $Supp_{D+d}^{(X)}$
5    If $Supp_{D+d}^{(X)}$ >= minsup then
6    add  X to $L_{D+d}$
7    add  X to $L_K^d$
8    End if
9    End while
```

Figure 4. Prune_R procedure.

```
Tree_Generate (D)
        \\ T is representing the tree space Diagram.
        \\ $R_1, R_2, R_3, \ldots, R_n$ Records of Database.
        \\ $X_1, X_2, X_3, \ldots, X_n$ No. of Items in each record.
        \\ Count is representing no. of times Itemset present in records.
1.      Starting from the first record R1.
2.      Initially take root node and no of edges according to the no of elements in first record. The value of
        element ($X_1, X_2, \ldots, X_n$) .shows the weight of edge and count =1
3.      In second step take edge from $X_2, X_3, \ldots, X_n$ in depth-first approach and generate all possible item set
        and enhance the value of count.
4.      In the third step take edge from $X_3, X_4, \ldots, X_n$ in depth-first approach and next n steps take edge Xn in
        depth-first approach and enhance count.
5.      These steps are recursively used for each of initial record and recursively generate the item set and
        enhance the value of count.
6.      The steps 1 to 5 repeat for every record in Database D
7.      Finally, we get a tree space diagram, which is the all possible itemsets of tree space diagram which are
        generated by using the forward and backward approach, i.e. backtracking.
```

Figure 5. Tree_Generate procedure.

Table 4. Sample sequence database.

Transaction Id (TID)		Itemsets	
D			
C1	(10 20)	(30 40)	50
C2	10	40	60
C3	(10 20)	40	
C4	10	(30 40)	60
d			
C5	10	(30 40)	(50 60)
C6	(10 20)	(30 40)	(50 60)
C7	10	(30 40)	50

Assume X is a candidate K-sequence in d. If $X \in L_{D+d}$ and if $X \notin L_D$, then X must belong to L_d. If $X \notin L_D$ and $X \notin L_d$ then $X \notin L_{D+d}$, by corollary 1.Otherwise, find out the support of X in D + d. Since, we have the support of X in D and d, we can quickly find out whether it is frequent or not. If (support_D (X) + support_d (x) < minsup |D + d|, then X is not frequent in D + d. By Lemma 1, all supersets of X that are infrequent are eliminated from L_D. Otherwise, added to L_{D+d}.

Lemma 3 Let X be a sequence. If $X \in L_D$ and $X \in L_d$, then $X \in L_{D+d}$.

In this case, we place X into L_{D+d} with the total support. If X is infrequent in D, we check for X in D + d by whether it is frequent or not. Also we know the support of X in D [support $_D$ (X)], it can be find out by rescan of D. We add X into L_{D+d} if support $_{D+d}$(X) > = minsup_{D+d}.

Lemma 4 STISPM generates the minimum number of candidate set.

Due to the lookahead pruning technique, the STISPM generates the minimum number of candidate sets.

6. Study with example

This section presents a working model STISPM. Table 4 contains four different transactions and each transaction has a maximum of five different items represented in D and three more transactions are added given in d.

Take the minsup = 50%

Find the large sequences in Database 'D' by traversing tree Tree_D with support ≥ minsup.

L_D^1 = {10, 20, 30, 40, 60}

L_D^2 = {(10 20), (10)(30), (10)(40), (10)(60), (20)(40), (30 40), (40)(60)}

L_D^3 = {(10 20) (40), (10) (30 40), (10) (40) (60)}

L_D^4 = {ϕ}

These are existing frequent sequences.

So $L_D = L_D^1 \cup L_D^2 \cup L_D^3$

L_D ={10, 20, 30, 40,60, (10 20), (10)(30), (10)(40), (10)(60), (20)(40), (30 40), (40)(60), (10 20)(40), (10)(30 40), (10)(40)(60)}

STISPM finds new frequent sequences in D + d.

Step by step execution of the **STISPM** algorithm is as follows

K = 1

First Iteration

Generate (T_d^1) the frequent 1-sequences in d by traversing tree Tree_d with support ≥ minsup.

T_d^1 = {10, 30, 40, 50, 60}

//set of large one sequence in d (new added transactions)

L_D^1 = {10, 20, 30, 40,60}

//set of large one sequence in D

P_set = $L_D^1 - T_d^1$

//P_set is the collection of sequence which is large in D but not large in d

P_set = {20}

Prune (P_set, K)
//pruning procedure that will prune all the supersets of a sequences from the L_D if that sequences is not large in D + d otherwise it check the largeness in D + d of the superset of that sequences present in L_D by adding them in the P_set.
For all X ∈ P_set do
X = (20)
Find support of (20) in D + d by adding count of (20) from tree Tree_D and Tree_d respectively.
Count of X = 20 in Tree_D is 2
Count of X = 20 in Tree_d is 1
Now sum of count is 3 corresponding to X = 20.
$Supp_{D+d}(20) = 3 < minsup$
So, we will remove (20) and all the supersets of (20) from L_D, Tree_D and P_set
Now L_D is
L_D = {10, 30, 40, 60, (10)(30), (10)(40), (10)(60), (30 40), (40)(60), (10)(30 40), (10)(40)(60)}
Now
P_set = {ϕ}
R_set = $T_d{}^1$ - L_D^1
R_set = {50}
//**R_set is the collection of sequence which is large in d but not large in D**
$L_d^1 = T_d{}^1$- R_set
L_d^1 = {10, 30, 40, 60}
$L_{D+d} = L_{D+d} \cup L_d^1$
L_{D+d} = {10, 30, 40, 60} //**add L_d^1 to L_{D+d}**
R_set ≠ {ϕ}
Prune_R (R_set, K)
//this procedure check the largeness of K-sequences in D + d which are frequent in d but not frequent in D by taking sum of count in tree Tree_D and Tree_d.
X = 50
Count of X = 50 in Tree_d is 3.
Count of X = 50 in Tree_D is 1.
Now sum of count is 4 corresponding to X = 50.
So $Supp_{D+d}(50) = 4 = minsup$.
Add X = (50) to L_{D+d} and L_d^1
Now R_set = {ϕ}
Now we have
L_{D+d} = {10, 30, 40, 50, 60}
L_d^1 = {10, 30, 40, 50, 60}
//Continue with the main program
K = 2 (Second Iteration)
Generate (T_d^2) the frequent 2-sequences in by traversing tree Tree_d with support ≥ minsup.
T_d^2 = {(10)(30), (10)(40), (10)(50), (10)(60), (30 40), (30)(50), (30)(60), (40)(50), (40)(60), (50 60}
L_D^2 = { (10) (30), (10)(40), (10)(60), (30 40), (40)(60)}
P_set = $L_D^2 - T_d^2$
//P_set is the collection of sequence which is large in D but not large in d
P_set = {ϕ}
R_set = T_d^2- L_D^2
R_set = {(10)(50), (30)(50), (30)(60), (40)(50), (50 60)}
//**R_set is the collection of sequence which is large in d but not large in D**
$L_d^2 = T_d^2$ - R_set
L_d^2 = {(10) (30), (10) (40), (10) (60), (30 40), (40) (60)}
$L_{D+d} = L_{D+d} \cup L_d^2$
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10) (40), (10) (60), (30 40), (40) (60)} //**add L_d^2 to L_{D+d}**
R_set ≠ {ϕ}
Prune_R (R_set, K)
//this procedure check the largeness of K-sequences in D + d which are frequent in d but not frequent in D by taking sum of count in tree Tree_D and Tree_d.
X = (10) (50)
Count of X = (10) (50) in Tree_d is 3.
Count of X = (10) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (10) (50).
So $Supp_{D+d}$ (50) = 4 = minsup.
Add X = (10) (50) to L_{D+d} and L_d^2
X = (30) (50)
Count of X = (30) (50) in Tree_d is 3.
Count of X = (30) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (30) (50).
So $Supp_{D+d}$ (30)(50) = 4 = minsup.
Add X = (30) (50) to L_{D+d} and L_d^2
X = (30) (60)
Count of X = (30) (60) in Tree_d is 2.
Count of X = (30) (60) in Tree_D is 1.
Now sum of count is 3 corresponding to X = (30) (60).
So $Supp_{D+d}$ (30) (60) = 3<= minsup.
X = (40) (50)
Count of X = (40) (50) in Tree_d is 3.
Count of X = (40) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (40)(50).
So $Supp_{D+d}$ (40) (50) = 4 = minsup.
Add X = (40) (50) to L_{D+d} and L_d^2
X = (50) (60)
Count of X = (50) (60) in Tree_d is 2.
Count of X = (50) (60) in Tree_D is 0.
Now sum of count is 2 corresponding to X = (50) (60).
So $Supp_{D+d}$ (50) (60) = 2<= minsup.
Now R_set = {ϕ}
Now we have
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)}
L_d^2 = {(10) (30), (10) (40), (10) (50), (10) (60), (30 40), (30) (50), (40) (50), (40) (60)}
//Continue with the main program
K = 3
Third Iteration
Generate (T_d^3) the frequent 3-sequences in by traversing tree Tree_d with support ≥ minsup.
T_d^3 = {(10)(30 40), (10)(30)(50), (10)(40)(50), (10)(40)(60), (30 40)(50), (30 40)(60)}
L_D^3 = {(10) (30 40), (10) (40) (60)}
P_set = $L_D^3 - T_d^3$

//P_set is the collection of sequence which is large in D but not large in d
P_set = { ϕ}
R_set = T_d^3 - L_D^3
R_set = {(10) (30) (50), (10) (40) (50), (30 40) (50), (30 40) (60)}
//R_set is the collection of sequence which is large in d but not large in D
L_d^3 = T_d^3- R_set
L_d^3 = {(10) (30 40), (10) (40)(60)}
L_{D+d} = L_{D+d} ∪ L_d^3
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)
(10) (30 40), (10) (40)(60)}}**//add L_d^3 to L_{D+d}**
R_set ≠ {ϕ}
Prune_R (R_set, K)
//This procedure check the largeness of k-sequences in D + d which are frequent in d but not frequent in D by taking sum of count in tree Tree_D and Tree_d.
X = (10) (30) (50)
Count of X = (10) (30) (50) in Tree_d is 3.
Count of X = (10) (30) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (10) (30) (50)
So $Supp_{D+d}$ (10) (30) (50) = 4 = minsup.
Add X = (10) (30) (50) to L_{D+d} and L_d^3
X = (10) (40) (50)
Count of X = (10) (40) (50) in Tree_d is 3.
Count of X = (10) (40) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (10) (40) (50)
So $Supp_{D+d}$ (10) (40) (50) = 4 = minsup.
Add X = (10) (40) (50) to L_{D+d} and L_d^3
X = (30 40) (50)
Count of X = (30 40) (50) in Tree_d is 3.
Count of X = (30 40) (50) in Tree_D is 1
Now sum of count is 4 corresponding to X = (30 40) (50)
So $Supp_{D+d}$ (30 40) (50) = 4 = minsup.
Add X = (30 40) (50) to L_{D+d} and L_d^3
X = (30 40) (60)
Count of X = (30 40) (60) in Tree_d is 2.
Count of X = (30 40) (60) in Tree_D is 1.
Now sum of count is 3 corresponding to X = (30 40) (60)
So $Supp_{D+d}$ (30 40) (60) = 3<= minsup.
Now R_set = {ϕ}
Now we have
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)
(10) (30 40), (10) (40)(60), (10)(30)(50), (10)(40)(50), (30 40)(50)}
L_d^3 = {(10) (30 40), (10) (40) (60), (10) (30) (50), (10) (40) (50), (30 40) (50)}
//Continue with the main program
k = 4
Fourth Iteration
Generate (T_d^4) the frequent 4-sequences in by traversing tree Tree_d with support ≥ minsup.
T_d^4 = {(10) (30 40) (50)}
L_D^4 = {ϕ}
P_set = L_D^4 – T_d^4
//P_set is the collection of sequence which is large in D but not large in d
P_set = {ϕ}
R_set = T_d^4 - L_D^4
R_set = {(10) (30 40) (50)}
//R_set is the collection of sequence which is large in d but not large in D
L_d^4 = T_d^4 – R_set
L_d^4 = {ϕ}
L_{D+d} = L_{D+d} ∪ L_d^4
L_{D+d} = L_{D+d} ∪{ϕ}
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)
(10) (30 40), (10) (40)(60), (10)(30)(50), (10)(40)(50), (30 40)(50)}**//add L_d^4 to L_{D+d}**
R_set ≠ {ϕ}
Prune_R (R_set, K)
//This procedure check the largeness of k-sequences in D + d which are frequent in d but not frequent in D by taking sum of count in tree Tree_D and Tree_d.
X = (10) (30 40) (50)
Count of X = (10) (30 40) (50) in Tree_d is 3.
Count of X = (10) (30 40) (50) in Tree_D is 1.
Now sum of count is 4 corresponding to X = (10) (30 40) (50)
So $Supp_{D+d}$ (10) (30 40) (50) = 4 = minsup.
Add X = (10) (30 40) (50) to L_{D+d} and L_d^4
Now R_set = {ϕ}
Now we have
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)
(10) (30 40), (10) (40) (60), (10) (30) (50), (10) (40) (50), (30 40) (50), (10) (30 40) (50)}
L_d^4 = (10) (30 40) (50)
//Continue with the main program
K = 5
Fifth Iteration
T_d^5 = Generate frequent 5-sequence from Tree_d
T_d^5 = {ϕ}
Now set of frequent Sequences in D + d
L_{D+d} = {10, 30, 40, 50, 60, (10) (30), (10)(40), (10) 50), (10)(60), (30 40), (30)(50), (40)(50), (40)(60)
(10) (30 40), (10) (40) (60), (10) (30) (50), (10) (40) (50), (30 40) (50), (10) (30 40) (50)}
After applying steps 17–22 of figure 1 we get Size of Longest Sequences is (n =) 4.
4-Sequences = {(10) (30 40) (50)}
Set of All SubSequences of {(10) (30 40) (50)} is
S_{D+d}^m = {10,30,40,50, (10)(30), (10)(40), (10)(50), (30)(50), (40)(50), (30 40), (10)(3040), (10)(30)(50), (10)(40)(50), (30 40)(50)}
L_{D+d}^m = L $_{D+d}$ - S_{D+d}^m
$L_{D+d}{}^m$ = {60, (10)(60), (40)(60), (10)(40)(60), (10) (30 40) (50)}

//End of main Prog.

L^{m}_{d+D} = {60, (10)(60), (40)(60), (10)(40)(60), (10)(30 40)(50)}

//The output is in $L_{D+d}{}^{m}$ (the maximum frequent Sequences on whole dataset)

7. Performance analysis of STISPM algorithm

This section is devoted for the discussion of the performance of the developed incremental algorithm STISPM for mining of sequential pattern. Algorithm STISPM implemented in JAVA and test runs are performed on HP compatible PC with a CORE i3 Processor of 1. 80 GHz and 4 GB of main memory running the Microsoft Windows 8.1 operating system. As for the test data, we have used the synthetic datasets of the form Cx. Ty. Lz. Dm. dn. This format means

Cx is the average no. of transaction per customer in the Dataset
Ty is the average no. of items per transaction
Lz is the average length of Maximal large sequence
|D| = m is the number of transaction in old dataset
|d| = n is the number of transaction in newly added dataset
We have used the synthetic data of the format C4. T3. L5. D50 k. dn where n varies from 10,000 to 50,000.

Figure 6 shows the execution time for $n = 30{,}000$, 40,000 and 50,000. The minimum support value is increased from 30% to 50%. To test the scalability with the number of transactions of d, the incremented database size n is set 10,000, 20,000, 30,000, 40,000 and 50,000. Figure 7 shows the results. Consider the two support values 30% and 40%. The execution time increases with the growth of d size. Thus, STISPM shows good scalability.

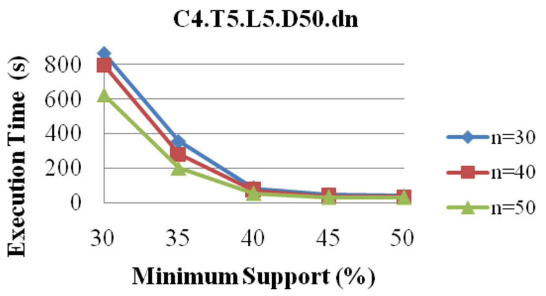

Figure 6. Execution time of STISMP for $n = 30{,}000$, 40,000 and 50,000.

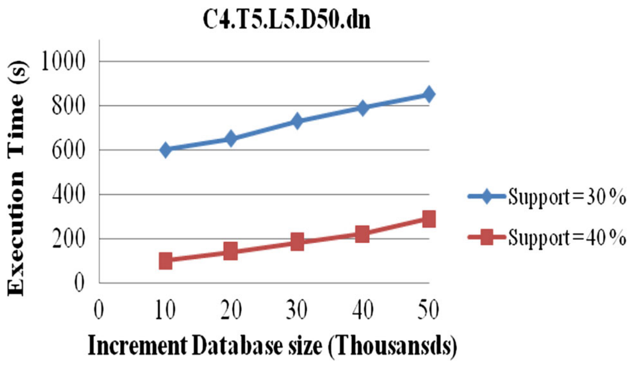

Figure 7. Scalability with the incremented database d.

8. Conclusions

In this research paper, we propose an algorithm STISPM for an incremental mining of sequential patterns using depth-first approach with backward tracking and dynamic look ahead pruning strategy. The structural characteristics of the sequence tree incorporate and enable STISPM to derive the full advantage of the results of the preceding mining. The unique feature and the advantages of STISPM is that it does not scan the existing database and also does not generate the candidate's sequences so as to find out and evaluate the new set of frequent sequences. Moreover, STISPM prunes the sequences that are not frequent as quickly as possible by a dynamic lookahead pruning strategy. Every time we execute and implement STISPM to find and get those sequences which are frequent in subsequent transaction but is not frequent in the whole database except the added one. Now the mechanism and technique systematically deletes all such supersets that are frequent in the added transaction from the existing frequent sequences and also from the sequence tree. In this manner, many of the sequences that are not frequent in the updated database is pruned precisely in the beginning itself, and this early stage pruning reduces the size of sequential tree drastically. This new innovative technique is very expedient and efficient in comparison to mining database technique which is long and tedious and performing every operation since the beginning.

References

[1] Agarwal R and Srikant R 1995 Mining sequential pattern. In: *Proceedings of the 11th International Conference on Data Engineering* pp. 3–14
[2] Zhao Q and Bhowmick S S 2003 Sequential pattern mining: A survey. *Technical Report,* CAIS, Nanyang Technological University Singapore. No. 118
[3] Han J, Pei J, Mortazavi Asl B, Chen Q, Dayal U and Hsu M C 2000a FreeSpan: Frequent pattern-projected sequential pattern mining. In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* pp. 355–359
[4] Cheung D W, Lee D W and Kao S D 1997 A general incremental technique for maintaining discovered association

rules. In: *Proceedings of the 5th International Conference on Database System for Advanced Applications*, pp. 185–194

[5] Lin M Y and Lee S Y 1998 Incremental update on sequential patterns in large databases. In: The 10th IEEE international conference on tools with artificial intelligence, pp. 24–31

[6] Sarda N L and Srinivas N V 1998 An adaptive algorithm for incremental mining of association rules. In: *The 9th International Workshop on Database and Expert Systems*, pp. 240–245

[7] Zhang S 1999 Aggregation and maintenance for database mining. *Intell. Data Anal.* 475–490

[8] Srikant R and Agrawal R 1996 Mining sequential patterns: Generalizations and performance improvements. In: *Proceedings of the 5th International Conference on Extending Database Technology,* pp. 3–17

[9] Garofalakis M N, Rastogi R and Shim K 1999 SPIRIT: Sequential pattern mining regular expression constraints. In: *Proceedings of the 25th international conference on very large data base*, pp. 223–234

[10] Parthasarathy S, Zaki M, Ogihara M and Dwarkadas S 1999 Incremental and interactive sequence mining. In: *Proceedings of the 8th International Conference on Information and Knowledge Management,* pp. 251–258

[11] Ayan N F, Tansel A U and Arkun E 1999 An Efficient algorithm to update large itemsets with early pruning. In: *Proceedings of the 5th ACM SIGKDD International Conferences on Knowledge Discovery and Data Mining*, pp. 287–291

[12] Omiecinski E and Savasere A 1998 Efficient mining of association rules in large dynamic databases. In: *Proceedings of 16th British national conference on databases,* pp. 49–63

[13] Cheng H, Yan X and Han J 2004 IncSpan: Incremental mining of sequential patterns in large database. In: *Proceedings of the 10th ACM SIGKDD International Conference Knowledge Discovery and Data Mining,* pp. 527–532

[14] Huang J W, Tseng C Y, Ou J C and Chen M S 2008 A general model for sequential pattern mining with a progressive database. *IEEE Trans. Knowl. Data Eng.* 20(9): 1153–1167

[15] Hong T P, Wang C Y and Tseng S S 2011 An incremental mining algorithm for maintaining sequential patterns using pre-large sequences. *Expert Syst. Appl.* 38: 7051–7058

[16] Pei J, Han J, Mortazavi Asl B, Pinto H, Chen Q, Dayal U and Hsu M C 2001 Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. In: *International Conference on Knowledge Discovery in Databases and Data Mining*, pp. 215–224

[17] Ayres J, Flannick J, Gehrke J and Yiu T 2002 Sequential pattern mining using a bitmap representation. In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 429–435

[18] Zaki M J 2001 SPADE: An efficient algorithm for mining frequent sequences. In: *Proc. Mach. Learn.* (special issue on unsupervised learning), 42: 31–60

[19] Han J, Pei J and Yin Y 2000b Mining frequent patterns without candidate generation. In: *The Proceedings of 2000 ACM SIGMOD International Conference on Management of Data* pp. 1–12

[20] Lin C W, Hong T P and Lu W H 2009 The Pre-FUFP algorithm for incremental mining. *J. Expert Syst. Appl.* 36: 9498–9505

[21] Hong T P, Lin J W and Wu Y L 2006 A fast updated frequent pattern tree. In: *The IEEE International Conference on System, Man, and Cybernetics*, pp. 2167–2172

[22] Cheung D W, Han J, Ng V T and Wong C Y 1996 Maintenance of discovered association rules in large databases: An incremental updating approach. In: *12th IEEE International Conference on Data Engineering,* pp. 106–114

[23] Agrawal R and Srikant R 1994 Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Databases*, pp. 487–499

[24] Thomas S, Bodagala S, Alsabti K and Ranka S 1997 An efficient algorithm for the incremental updating of association rules in large databases. In: *Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining*, pp. 263–266