CSSF Miner: A Technique for Mining of Constraint Sequential Patterns from Progressive Database

V.Candra Shekhar Rao  
Associate Professor, Dept of Computer Science,  
KITS, Warangal, Research Scholar JNTU Hyderabad  
JNTU, Nachupally, Karimnagar Dist.

Dr. P.Sammuulal  
Assistant Professor, Dept of CSE,  
JNTU, Nachupally, Karimnagar Dist.

Abstract—Sequential pattern mining is a significant data-mining method for determining time-related behavior in sequence databases. The information achieved from sequential pattern mining can be used in marketing, medical records, sales analysis, and so on. Existing methods only focus on the concept of frequency because of the assumption that sequences’ behaviors do not change over time. Several efficient algorithms for maintaining sequential patterns have been developed. Here, we have developed proposed Algorithm for Mining of constraint sequential patterns from progressive database. In order to efficiently capturing the dynamic nature of data addition and deletion into the mining problem, initially, we construct the updated CSSF-trie from the static database. Then, the database gets updated from the distributed sources; data in the database may be static, inserted, or deleted. Whenever the database is updated from the multiple sources, CSSF trie is also updated by including the updated sequence. Then, the updated CSSF-trie is used to mine the progressive CSSF-patterns using the proposed trie pattern mining algorithm. Finally, the experimentation is carried out using the synthetic and real life distributed databases that are given to the progressive CSSF-miner using thread environment. The experimental results and analysis provides better results in terms of the generated number of sequential patterns, execution time and the memory usage over the existing IncSpan algorithm.

Index Terms—Trie, Sequential pattern mining, CSSF, Progressive database, updated CSSF-trie.

I. INTRODUCTION

Sequential pattern mining: Sequential pattern mining is an important subject of data mining, a further promotion of association rule mining, and it is also widely applied [1]. Sequential pattern mining algorithms address the problem of discovering the existent frequent sequences in a given database [2]. Sequential pattern mining is closely related to association rule mining, except that the events are linked by time [3]. Sequential patterns indicate the correlation between transactions while association rule represents intra transaction relationships. In association rule mining, the mining patterns are about which items are brought together frequently, those items must come from the same transaction. While the results of sequential pattern mining are about which items are bought in a certain order by the same customer, with those items coming from different transactions [4]. Sequential patterns can help managers determine which items are bought one after another in a sequence [5], or to analyze browsing orders of homepages in a Web site [6] and more.

Sequential pattern mining is commonly defined as finding the complete set of frequent subsequences in a set of sequences [7]. Sequential pattern is a sequence of itemsets that frequently occur in a specific order; all items in the same itemset are supposed to have the same transaction time value or within a time gap. Each sequence corresponds to a temporarily ordered list of events, where each event is a collection of items (itemset) occurring simultaneously. The temporal ordering among the events is induced by the absolute timestamps associated with the events [8]. Usually, all the transactions of a customer are together viewed as a sequence, called customer-sequence, where each transaction is represented as an itemset in that sequence and all the transactions are listed in a certain order with regard to the transaction-time [4]. The process of mining sequential patterns from a customer transaction database is described as follows:

Let \( D \) be a set of customer transactions where each transaction \( T \) consists of a customer-id, a 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 non-empty set of items. A sequence \( S \) is a set of itemsets ordered according to their time stamp. It is denoted by \( \langle s_1, s_2, \ldots, s_n \rangle \), where \( s_j, j \in 1, \ldots, n \) is an itemset. A \( k \)-sequence is a sequence of \( k \) items (or of length \( k \)). A sequence \( \langle s_1, s_2, \ldots, s_n \rangle \) is a sub-sequence of another sequence \( \langle s_1', s_2', \ldots, s_m' \rangle \) if there exist integers \( i_1 < i_2 < \ldots < i_j < \ldots < i_n \) such that \( s_1 \subseteq s_1', s_2 \subseteq s_2', \ldots, s_n \subseteq s_n' \) [9]. The problem of mining sequential patterns is to find all sequences \( s \) such that \( \text{supp}(s) \geq \text{msup} \) for a database \( D \), given a support threshold \( \text{msup} \).

The task of discovering all frequent sequences in large databases is quite challenging, since the search space is extremely large. For instance, with \( m \) attributes there are \( O(m^k) \) potentially frequent sequences of length \( k \) [10]. The factors, which make sequential pattern mining a very difficult and time-consuming one, are as follows; first, the formation of a pattern is not limited to single items but itemsets. Second, neither the number of itemsets in a pattern nor the number of items in an itemset is known a priori. Third, patterns could be formed by any permutation, of any combination of possible items in the database [11]. Since its introduction in 1995 by Agrawal et al. [7], the task of mining sequential patterns has received a great deal of attention. Following their work, there have been many studies on sequential pattern mining and its applications [14, 15]. Although efficiency of mining the complete set of sequential patterns has been improved substantially, in many
cases, sequential pattern mining still faces tough challenges in both effectiveness and efficiency. On the one hand, there could be a large number of sequential patterns in a large database. A user is often interested in only a small subset of such patterns. Presenting the complete set of sequential patterns may make the mining result hard to understand and hard to use [13].

**Constraint-based sequential pattern mining:** On the other hand, although efficient algorithms have been proposed, mining a large amount of sequential patterns from large data sequence databases is inherently a computationally expensive task. If we can focus on only those sequential patterns interesting to users, we may be able to save a lot of computation cost by those uninteresting patterns [13]. In recent years, constraint-based sequential pattern mining algorithms have received a great deal of attention among researchers. A constraint C for sequential pattern mining is a boolean function C(α) on the set of all sequences. The problem of constraint-based sequential pattern mining is to find the complete set of sequential patterns satisfying a given constraint C [13]. Constraints can be examined and characterized from different points of views. The category of constraints includes the following: time constraints, item constraints, length constraints, super-pattern constraints, regular expression constraints, user defined constraints and more. In the context of constraint-based sequential pattern mining, Srikant and Agrawal have generalized the scope of sequential pattern mining to include time constraints, sliding time window, and user-defined taxonomy, and also have presented an a priori-based, improved algorithm GSP (i.e., Generalized Sequential Patterns) [14]. Following this, many effective algorithms have been presented with the incorporation of different constraints [19-24].

**Incremental Mining of Sequential Patterns:** Traditional methods for data mining typically make the assumption that the data is centralized, memory-resident, and static. This assumption is no longer tenable. Such methods waste computational and input/output (I/O) resources when data is dynamic, and they impose excessive communication overhead when data is distributed. Efficient implementation of incremental data mining methods is, thus, becoming crucial for ensuring system scalability and facilitating knowledge discovery when data is dynamic and distributed [12]. Incremental algorithms essentially reuse previously mined information and try to combine this information with the fresh data to efficiently compute the new set of frequent itemsets. In reality, the contents of databases are updated incrementally in many application domains. For example, customer shopping transaction database is growing daily due to the appending of newly purchased items for existing customers for their subsequent purchases and/or insertion of new shopping sequences for new customers [16]. Therefore, to get all sequential patterns, the mining algorithm has to be run whenever the database changes, because that some sequences which were not frequent in old database may become frequent in updated database.

Obviously, the discovery of sequential patterns from scratch every time is ineffective. This leads to the study of the incremental mining algorithm of sequential patterns.

When new sequences are added into old databases, the incremental mining algorithm minimizes the computational and I/O costs by re-using the information from the previous mining results from old database [17]. Incremental mining of sequential patterns is the process of generating new patterns in the updated database (old + new data) by using only the updated part (new data) and previously generated old patterns. However, it is nontrivial to mine sequential patterns incrementally, especially when the existing sequences grow incrementally because such growth may lead to the generation of many new patterns due to the interactions of the growing subsequences with the original ones [16]. Currently, several algorithms have been proposed for incremental mining of sequential patterns [9], [11], [13], [18], [9], [16], [17]. Of these, the IncSpan algorithm proposed by Cheng et al. [25] has gained immense importance in incremental mining of sequential patterns.

The organization of the paper is as follows: The review of related research is given in section II. The problem statement and the proposed algorithm for mining of CSSF-sequential patterns is given in section III. The experimental results and its discussion are presented in section 4 and the conclusions are summed up in section IV.

**II. LITERATURE REVIEW**

The literature presents with a huge number of approaches for constraint-based sequential pattern mining and incremental mining of sequential patterns. In recent times, developing approaches for incremental mining of constraint-based sequential patterns has gained immense importance in real life applications. A concise review of some recent researches related to the incremental mining of sequential patterns is presented here.

Jiaxin Liu [27] have proposed a data storage structure, called frequent sequence tree, and give the construction algorithm of frequent sequence tree, called Con_FST. The root node of the frequent sequence tree stored the frequent sequence tree support threshold and the path from the root node to any leaf node represents a sequential pattern in the database. Frequent sequence tree stored all the sequential patterns with its support that meet the frequent sequence tree support threshold, so when the support was changed, the algorithm which uses frequent sequence tree as the storage structure could find all the sequential patterns without mining the database. Philippe Fournier et al. [32] have presented a Rule Growth, an algorithm for mining sequential rules common to several sequences. Unlike other algorithms, Rule Growth used a pattern-growth approach for discovering sequential rules such that it can be much more efficient and scalable. They have presented a comparison of Rule Growth’s performance with current algorithms for three public datasets. The result showed that Rule Growth clearly outperforms current algorithms for all three datasets under low support and confidence threshold.

K.M.V. Madan Kumar et al. [31] have proposed algorithm used the concept of “percentage of participation” instead of occurrence frequency for every possible combination of items or item sets. The concept of percentage of participation was calculated based on the minimum support
threshold for each item set. The used algorithm by name “MS-SPADE”, which stands for Multiple Support Sequential Pattern Discovery using Equivalent classes, which discovers sequential patterns by considering different multiple minimum support threshold values for every possible combinations of item or item sets. Chuan cong Gao [28] have proposed an algorithm which stores only frequent closed prefixes in its enumeration tree structure, used for mining and maintaining patterns in the current sliding window, to solved the frequent closest sequential pattern mining problem efficiently over stream data. Some effective search space pruning and pattern closure checking strategies have been also devised to accelerate the algorithm. The result showed that algorithm outperformed other state-of-the-art algorithm significantly in both running time and memory used.

Jiaxin Liu [29] have proposed that the structure of sequence tree based on projected database, called sequence tree, and give the Steeps algorithm which was used to construct the sequence tree. Sequence tree was a data storage structure; it has been the similar in structure to the prefix tree. But, the sequence tree stores all the sequences in the original database. The path from the root node to any leaf node represents a sequence in the database. The structural characteristic of sequence tree makes it suitable for incremental sequential pattern mining. The result showed that the incremental mining algorithm of sequential patterns which uses the sequence tree as the storage structure for sequences outperformed Prefix Span in space cost on condition that the support threshold was smaller. To capture the dynamic nature of data addition and deletion, Jen-Wei Huang et al. [26] have proposed a general model of sequential pattern mining with a progressive database while the data in the database may be static, inserted or deleted. In addition, they presented a progressive algorithm Pisa, standing for Progressive mining of Sequential pAtterns, to progressively discover sequential patterns in defined time period of interest. The period of interest is a sliding window continuously advancing as the time goes by. 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. The height of the sequential pattern tree proposed was bounded by the length of period of interest, thereby effectively limiting the memory space required by Pisa that is significantly smaller than the memory needed by alternative methods.

B. N. Keshavamurthy et al. [33] have proposed an algorithm for mining of frequent pattern in progressive databases. In real world applications such as market basket analysis of retail-shop where the items were associated static attribute weight, which reflects each item has different importance and dynamic attribute support, which represents the frequency of an item. The mining of items which has having both static and dynamic attributes reveals an important knowledge than the traditional patterns. They have used two notions in the process of mapping input items to general tree structure. One, the product of dynamic attribute value support and static attribute weight should be greater than user defined threshold. Second, the dynamic attribute value support should be greater than user defined threshold.

Tzung-Pei et al. [30] have proposed an incremental mining algorithm for maintaining sequential patterns based on the concept of pre-large sequences to reduce the need for rescanning original databases. Pre-large sequences were defined by a lower support threshold and an upper support threshold that act as gaps to avoid the movements of sequences directly from large to small and vice-versa. The 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. Thus, as databases grow larger, the numbers of new transactions allowed before database rescanning was required also grow.

III. PROPOSED METHODOLOGY FOR MINING OF CONSTRAINT SEQUENTIAL PATTERNS FROM PROGRESSIVE DATABASE

A. Problem description

The problem of mining of constraint sequential is discussed and some basic definitions are described in this subsection. The problem of mining sequential patterns was first introduced in [36] and extended in [37]. This section describes concise description of sequential pattern mining and constrained sequential pattern mining.

Sequential Pattern Mining:

The sequential pattern mining problem is to mine the complete set of sequential patterns with respect to a given sequential database $D$ and a support threshold min_sup. Let $D$ be a sequential database where each transaction $T$ contains customer-id, a transaction time and a set of items entailed in the transaction. Let $I = \{i_1, i_2, ..., i_m\}$ be a set of items. An itemset is a non-empty subset of items, and an itemset with $p$ items is called a $p^{th}$ itemset. A sequence $S$ is an ordered list of itemsets based on their timestamp. It is represented by $\{q_1, q_2, ..., q_n\}, where q_j, J \in 1, 2, ..., n$ is an itemset. A sequence of $p$ items (or of length $p$) is called $p$-sequence. A sequence $\{q_1, q_2, ..., q_n\}$ is a sub-sequence of another sequence $\{q_1', q_2', ..., q_m'\}, (n \leq l)$, if three exist integers $a_1 < a_2 < ... a_j < a_m$ such as $q_1 \subseteq q_{i_1}, q_2 \subseteq q_{i_2}, ..., q_n \subseteq q_{i_n}$. The mining of sequential patterns is to discover all sequences $S$ such that $Sup(S) \geq min_sup$ for a database $D$.

Constrained Sequential Pattern Mining:

The problem of mining constraint based sequential patterns is to discover the complete set of sequential patterns satisfying a specified constraint $C$. The literature [38] presents several constraints that are used in the sequential pattern mining process. Based on this problem statement, we define the important terms used in the proposed approach to mine the progressive CSF patterns. Furthermore, we define some other definitions utilized in the proposed approach.
\textbf{Definition 1 (CSSF trie)}: For a sequence database $D$, we can construct a CSSF-trie after mining the CSSF patterns from it. Here, every node $n$ in the CSSF-trie contains items and its relevant information, represented as $n = \{(p(t_1, t_n)), (C, S, F)\}$, where $p$ is the item, $t_1$ is the starting time interval, $t_n$ is the ending time interval, $C$ is compactness, $S$ is the seasonal monetary and $F$ is frequency. Here, the depth of the CSSF-trie $(d)$ is equivalent to the larger length of the CSSF-sequential patterns.

\textbf{Definition 2 (Empty node)}: A node in the CSSF-trie is called as empty node only if (i) $t_1$ and $t_n$ is filled with zero, (ii) $p$ should contain the item information and (iii) $C, S$ and $F$ have the zero value. This node is necessary for building the CSSF-trie after mining the sequences from the static database because the CSSF-miner does not satisfy the downward closure property. So, some of the sequential patterns are frequent but, their subsets may not be frequent. These types of subsets are stored in the empty nodes, but their supersets are stored in the precious CSSF-node that is frequent.

\textbf{Definition 3 (updated CSSF trie)}: After inserting some nodes in CSSF-trie on behalf of updated database, then it is called as, updated CSSF-trie, in which some nodal information may be updated or some new nodes may be included.

B. Proposed Algorithm for Mining of constraint sequential patterns from progressive database

To solve the progressive sequential pattern mining algorithm, we propose a progressive mining algorithm CSSF. In our proposed approach, we have defined the terms such as, sequence, compactness sequence, seasonal sequence, and seasonal compact sequence.

\textbf{Definition 4 (Compactness sequence)}: Let $S = (q_1, t_1, M_1), (q_2, t_2, M_2), \ldots, (q_m, t_m, M_m)$ be a sequence of itemsets, where $t_1 < t_2 < \ldots < t_m$ and $C_T$ be the predefined threshold. $S$ is the compact sequence, if compactness constraint is satisfied, i.e. $t_m - t_1 = C_T$.

\textbf{Definition 5 (Sequential sequence)}: Let $D$ be the sequential database containing itemsets, the sequential sequence should occur within "$T1$ or $T4$ or $T5$".

\textbf{Definition 6 (Seasonal Compactness sequence)}: Let $D$ be the sequential database containing itemsets, $C_T$ be the predefined compactness threshold. $S$ is said to be a compact frequent sequence of $D$ if and only (a) It should be sequence of $T1$ or $T4$ or $T5$, (b) compact should be above the threshold.

Generally, the change on a sequential database can be categorized as 1) deleting records, 2) inserting new records, and 3) appending new items on the existing records. By managing these issues, the proposed algorithm was designed with the aid of five important steps. The procedure used for mining high items involves the following two important steps.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>ab</td>
<td></td>
<td>f</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>U2</td>
<td>a</td>
<td>e</td>
<td></td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>U3</td>
<td>d</td>
<td>f</td>
<td>d</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>U4</td>
<td></td>
<td>g</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>U5</td>
<td>e</td>
<td></td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U6</td>
<td></td>
<td></td>
<td>m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) Construction of CSF-trie

Firstly, we have used three concepts namely, compactness, seasonal sequence and seasonal compact sequence and these three concepts detailed in definition 4.5.6. To begin with, the proposed algorithm discovered the 1-length seasonal compact frequent patterns (1-CSSF) by considering the compactness threshold and support threshold. Then, we filtered the 1-length compact seasonal frequent sequential patterns (1-CSSF) from the mined 1-CSF patterns by inputting the seasonal constraint. Subsequently, we develop the projected database corresponding to the mined 1-CSF patterns and 2-CSF patterns that are mined from the projected database. Again, we found the 2-CSF sequential patterns from it and the process was applied recursively until all length CSSF sequential patterns were mined.

Example: The example database is given in Table 1 in which the timestamps T1 to T4 are static set of data, whereas the timestamps T5 to T6 are the updated set of data. The corresponding seasonal values of all the items are given in Table 2 and the mined CSSF-sequential patterns using for the input thresholds, \(\min_\text{sup} \geq 2\), \(C_T \leq 2\), \(T_m = 6\).

Table 1 provides an example of a transaction database and Table 2 gives the seasonal time transaction data.

(2) Structuring the CSSF-trie from the CSSF patterns

Step 1: Construction of CSSF-trie
Step 2: Building up the CSSF-trie from the CSSF patterns
Step 3: Managing the updated CSSF-trie
Step 4: Managing the node deletion operation in the updated CSSF-trie
Step 5: Mining of constraint sequential patterns from the CSSF-trie
We have formed the CSSF-trie from the assembled CSSF sequential pattern after mining the CSF-sequential pattern. The process of building up the CSSF-trie is described as follows. The compactness value, seasonal sequence value and the frequency value of each of the patterns should be maintained properly. The CSSF trie that contains all the sequential patterns are building up, by which mines the progressive CSSF-patterns without candidate generation, requires the less database scans to achieve a highly compactness, frequency and the seasonal trie structure. According to the frequency and compactness list, it produces a CSSF-pattern trie, which can store compact information on transactions involving sequential patterns. At first, transactions are inserted into the CSSF-trie according to a predefined order one by one. The order of all the patterns of a CSSF-trie is maintained by a list, which maintains the current frequency value, seasonal sequence and the compactness value with the timestamp of each item.

**Example:** The first insertion phase begins with the root node by taking all the mined patterns. By taking the first node ‘ab’, the the obtained sequential patterns from the CSFF-mining algorithm are <a>, <d> in first timestamp. Initially, the empty node ‘ab’ is appended with the root node of the trie by giving the corresponding compactness value, seasonal value and the frequency value.

Initially, we have taken the first node <ab>, and then first timestamp nodes adding continuously with this first node. At this time, whether we check previous two nodes based on the compact value, which is add or create new node. This process is shown in figure 1. When every node adding process, the compactness, seasonal and frequency have updated. Likewise, all the remaining patterns are utilized to build the CSSF-pattern trie. The final CSSF-trie for the static database is shown in figure 1.

![CSSF-trie of the example database](image)

**(3) Managing the update operation**

After building up the CSSF-trie from the example database, we have to build the trie structure of the updated sequences. After inserting some of the transactions, if the items order of the list deviates from the current compactness, frequency and seasonal sequence to a specified degree, the CSSF-trie is dynamically restructured by the current compactness, frequency and seasonal sequence and the list updates the pattern order with the current list. The sequential patterns obtained from the updated sequences are incremented based on the timestamps, compactness, seasonal value and the frequency of each patterns. While updating the trie structure, CSSF-trie constantly maintains the initial sort order of the sequential patterns with their information. Thus, it adds the new frequent items at the end of a list and it constructs to maintain the frequency of each item and in the trie structure as new nodes. The information about the compactness, frequency and the seasonal value should be updated in a timely manner. The timestamp of the sequence in the child node should be updated as the new one. This is reasonable because for every element between the old timestamp and the new one, they are already appended to this node as a candidate sequential pattern with the old timestamp. Thus, the sequential patterns between the old timestamp and the new one can be found. Additionally, for the elements after the new timestamp, appending them to the node having the
sequence with the new timestamp is the only way to find up-to-date sequential patterns beginning at the new timestamp.

**Example:** By considering the updating nodes of the CSSF-trie, the newly inserted items are arrived in a periodic manner. Here, in timestamp T5, the items ‘<f>’, ‘<h>’, ‘<g>’, and ‘<gs>’ are the new set of items. The updated CSSF-trie with timestamp T5 and T6 is given in Figure 2. In the CSSF-trie, the newly updated node to the root node is marked as in dotted line, whereas the update process is done in the existing nodes is indicated as a thin line and the dark line represents the nodes in which there is no update is carried out.

![Fig. 2. Updated CSSF-trie](image)

(4) Managing the node deletion operation in the updated CSSF-trie

When every three time stamp, the three conditions i.e. definition 3, 4, 5 have checked in updating CSSF-trie. If any nodes are not satisfy for definition 3, 4, 5, then nodes are deleted. When mining of progressive CSSF sequential patterns, the newly arrived patterns may not be identified as frequent one if the static database is a larger one. It is noted that users are usually more interested in the recent data than the old ones. So, the deletion of an item from the CSSF-trie is carried out utilizing the time information stored in every node. Thus, the incompact nodes and the non-zero infrequent nodes should be deleted from the final updated CSSF-trie.

**Example:** While deleting the obsolete sequences, the timestamp of the nodes which couldn’t satisfy the time sequences as the updating process is carried out. When every three time stamp, the three conditions i.e. definition 3, 4, 5 have checked in updating CSSF-trie and we have deleted the incompact nodes, which don’t satisfy the user specified threshold, where there is no update process are carried out. As well, we have removed the non-zero infrequent nodes in which the frequent value is less than the threshold. The CSSF-trie with no incompact nodes is shown in Figure 2. The final CSSF-trie without non-zero infrequent nodes is shown in Figure 3.

![Fig. 3. Final updated CSSF-Trie](image)

(5) Mining of progressive CSSF patterns from the progressive database using proposed trie pattern mining algorithm

The progressive CSSF patterns are mined from CSSF trie based on the user specified thresholds after the construction of updated CSSF-trie. Here, we have designed trie pattern mining algorithm that uses the top-down process to mine the CSSF-patterns. We start with the mining process from the top nodes of the CSSF-trie and their corresponding paths are extracted from it. Then, by combining the nodes of each level, the progressive CSSF patterns are obtained. The pseudo code for the proposed procedure for mining the progressive CSSF-patterns is given as follows.

**Pseudo code:**

**Input:** CSSF-trie, min_sup, compactness, Timestamp \( T_i \)

**Output:** A complete set of Progressive CSSF patterns

**Assumptions:**

- \( n \rightarrow \) Number of nodes (next to the root node) in the constructed CSSF-trie
- \( \text{min\_sup} \rightarrow \) Minimum support Threshold
- \( \text{PCSSF\_pat} \rightarrow \) Progressive CSSF-patterns
- \( k \rightarrow \) Number of distinct paths
- \( D \rightarrow \) Depth of the path
- \( p_l \rightarrow \) Item information in the node

**Pseudo code:**

```pseudo
cbegin
  For each node m in CSSF trie
  For (i=1 to i< timestamp size)
    If i equal to 0
      Each node in timestamp is added as root node
    Else
      Check the compactness value
      If condition satisfied then the node value added as corresponding trie
    Else
      Add new node on trie
      End if
    End if
  End for
End
```

**Sub routine:** du_miner \((d,l))\)

```pseudo
cbegin
  For all nodes \((d,l))\)
  S\_pat<<p_l
  for (i=1;i<i<D ; i++)
    S\_pat<<p(i+1)
  End for
end```

**Figure 4:** Pseudo code of proposed algorithm for Mining of constraint sequential patterns from progressive database

**Example:** From the final updated CSSF-trie shown in figure 3, one of the top node `<ab>` and its corresponding paths are extracted. From the paths, we have combined each level of nodes so that the progressive CSSF patterns, `<c>`, `<f>`, `<h>` are obtained. The mined sequential CSSF-patterns for all the top nodes are given in the below table 3.

**Table 3. Final Progressive CSSF-Patterns**

<table>
<thead>
<tr>
<th>CSFF-patterns</th>
<th><code>&lt;ab&gt;</code></th>
<th><code>&lt;ab&gt;,&lt;c&gt;,&lt;f&gt;</code></th>
<th><code>&lt;a&gt;</code></th>
<th><code>&lt;a&gt;,&lt;e&gt;</code></th>
</tr>
</thead>
</table>
IV. RESULTS AND DISCUSSION

The experimental results of the proposed algorithm for mining of progressive CSSF-sequential patterns from a progressive database are described in this section. The experimental results and analysis of the CSSF-sequential patterns are done with the aid of the well-known incremental IncSpan Algorithm [34].

A. Experimental Design

The proposed algorithm for mining of CSSF-sequential patterns is programmed using Java (jdk 1.6). The experimentation has been carried out on a 2.9 GHz, dual core PC machine with 1 GB main memory running a 32-bit version of Windows 7. The proposed algorithm execute in a distributed environment that means the updation of data records can be done from the multiple sources. So, we run the algorithm in thread environment, in which the updation of data records is done in various threads. The performance of the proposed algorithm has been evaluated using the synthetic datasets as well as real life datasets. Synthetic dataset: We have generated a set of synthetic data sequence by a data generator similar in spirit to the IBM data generator designed for testing sequential pattern mining algorithms.

Real life datasets: We make use of “MSNBC.com Anonymous Web Data” [35] that was taken form UCI machine learning repository. This data describes the page visits of users who visited msnbc.com on September 28, 1999. Visits are recorded at the level of URL category (“frontpage”, “news”, “tech”, “local”, ”opinion”, “on-air”, “misc”, “weather”, “health”, “living”, “business”, “sports”, “summary”, “bbs” (bulletin board service), ”travel”, ”msn-news”, and ”msn-sports”) and are recorded in time order.

B. Performance Evaluation

The performance of the proposed CSSF-sequential pattern mining algorithm from the progressive database is evaluated by three standard evaluation measures. They are:

- **Number of sequential patterns**, i.e., the significant number of sequential patterns generated based upon the given minimum support threshold,
- **Execution time**, i.e., the time taken to execute the computer program and it characteristically depends with the input size
- **Memory usage**, i.e., the memory utilized by the current jobs present in the particular system. We have analyzed our proposed algorithm with the well known incremental algorithm, IncSpan by the synthetic and the real life datasets.

1) Analysis of the progressive CSSF- miner with Synthetic dataset

With the help of the Synthetic dataset, we have analyzed the mined CSSF-sequential patterns with the IncSpan algorithm by three ways of evaluation measures with different support values. We have done the analysis and plotted as a graph by computing the generated number of sequences, execution time and the memory usage with different minimum support threshold. By analyzing the plotted graphs of figure 5, 6 and 7 using the synthetic datasets, we have found that the proposed progressive CSSF-Miner algorithm efficiently mined (number of sequences=523 for 40 support values) the sequential patterns than the incremental IncSpan algorithm. Here, the input sequences have been varied in certain time intervals. The generated number of sequences shows better results in our proposed approach is given in Fig. 5. But in figure 6, the corresponding execution time of the CSSF-Miner gets slightly slipped down (support values=50 and 60) in some cases than the IncSpan algorithm. The effective usage of the memory in the proposed algorithm is shown in figure 7.

2) Analysis of the progressive CSSF- miner with Real life dataset

![](image1)

![Fig. 5. Generated sequential patterns of synthetic dataset with different support values](image2)

![Fig. 6. Execution time of synthetic dataset with different support values](image3)

![Fig. 7. Memory usage of synthetic dataset with different support values](image4)
With the aid of the Real life dataset, we have analyzed the mined CSSF-sequential patterns with the IncSpan algorithm by three ways of evaluation measures with diverse support values. We have done the analysis and plotted as a graph by computing the generated number of sequences, execution time and the memory usage with different minimum support threshold. By analyzing the plotted graphs of figure 8, 9 and 10 using the Real life datasets, we have find that the proposed progressive CSSF-Miner algorithm efficiently mined (number of sequence=99 for 40 support values) the sequential patterns than the incremental IncSpan algorithm. Here, the input sequences have been varied in certain time intervals. The generated number of sequences shows better results in our proposed approach is given in Fig. 8. In figure 9, the corresponding execution time of the CSSF-Miner shows better results (time=23 for 40 support values) than the IncSpan algorithm. The effective usage of the memory in the proposed algorithm is shown in figure 10.

V. CONCLUSION

In this paper, we have proposed algorithm for Mining of constraint sequential patterns from progressive database. In order to efficiently capturing the dynamic nature of data addition and deletion into the mining problem, initially, we construct the updated CSSF-trie from the static database. Then, the database gets updated from the distributed sources; data in the database may be static, inserted, or deleted. Whenever the database is updated from the multiple sources, CSSF trie is also updated by including the updated sequence. Then, the updated CSSF-trie is used to mine the progressive CSSF-patterns using the proposed trie pattern mining algorithm. Finally, the experimentation is carried out using the synthetic and real life distributed databases that are given to the progressive CSSF-miner using thread environment. The experimental results and analysis provides better results in terms of the generated number of sequential patterns, execution time and the memory usage over the existing IncSpan algorithm.

REFERENCES
