Tree Based Sequential Pattern Mining

Ashin Ara Bithi
Dept. of Computer Science & Engineering
University of Dhaka

Manira Akhter
Dept. of Computer Science & Engineering
University of Dhaka

Abu Ahmed Ferdaus
Assistant Professor
Dept. of Computer Science & Engineering
University of Dhaka

Abstract— **Sequential pattern mining is an important research area in data mining field for extracting useful knowledge from sequence databases. In this paper, we have proposed a Tree based sequential pattern mining algorithm which can generate sequential patterns from the fast updated sequential pattern tree (called FUSP-tree) structure by recursively creating set of small trees from the large tree. FUSP-tree stores complete set of sequences with only frequent items, their frequency and links for the given sequence database. So, it can generate the complete set of sequential patterns without generating any unnecessary candidate sequences and without repeated scanning the original databases. We have compared our proposed approach with two state-of-the-art algorithms and our performance study shows that, for static datasets, Tree based sequential pattern mining algorithm is much faster than existing apriori based GSP algorithm and it is also faster than existing PrefixSpan algorithm which is pattern growth approach.**

Keywords: Sequential Pattern, Data Mining, FUSP-tree, Sequence Database, Frequent Pattern.

I. INTRODUCTION

Sequential pattern mining in transactional databases plays an important role in data mining field. Sequential pattern mining means discovering all the frequently occurring ordered events or subsequences from sequence databases. Sequential pattern mining is widely used in the analysis of customer shopping behavior, web access patterns, in the analysis of biological sequences, sequences of events in science and engineering, and in natural and social developments. Agrawal and Srikant first introduced sequential pattern mining in 1995 [1]. Based on their study, sequential pattern mining is stated as follows: "*Given a sequence database or a set of sequences where each sequence is an ordered list events or elements and each event or element is a set of items, and given a user-specific minimum support threshold or min_sup, sequential pattern mining is the process of finding the complete set of frequent subsequences, that is, the subsequences whose occurrence frequency in the set of sequences or sequence databases is greater than or equal to min_sup.*" Past studies developed two major classes of sequential pattern mining methods. First class proposed several mining algorithms [1] [2] [3] based on apriori property which states that, every nonempty subsequences of a sequential pattern are also a sequential pattern. Among them, GSP [2] and SPADE [3] are most efficient apriori- based algorithms. Both of them find all sequential patterns by using level-wise candidate sequences generate and test approach which increases the time and space complexity. Another class proposed algorithms like FreeSpan [4] and PrefixSpan [5] based on pattern growth approach. Pattern growth approach does not generate any candidate sequences like apriori based methods GSP and SPADE, but it creates lots of projected databases and each time it needs to scan the projected databases to find the frequent items. In this paper, we present a Tree based sequential pattern mining algorithm which mines the patterns from the tree structure. This algorithm first generates a FUSP-tree which is proposed by Lin in 2008 [6], from original sequence database to store only frequent items. For this reason, the large database is compressed into a smaller data structure. When frequent items are not changed, the approach doesn't need to rescan the original database once the tree is created and can get the results only from the tree. Then, it recursively projects the FUSP-tree into a set of smaller projected trees based on the current sequential patterns and sequential patterns are grown by exploring only locally frequent fragments in each projected tree. It finds all sequential patterns without generating any candidate sequence and links stored in the FUSP-tree help it to find the frequent items easily without scanning each projected trees.

In the rest of the paper, section II describes related works; section III introduces our concept of Tree based mining with an example. Performance analysis is shown in section IV and finally section V draws conclusion that points out the potentiality of our work.

II. REVIEW OF WORKS

We have studied a set of mining approaches to understand the effectiveness of pattern discovery in data mining field. Some of them are described sequentially in this section.

A. FP-Growth Algorithm

FP Growth algorithm [7] was proposed by Han, Pie & Yin in 2000 to mine the frequent patterns from the tree structure without candidate set generation. They proposed two novel algorithms, one is FP-tree construction algorithm to keep the large database in a compact form and another is FP-growth algorithm to mine the frequent patterns from the FP-tree. FP-tree structure is a compress tree structure which stores only frequent items in the tree. The construction process of the FP-tree was subdivided into two parts. First, it scans the original database to find the frequent items and their support-count, after that; it scans again the database from first transaction to last to construct the tree. The links between parent node and child node are singly directed. A header table was also kept to store the frequent items and the links to the first occurrence of those items into the tree. After the tree is completed, the actual mining algorithm (FP-growth) is initiated to find the frequent patterns recursively from the tree. The recursive process is started from the lowest frequent items in the header table.

B. GSP Algorithm

GSP (Generalized Sequential Patterns) [2] is a sequential pattern mining algorithm which was proposed by Srikant and Agrawal in 1996. GSP is an Apriori based algorithm. It generates lots of candidate sets and it tests them by multiple passes. The algorithm to find the sequential patterns is outlined as follows: **First**, it scans the database to find the frequent items, that is, those with equal or greater than minimum support. All of those frequent items are length-1 frequent sequences. **Second**, each of them starts with a seed set of sequential patterns to generate new potentially sequential patterns, called candidate sequences. Each candidate sequence contains more than one item from which pattern it is generated. The length of each sequence is the number of instances of items in a sequence. All of the candidate sequences have the same length in a given pass. To find the frequent sequence, the algorithm then scans the database and discards those candidates which are infrequent. **Finally**, after getting the frequent sequences it makes those sequences as the seed for the next pass. The algorithm terminates, when there are no frequent sequences at the end of a pass, or when there are no candidate sequences generated.

C. PrefixSpan Algorithm

PrefixSpan [5] is a projection-based, sequential pattern-growth approach for efficient and scalable mining of sequential patterns, which is an extension of FP-growth [7]. Unlike apriori-based algorithms it does not create large number of useless candidate sets and generates complete set of sequential patterns from large databases efficiently. The major cost of PrefixSpan is database projection, i.e., forming projected databases recursively. To find the sequential patterns, PrefixSpan recursively projects a sequence database into a set of small projected databases and sequential patterns are grown in each projected database by exploring only locally frequent fragments. In this approach, sequential patterns from sequence database can be mined by a prefix-projection method in the following steps: (1) Find length-1 sequential patterns. Scan database once to find all the frequent items in sequences. Each of these frequent items is a length-1sequential pattern. (2) Divide search space. The complete set of sequential patterns can be partitioned according to the number of length-1 sequential patterns (prefixes) found in step-1. (3) Find subsets of sequential patterns. The subsets of sequential patterns can be mined by constructing the corresponding set of projected databases and mining each recursively.

III. Proposed Approach

Here we have proposed our Tree based sequential pattern mining approach which is based on FUSP-tree structure. Our objective is to find the sequential patterns from the tree by generating set of small projected trees from the large tree recursively. For understanding about the FUSP-tree, a brief overview of this tree structure is given below:

A. FUSP Tree Strucure

To efficiently mine the sequential patterns, Lin et al.2008 proposed the FUSP-tree [6] structure and its maintenance algorithm. FUSP-tree consists of one root node labeled as 'root' and a set of prefix subtrees as the children of the root. Each node in the prefix subtrees contains ***item-name***; which represents the node contains that item, ***count***; the number of sequences represented by the section of the path reaching the node, and ***node-link***; links to the next node of that item in the next branch of the FUSP-tree. The FUSP-tree contains a Header-Table which store frequent item, their count and the link of first occurrence in the tree of that item. This table helps to find appropriate items or sequences in the tree. The construction process is similar to FP-tree [7] i.e. the construction process is executed tuple by tuple from first sequence to last. But the differences from FP-tree are, the link between two nodes is symbolized by's' or 'i' as like IncSpan [8]. Here, symbol 's' indicates the sequence relation between two different events in a sequence and symbol 'i' indicates the itemset relation between two items in a event and also the links are bidirectional which will help to update process easier. For example, in Table 1 sequence database, the first sequence is "10: < {a}{a b c}{a c}{d}{c f}>". This sequence has five events or elements, namely {a}, {a b c}, {a c}, {d}, {c f}. First event contain only one item {a}. Item {a} is the first node < a: 1> of the tree which is linked as a child of the root and the link between root and first node is marked as 's'. Second event contains three items. So, for this event, there are three nodes <a: 1>, <b: 1>, < c: 1>. Node <a: 1> is linked to the first node <a: 1 > of the tree and the link between them is marked as 's'. Node <b: 1> is linked with second node <a: 1 > and node <c: 1> is linked to node <b: 1 >. Both links between node <a: 1> and<b: 1>, and, node<b: 1> and <c: 1> are marked as 'i' respectively. This four nodes also store in the Header-Table. In this similar way, the whole tree is constructed. The FUSP-tree for the Table 1 sequence database is shown in Figure 1.

B. The Algorithm

1) Notation

Here, **k** = Length of sequential pattern, **α** = Sequential pattern, **s** = Label of edge between two different events, **i** = Label of edge between two items in the same event, **f** = Frequent item.

2) The Proposed Algorithm

The details of the proposed Tree based sequential pattern mining algorithm are described as a pseudo code below:

Input: A sequence database which contains customer-id and customer sequence and a minimum support threshold min_support.

Output: The complete set of sequential patterns.

Algorithm: (Tree Based Sequential Pattern Mining)

Step 1: Scan the sequence database to find length-1 sequential patterns and their counts.

Step 2: Construct the FUSP-tree and its corresponding header table. The construction process is described above in this section.

Step 3: Call, **Tree Based Sequential Pattern Mining** (FUSP-tree, k, Header-Table)

```
{
 for each length-k sequential pattern α from header table
 {
  Generate new projected FUSP-tree and header table with new frequent item f;
  for each edge which prefix is α of the old FUSP-tree
 {
    if the label of the edge is 's' then
    Append (f) to the α to form the new (k+1) sequential pattern $\alpha'$;
   else if the edge label is 'i' then
   Assemble f to the last event of α to form new (k+1) sequential pattern $\alpha'$;
   Print $\alpha'$;
 }
 Call, Tree Based Sequential Pattern Mining (new FUSP-tree, k+1, new header table);
}
 }
```

C. An Example

For proper understanding of our proposed approach, in this section, we will try to describe our algorithm step by step with the help of an example. As for input, our algorithm just takes a sequence database and minimum support threshold, min_support.

Table 1: Sequence Database [5]

Sequence ID	Sequence
10	<{a}{a b c}{a c}{d}{c f}>
20	<{a d}{c}{b c}{a e}>
30	<{e f}{a b}{d f}{c}{b}>
40	<{e}{g}{a f}{c}{b}{c}>

In our example, we have used the above sequence database which is shown in Table 1 and we can see that, the table contains four sequences and seven items which are: {a, b, c, d, e, f, g}. And let the minimum support is 50% means for four sequences, the minimum support is (4*0.5) = 2.

Table 2: Length-1 Patterns and Their Support Counts

Length-1 Sequential Patterns	Support Count
a	4
b	4
c	4
d	3
e	3
f	3

The algorithm's steps are described below using the above database:

Step1: Scan the sequence database shown in Table 1 to find the length-1 sequential patterns and their counts. After this step we get the length-1 sequential patterns. The length-1 sequential patterns and their corresponding counts are given in Table 2.

Step 2: Again scan the sequence database to create the FUSP-tree and the Header Table. We have described the construction process of FUSP-tree above in this section using the same database. After this step we get the final FUSP-tree which is shown in Figure 1.

Step 3: After the FUSP-tree is constructed, the mining process will be executed to get the sequential patterns from the tree. The mining process is briefly described below:

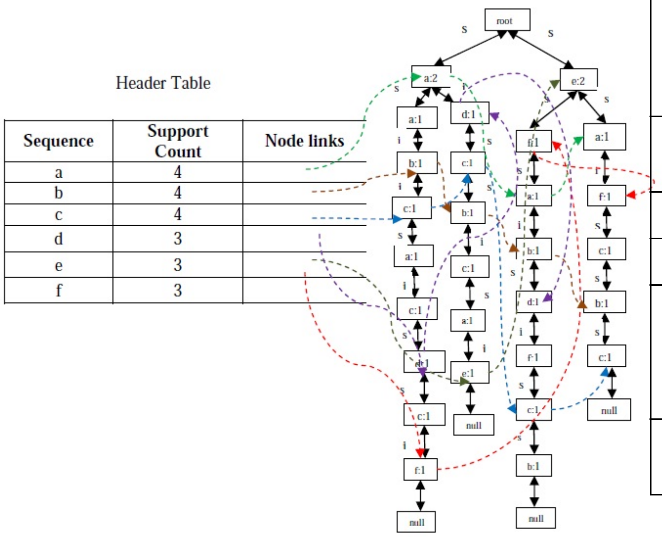
Sub Step1: From the header table we first pick the top prefix item <{a } > and by using its link we find the first occurrence of that item in the FUSP-tree. From the Figure 1 we see that first occurrence of <{a} > has two child nodes, so we get two branches which are shown in figure 2. From the first node of < {a} >, we get the next link for the same item's node and through this node we get another branch, so we get three branches for prefix < {a}> which are shown in Figure 3 and by this way we get all the branches of the new projected tree prefixed with < {a}> which are shown in Figure 4. All the length-2 sequential patterns for < {a}>-projected tree are found by concatenation of prefix < {a}> with the frequent sequences shown in the Header Table of Figure 4. Here, '-b' means item in the same event and only 'b' means item in different events. So, the length-2 sequential pattern for '-b' is <{a b}>:2 and for only 'b' is <{a}{b}>:4 and similarly others are: <{a}{a}>:2, <{a}{c}>:4, <{a}{d}>:2, <{a}{f}>:2. By applying similar process in < {a}> projected FUSP-tree of Figure 4, we get the <{a}{a}>, <{a}{b}>-projected FUSP-trees shown in figure 5 and 6 respectively and so on. But <{a}{a}> tree has no frequent item. So, we don't get any sequential pattern from this tree. We can get the length-3 patterns from <{a}{b}>, <{a}{c}>, <{a b}>, <{a}{d}>, <{a}{f}>-projected trees and then we can get length-4 patterns by constructing the projected trees from the length-3 prefix projected trees and then length-5,6,7 and so on, while header tables with frequent items are also found for the corresponding projected trees.

Sub Step 2: By applying Sub Step 1 in Figure 1, we can get the <{b}>, <{c}>, <{d}>, <{e}>, <{f}>-projected FUSP-trees and their sequential patterns recursively.

The approach described above generates complete set of sequential patterns without creating any candidate set which are shown in Table 3.

Table 3: The Resulting Sequential Patterns with prefix <a>, <b>, <c>, <d>, <e> and <f>

<a>: <{a}>, <{a}{a}>, <{a}{b}>, <{a}{b c}>, <{a}{b c}{a}>, <{a}{b}{a}>, <{a}{b}{c}>, <{a b}>, <{a b}{c}>, <{a b}{d}>, <{a b}{f}>, <{a b}{d}{c}>, <{a}{c}>, <{{a}{c}{a}>, <{a}{c}{b}>, <{a}{c}{c}>, <{a}{d}>, <{a}{d}{c}>, <{a}{f}>
<b>: <{b}>, <{b}{a}>, <{b}{c}>, <{b c}>, <{b c}{a}>, <{b}{d}>, <{b}{d}{c}>, <{b}{f}>
<c>: <{c}>, <{c} {a}>, <{c} {b}>, <{c} {c}>
<d>: <{d}>, <{d}{b}>, <{d}{c}>, <{d}{c}{b}>
<e>: <{e}>, <{e}{a}>, <{e}{a}{b}>, <{e}{a}{c}>, <{e}{a}{c}{b}>, <{e}{b}>, <{e}{b}{c}>, <{e}{c}>, <{e}{c}{b}>, <{e}{f}>, <{e}{f}{b}>, <{e}{f}{c}>, <{e}{f}{c}{b}>
<f>: <{f}>, <{f}{b}>, <{f}{b}{c}>, <{f}{c}>, <{f}{c}{b}>


Header Table

Sequence	Support Count	Node links
a	4	
b	4	
c	4	
d	3	
e	3	
f	3	

Figure 1: FUSP-Tree of Table 1

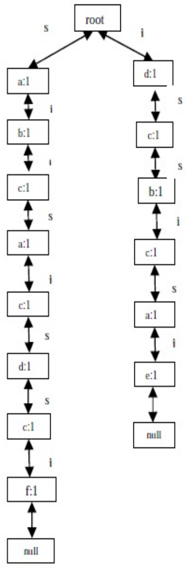


Figure 2: Two Projected Branches for First occurrence of < {a}>

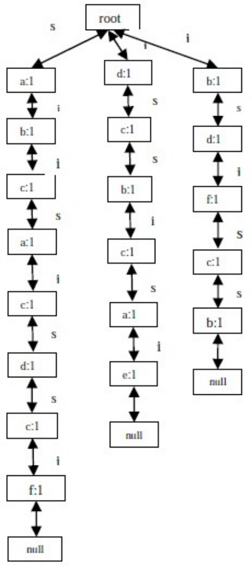
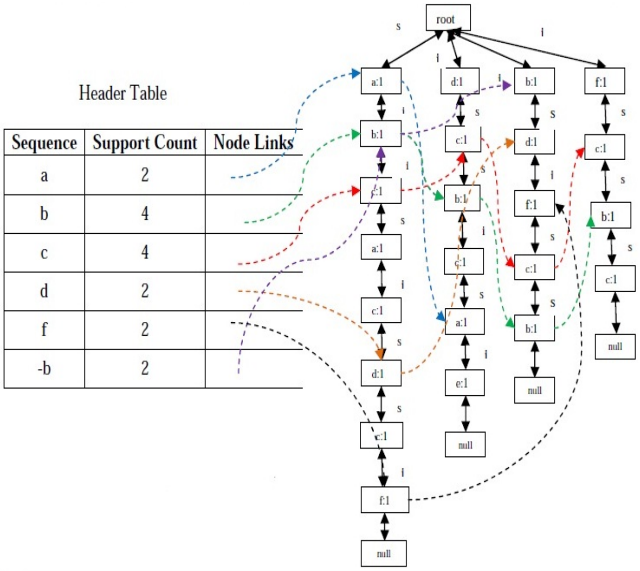


Figure 3: Three Projected Branches for Prefix < {a}>



Sequence	Support Count	Node Links
a	2	
b	4	
c	4	
d	2	
f	2	
-b	2	

Figure 4: The < {a}> Projected Tree and its Corresponding Header Table

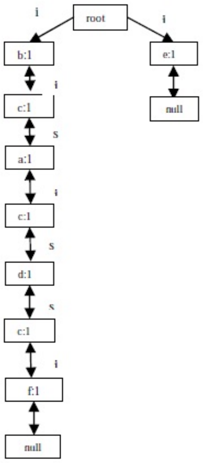
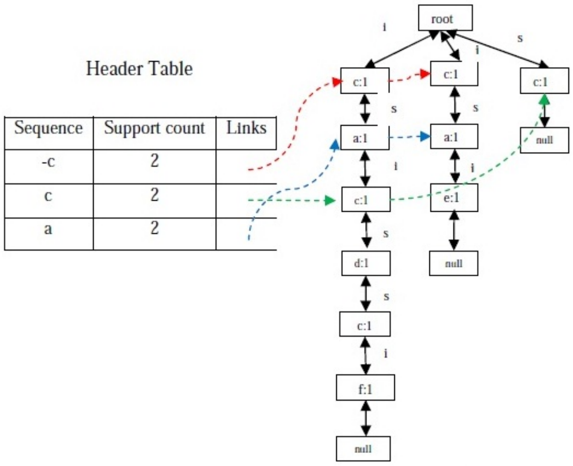


Figure 5 : <{a} {a}> Projected Tree



Sequence	Support count	Links
-c	2	
c	2	
a	2	

Figure 6 : <{a} {b}> Projected Tree and its Corresponding Header Table

IV. PERFORMAANCE ANALYSIS

Here, we represent a performance comparison of proposed Tree based sequential pattern mining approach with GSP and PrefixSpan for two datasets. All the experiments were conducted on a 2.27-GHz Intel core^{TM} i3-330 processor with 3GB main memory, running on Microsoft Windows 7. All the programs were written in NetBeans IDE 7.0 M2 with JDK 5. We did not directly compare our data with those in some published reports running on different machines. Instead, we also implemented GSP and PrefixSpan algorithms to the best of our knowledge based on the published reports on the same machine and compared in the same running environment.

A. Datasets

We have used two datasets, Mushroom [9] and Chess [9] for evaluation of experimental results. Usually these datasets are used for generating frequent patterns, since in our study we are working on sequential pattern mining, so we have used these datasets by considering each transaction as a sequence and each item of the transaction as a single item element in that sequence. Obviously, while considering these datasets for sequential pattern mining, they will also generate long sequential patterns. The Properties of these datasets, in terms of the number of distinct items, the number of sequences, the maximum sequence size, and the average sequence size, are shown below by a Table 4.

Table 4: Properties of Experimental Datasets

Dataset	Items	No. of Sequences	Max Size	Avg Size
Mushroom	119	8124	23	23.0
Chess	75	3196	37	37.0

B. *Experimental Result*

Comparisons between GSP, PrefixSpan and Tree based mining algorithm for different minimum support threshold values for these datasets are shown below:

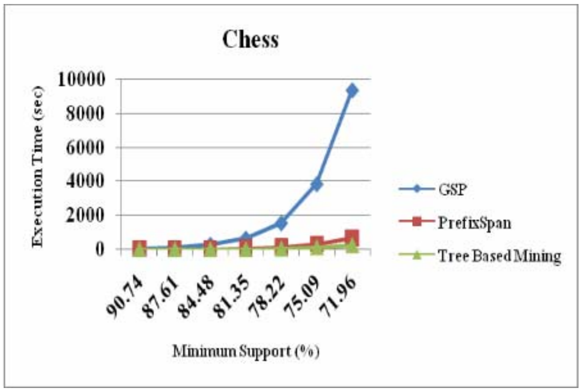


Figure 7: Comparisons between execution time and minimum support for Chess dataset

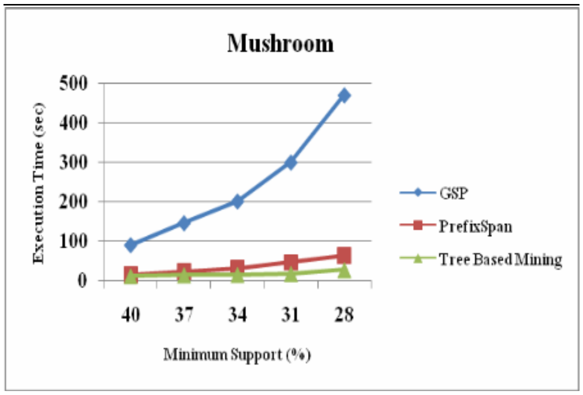


Figure 8: Comparisons between execution time and minimum support for Mushroom dataset

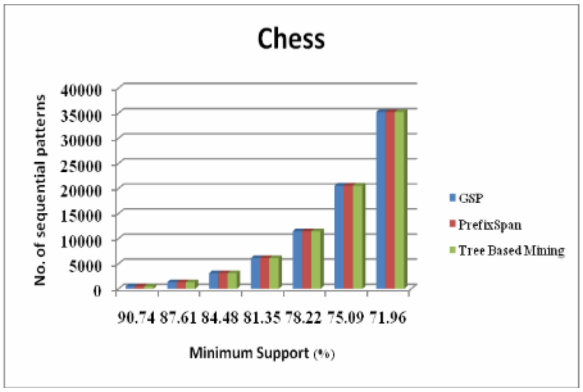


Figure 9: Comparisons between No. of Sequential Patterns and Minimum Support for Chess dataset.

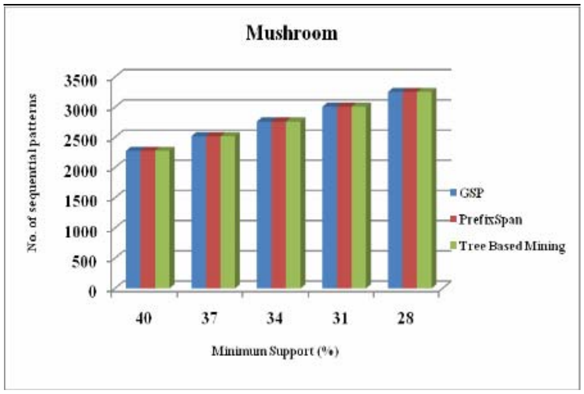


Figure 10: Comparisons between No. of Sequential Patterns and Minimum Support for Mushroom dataset.

The experimental results shown in Figure 7 and 8 are depicted to show the execution time of the three algorithms at different support thresholds. It can be observed from Figure 7 and 8 that, the execution times maintain the order "Tree based approach < PrefixSpan < GSP" when Chess and Mushroom datasets are used respectively. Thus, we can conclude that, our Tree based sequential pattern mining approach performs much better than GSP algorithm and also our approach better than PrefixSpan. This is also to be mentioned that, the proposed Tree based approach generates same number of sequential patterns for different minimum support thresholds as generated by GSP and PrefixSpan algorithms that are shown in Figure 9 and 10.

V. CONCLUSION

In this paper, a Tree based sequential pattern mining algorithm is proposed, where a large tree is recursively projected into a set of small projected trees and grows sequential patterns in each projected tree by exploring only locally frequent fragments. This algorithm mines the complete set of sequential patterns without generating any candidate sequences. So, it reduces the effort of candidate sequences generation. Links stored in the FUSP-tree help it to find the frequent items easily without scanning each projected trees. So, it also reduces the repeated scanning of database. A comprehensive performance study shows that, for static databases, this algorithm always outperforms GSP which is apriori based and it also outperforms pattern growth based PrefixSpan algorithm. In this study, we have designed our method to work only for static datasets and as shown above, we have achieved satisfactory outcome. But scopes are there to improve the algorithm to handle the dynamic databases. In future study, we will extend our algorithm for dynamic databases and hope it will give better performance than GSP and PrefixSpan.

REFERENCES

[1] Agrawal R and Srikant R, “Mining Sequential Patterns”, in Int'l. Conf. Data Engineering (ICDE 95), pp.3-14, 1995.

[2] Srikant R, Agrawal R: “Mining Sequential Patterns: Generalizations and Performance Improvements”, in Int'l Conf Extending Database Technology. Springer pp.3-17, 1996.

[3] M. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences,” Machine Learning, vol. 40, pp. 31-60, 2001.

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu,“FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining,” Proc. 2000 ACM SIGKDD Int’l Conf. Knowledge Discovery in Databases (KDD ’00), pp. 355-359, Aug. 2000.

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, “PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth,” Proc. 2001 Int’l Conf. Data Eng. (ICDE ’01), pp. 215-224, Apr. 2001.

[6] C. W. Lin, T. P. Hong, Wen-Hsiang Lu and Wen-Yang Lin, “An Incremental FUSP-Tree Maintenance Algorithm,” The Eighth International Conference on Intelligent System Design and Application, pp.445-449,2008.

[7] J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation,” The 2000 ACM SIGMOD International Conference on Management of Data, pp. 1-12, 2000.

[8] H. Cheng, X. Yan and J. Han, “Incspan: Incremental Mining of Sequential Patterns in Large Database,” The ACM SIGKDD international conference on Knowledge discovery and data mining, pp.527-532, 2004.

[9] Web link: Frequent Itemset Mining Implementations Repository
http://fimi.cs.helsinki.fi/ (last accessed on 19 December, 2012)