IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 1, Ver. VIII (Feb. 2014), PP 110-116
www.iosrjournals.org

Incremental Sequential Pattern Tree Mining

Ashin Ara Bithi[1], Z.M. Tarek Shahriar[2], Abu Ahmed Ferdaus[3]
[1]*(Dept. of Computer Science & Engineering, Asian University of Bangladesh, Bangladesh)*
[2]*(Dept. of Computer Science & Engineering, Asian University of Bangladesh, Bangladesh)*
[3]*(Assistant Professor, Dept. of Computer Science & Engineering, University of Dhaka, Bangladesh)*

Abstract: *In this paper, we have proposed an Incremental Sequential Pattern Tree mining algorithm to retrieve new updated frequent sequential patterns from dynamic sequence database. Sequential Pattern Tree stores both frequent and non-frequent items from the old sequence database. So that, our proposed algorithm updates the old Sequential Pattern Tree by scanning only the new sequences, does not require to scan the whole updated database (old + new) that reduces the execution time in reconstructing the tree. We have compared our proposed incremental mining approach with three existing algorithms those are GSP, PrefixSpan and FUSP-Tree Based mining and we have got satisfactory results.*

Keywords: *Data Mining, Sequence Database, Sequential Pattern, Sequential Pattern Mining, Frequent Patterns, Tree Based Mining, Incremental Mining.*

1. INTRODUCTION

Data mining provides a proficient way to extract useful information and helpful knowledge from large amount of data that are explored in a gigantic manner day by day. Sequential pattern mining in transactional databases plays an important role in data mining field since it concentrates mostly on modeling customer purchase behavior. It is the process of finding the complete set of frequent occurring ordered events or subsequences from a set of sequences or sequence database. The advantage to find the sequential patterns is, we can find the customer sequences and predict the probability to buy some items in next transactions by the customers. For example, if a customer bought bread and egg in one transaction, then, we can predict the probability to buy milk by this customer in the next: that is, if {bread, egg} then {milk}. It is widely used in the analysis of customer purchase patterns or web access patterns, sequencing or time-related processes such as scientific experiments, natural disasters, and in DNA sequences, etc. 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 database 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. During mining, the apriori based sequential pattern mining algorithms generate huge number of candidate patterns and these algorithms scan the whole database multiple times which increase both the time and space complexity. Another class proposed algorithms like FreeSpan [4] and PrefixSpan [5] that are based on pattern growth approach. Pattern growth approaches can find the complete set of frequent sequential patterns without generating any candidate patterns. But, they recursively generate lots of projected databases during mining and they scan each projected database numerous times to find the frequent items from projected database which are both space and time consuming. The entire discussed sequential pattern mining algorithms {[1], [2], [3], [4], and [5]} are inefficient for dynamic database. Because, they work in a one-time fashion: mine the entire database and obtain the complete set of frequent sequential patterns. However, in many applications, databases are updated incrementally. For example, customer shopping transaction database is updated daily due to the appending of newly purchased items from existing customers or insertion of new shopping sequences from new customers. The sequential patterns for the old database may become invalid on the updated database, so, it requires mining the updated database for the new subsequences. But, it is not proficient way to mine the updated databases from scratch by scanning the whole updated database (old + new). However, FUSP-Tree [6] structure works efficiently for incremental mining due to its tree structure. But, during mining [7], FUSP-Tree generates lot of projected trees and it scans the whole database two times. Also, FUSP-Tree stores only frequent items into the tree. So, during incremental mining, when any infrequent item in the old database become frequent in the updated database, in this situation, this algorithm rescans the old database to find the count of the infrequent item from the old database, i.e., it scans the whole updated database (old + new) to build the new updated FUSP-Tree. In this paper, we have developed an efficient incremental mining algorithm that updates the Sequential Pattern Tree [8] by scanning only the new sequences. Sequential Pattern Tree [8] structure stores both frequent and non-frequent items from the old sequence database. So, when new sequences appear, our algorithm updates the old Sequential Pattern Tree by scanning only the new sequences, does not require scanning the whole updated database (old + new) and then, the updated new Sequential Pattern Tree is mined from the beginning to retrieve the new frequent sequential patterns without re-constructing the intermediate projected trees and candidate sequences. To build this new tree structure, we need only one scan of new sequences due to storage of both frequent and non-frequent items in the tree that reduces the tree reconstruction time drastically. The technique proposed for incremental mining in this paper present a much better performance than that achieved by GSP [2], PrefixSpan [5], and FUSP-Tree Based Mining [7] techniques.

In the rest of the paper, section II describes related works; section III introduces our concept of Incremental Mining with 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.

1.1 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. When new sequences come, GSP starts its mining process from the beginning. That means, GSP scans the whole updated database (old + new) during incremental mining so that multiple scanning of database and candidate sequences generation increase as the size of the database increases.

1.2 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 [9]. 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-1 sequential 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. During incremental mining, PrefixSpan mines the updated database from the scratch by scanning the whole updated database (old + new). So that, projected database generation and scanning of each projected database increase at the same time as the size of the database raises.

1.3 FUSP – Tree Algorithm

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 frequent 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 node 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

[9] i.e. the construction process is executed tuple by tuple from first sequence to last. To create this tree, it requires two scans of large database which increases the tree construction time. Mining process of FUSP-Tree [7] is almost similar to PrefixSpan [5] and FP-growth [9] algorithms. After the FUSP-Tree [6] is maintained, the final frequent sequences can then be found by a recursive method from the tree. This method finds the sequential patterns from the FUSP-Tree structure by generating set of small projected trees from the large tree recursively. It generates no candidate sets but it generates many projected trees for each prefix sequence which require more memory [7]. During incremental mining, when any infrequent item in the old database become frequent in the updated database, in this situation, this algorithm rescans the old database to find the support of the infrequent item from the old database, i.e., it scans the whole updated database (old + new) to update the old FUSP-Tree structure due to storage of frequent items only. So that scans of database increases at the same time as the size of the database raises.

III. PROPOSED APPROACH

Here, we have described our proposed Incremental Sequential Pattern Tree Mining approach with example for finding new updated frequent sequential patterns from the updated sequence database. The algorithm for Incremental Sequential Pattern Tree mining is given in Algorithm 1.

Algorithm 1: (Incremental Sequential Pattern Tree Mining: mining new updated frequent subsequences from updated sequence database)

Input: Old Sequential Pattern Tree, New Sequence Database and Minimum Support Threshold (min_sup).

Output: The complete set of new updated frequent sequential patterns.

Method:

1. Scan only new sequence database once and update the old Sequential Pattern Tree using Algorithm 2.
2. Then, recursively mines the new updated Sequential Pattern Tree to find the new updated frequent sequential patterns using mining algorithm proposed in [8].

1.4 Incremental Sequential Pattern Tree Mining

In our study, we have extended Sequential Pattern Tree structure for incremental sequential pattern mining process. When new sequences come, each new sequence is inserted as branch into the old Sequential Pattern Tree and the items in the old Header Table is also updated using the Algorithm 2 presented in this section. When sequence database is updated with new sequences, it is not required to scan the entire sequence database (old + new), only needs to scan the new sequences to update the existing Sequential Pattern Tree and Header Table that reduces the reconstruction time of the tree. Then, initiate the sequential pattern mining process from the beginning using the Algorithm that present in [8] to find the new updated frequent sequential patterns from the updated Sequential Pattern Tree.

Algorithm 2: (Construction of Incremental Sequential Pattern Tree from new Sequence Database)

Input: New Sequence Database, Old Sequential Pattern Tree T with old Header Table.

Output: New Updated Sequential Pattern Tree, T′ with updated Header Table.

Method:

```
1         Scan the new sequence database.
2         Initially, set current _node = root of T.
3         for each new sequence $S_i$ till the end of new sequence database
3.1         for each event $e_j$ in $S_i$
3.1.1         for each item I in the $e_j$
3.1.1.1         if current_node has a child node c which c.label = I and c.transaction ID = j, then set c.count += 1 and
                current_node = c.
3.1.1.2         Otherwise,
3.1.1.2.1         Create a New node label with I
3.1.1.2.2         New node.count = 1.
3.1.1.2.3         New node.transaction ID = j.
3.1.1.2.4         Store New node in the current_node's successor link.
3.1.1.2.5         Set current_node = New node.
3.1.1.3         end if
3.1.1.4         For the new branch of each distinct item I, increment the count of the corresponding item I in the
                Header Table of T if item I already exist in the old Header Table; otherwise, add item I in the Header
                Table of T and set count to 1.
3.1.2         end for
3.2         end for
4         current_node = root of T.
5         end for
```

1.5 Example Incremental Sequential Pattern Tree Mining

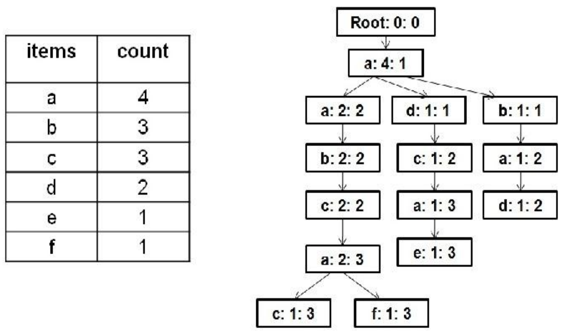
In this section, we have described our proposed incremental sequential pattern mining approach with a suitable example. Old Sequential Pattern Tree for the old sequence database shown in Table 1 is given in Fig 1 [8]. Table 2 shows new sequence database that is appended with the old sequence database shown in Table 1. Scan only the new sequence database once and insert the first sequence (ad)c(ae) as existing branch into the old tree shown in Fig 1. Only the count of each node in this branch increment. The next sequence f(bc)(ae) is inserted into the old tree as a new branch using Algorithm 2. Create a node (f: 1: 1) and insert it as child node of the Root node and then, derives the branch "(f: 1: 1) → (b: 1: 2) → (c: 1: 2) → (a: 1: 3) → (e: 1: 3)". Header Table is also updated accordingly. After insertion of the second sequence of Table 2, we can find the complete new updated Sequential Pattern Tree and updated Header Table which is shown in Fig 2.

Table 1 Old Sequence Database

Sequence ID	**Sequences**
10	a(abc)(ac)
20	(ad)c(ae)
30	a(abc)(af)
40	(ab)(ad)

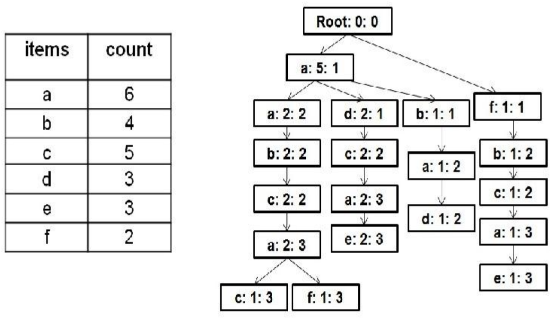
Table 2 New Sequence Database

Sequence ID	Sequences
50	(ad)c(ae)
60	f(bc)(ae)



items	count
a	4
b	3
c	3
d	2
e	1
f	1

Figure 1 Old Sequential Pattern Tree Along with Header Table for Table 1



items	count
a	6
b	4
c	5
d	3
e	3
f	2

Figure 2 Updated Sequential Pattern Tree Along with Updated Header Table

Suppose the minimum support threshold for the updated sequence database is 50% or 3 (6*50% = 3) as the total sequences for the updated sequence database is 6 (old + new). So, item e: 3 that is infrequent in the old database (4*50% = 2) become frequent in the updated database. Only item f: 2 is non frequent for the updated database. Then, initiate the sequential pattern mining process from the beginning of the Updated Sequential Pattern Tree shown in Fig 2 using the mining method that was proposed in [8]. The first item in the Header Table is 'a'. Find the first occurrence nodes labeled as 'a' using depth-first-search from each suffix branch of the Root node of the tree. The first occurrence nodes of item 'a' are (a: 5: 1) and (a: 1: 3) shown in Fig 2. The sum of counts of these nodes is 6 $\geq$ minimum support threshold. Transaction IDs of these nodes are not matched with the transaction ID of Root node, 0. So, the frequent sequential pattern is (a) and now the list of mined frequent sequential patterns is {(a): 6}. The mining of frequent 2-sequences that start with item 'a' would continue with the suffix trees rooted at node (a: 5: 1) and (a: 1: 3). Again, the first occurrence nodes labeled as 'a' from suffix tree of root node (a: 5: 1) are (a: 2: 2), (a: 2: 3) and (a: 1: 2) shown in Fig 2. The sum of counts of these nodes is 5 $\geq$ minimum support threshold. Transaction IDs of these nodes are not matched with the transaction ID of node (a: 5: 1). So, the frequent sequential pattern is (a)(a) and now the list of mined frequent sequential patterns is {(a): 6, (a)(a): 5}. No frequent 3-sequences exist for (a)(a) sequence. So, stop here. Backtrack and start again for item 'b' in the Header Table and find the first occurrence nodes (b: 2: 2) and (b: 1: 1) labeled as 'b' from the suffix tree rooted at node (a: 5: 1) shown in Fig 2. The sum of counts of these nodes is 3 $\geq$ minimum support threshold. Transaction ID of node (b: 1: 1) is matched with the transaction ID of node (a: 5: 1) and so, node (b: 1: 1) is considered as i-relation node of (a: 5: 1). The parent node of node (b: 2: 2) is (a: 2: 2) where label of node (a: 2: 2) is same as root node's label (a: 5:1). That's why node (b: 2: 2) is also considered as i-relation node of (a: 5: 1). So, (ab) is the new frequent sequential pattern and this time, the list of mined frequent sequential patterns is {(a): 6, (a)(a): 5, (ab): 3}. By continuing this process, the complete new updated frequent sequential patterns for the updated database are {(a): 6, (a)(a): 5, (ab): 3, (ab)(a): 3, (a)(c): 4, (a)(c)(a): 4, (ad): 3, (ae): 3, (b): 4, (b)(a): 4, (bc): 3, (bc)(a): 3, (c): 5, (c)(a): 5, (c)(ae): 3, (c)(e): 3, (d): 3, (e): 3}.

From the above discussion, we can conclude that, our proposed incremental approach updates the old tree by scanning only the new sequences once and finds new updated frequent sequential patterns by mining the updated tree from the beginning.

IV. PERFORMANCE ANALYSIS

Here, we represent a performance comparison of proposed Incremental Sequential Pattern Tree Mining approach with GSP, PrefixSpan and FUSP-Tree Based mining on both synthetic and real-life datasets. All the experiments were conducted on a 2.80-GHz Intel(R) Pentium(R) D processor with 1.5GB main memory, running on Microsoft Windows 7. All the programs were written in NetBeans IDE 6.8 with JDK 6. We did not directly compare our data with those in some published reports running on different machines. Instead, we also implemented GSP, PrefixSpan and FUSP-Tree Based mining algorithms to the best of our knowledge based on the published reports on the same machine and compared these four algorithms in the same running environment.

1.6 Datasets

We have used four datasets, three real-datasets, BMS-WebView-1 [10], BMS-WebView-2 [10], and BMS-POS [10], as well as a Synthetic dataset T10I4D100K [10] for evaluation of experimental results. We use 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, the average sequence size, and type are shown below by Table 3.

Table 3 Properties of Experimental Datasets

Dataset	Distinct Items	No. of Sequences	Max Size	Avg Size	Type
T10I4D100K	870	100000	29	10.1	Synthetic
BMS-WebView-1	497	59602	267	2.5	Real
BMS-WebView-2	3340	77512	161	5.0	Real
BMS-POS	1657	515597	164	6.5	Real

1.7 Experimental Result

In this section, we have presented our performance analysis between GSP, PrefixSpan, FUSP-Tree Based mining, and Incremental Sequential Pattern Tree Mining method for incremental mining by using above four datasets. During incremental mining, new sequences are added to the old sequence database i.e. size of the sequence database increases and minimum support threshold remain same.

Comparisons between four algorithms for different incremental databases from above four datasets are shown below.

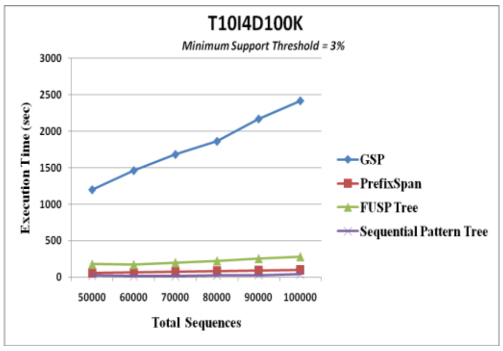


Figure 3 Comparison Between Execution Time and Various size of Datasets for T10I4D100K

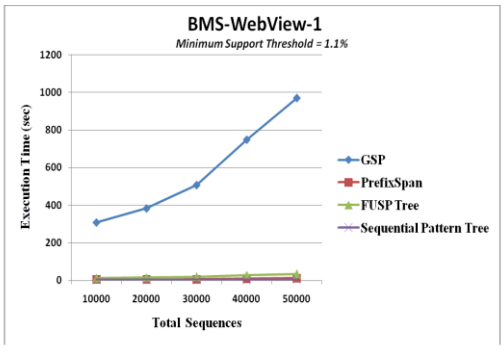


Figure 4 Comparison Between Execution Time and Various size of Datasets for BMS-WebView-1

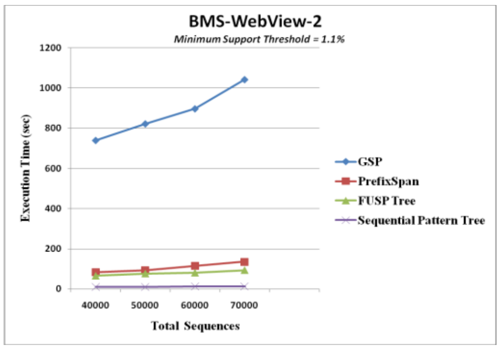


Figure 5 Comparison Between Execution Time and Various size of Datasets for BMS-WebView-2

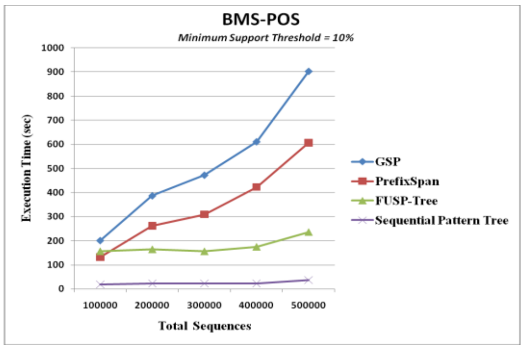


Figure 6 Comparison Between Execution Time and Various size of Datasets for BMS-POS

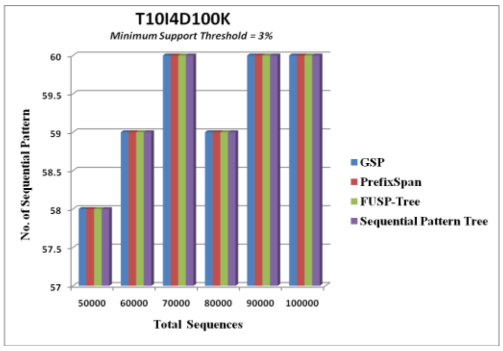


Figure 7 Comparison Between No. of Sequential Patterns and Various size of Datasets for T10I4D100K

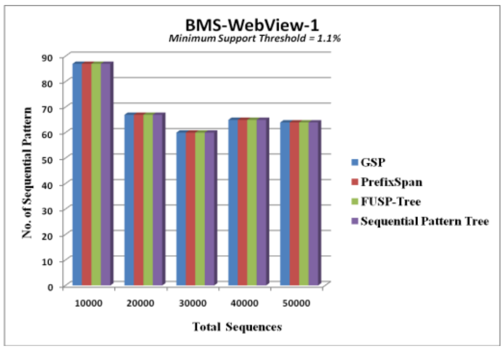


Figure 8 Comparison Between No. of Sequential Patterns and Various size of Datasets for BMS-WebView-1

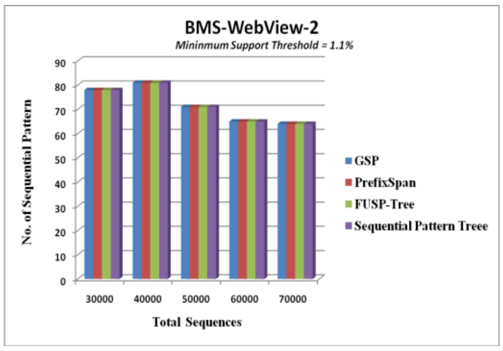


Figure 9 Comparison Between No. of Sequential Patterns and Various size of Datasets for BMS-WebView-2

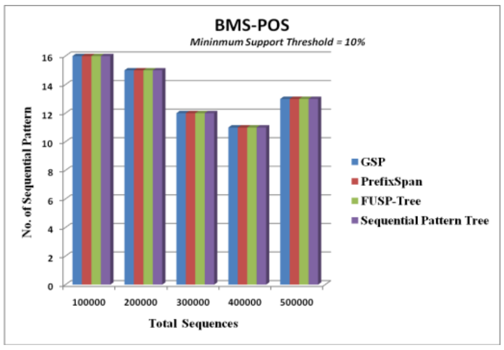


Figure 10 Comparison Between No. of Sequential Patterns and Various size of Datasets for BMS-POS

All the experimental results in Fig 3, 4, 5, and 6 are depicted to show the execution time of the four algorithms at various incremental databases. It can be observed from these Figures that, the execution times of our proposed incremental approach much better than GSP, PrefixSpan and FUSP-Tree Based mining for both synthetic and real-datasets. Thus, we can conclude that, our proposed approach achieves better performance than other three existing methods for incremental mining. Because, during incremental mining, our proposed approach updates the old Sequential Pattern Tree by scanning only the new sequences once that reduces the both scans of database and tree construction time considerably. This is also to be pointed out that, our proposed approach generates same number of frequent sequential patterns for different size of databases as generated by GSP, PrefixSpan, and FUSP-Tree Based mining algorithms that shown in Fig 7, 8, 9, and 10 respectively.

V. CONCLUSION

Though GSP, PrefixSpan, and FUSP-Tree Based mining find the entire frequent sequential patterns but for incremental database these approaches cannot work efficiently. GSP and PrefixSpan algorithms scan the both old large database and new incremental database to find the new updated frequent sequential patterns that means they start from the scratch so that both space and time complexity of these algorithms raise as the size of the database increases. FUSP-Tree Based mining algorithm scans the whole updated database (old + new) during tree construction time if infrequent items in the old database become frequent in the updated database that increases the FUSP-Tree construction time considerably. For this reason, we have proposed an incremental mining algorithm which updates the old Sequential Pattern Tree for the incremental database by scanning only the new sequences once and then, the updated Sequential Pattern Tree is mined to find the new frequent sequential patterns from the beginning. Because of storing both frequent and infrequent items in the Sequential Pattern Tree and scanning only the new sequences, our approach can reduce the scans of whole updated database as well as tree reconstruction time significantly. We can strictly notify that, our approach works well than GSP, PrefixSpan, and FUSP-Tree Based mining for dynamically updated database. Also, our approach can find out all the frequent sequential patterns like GSP, PrefixSpan and FUSP-Tree Based mining can do. We have cleared this expectation from the graphical result which is already shown in our performance analysis section.

REFERENCES

[1] R. Agrawal and R. Srikant, "*Mining sequential patterns*," in ICDE, P. S. Yu and A. L. P. Chen, Eds. IEEE Computer Society, 1995, pp. 3-14. [Online] Available: http://doi.ieeecomputersociety.org/10.1109/ICDE.1995.380415

[2] R. Srikant and R. Agrawal, "*Mining sequential patterns: Generalizations and performance improvements*," in EDBT, ser. Lecture Notes in Computer Science, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds., vol. *1057*. Springer, 1996, pp. 3-17. [Online] Available: http://dx.doi.org/10.1007/BFb0014140

[3] M. J. Zaki, "*Spade: An efficient algorithm for mining frequent sequences*," Machine Learning, vol. *42*, no. 1/2, pp. 31-60, 2001. [Online] Available: http://dx.doi.org/10.1023/A:1007652502315

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu, "*Freespan: frequent pattern-projected sequential pattern mining*," in KDD, 2000, pp. 355-359. [Online] Available: http://doi.acm.org/10.1145/347090.347167

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "*Prefixspan: Mining sequential patterns by prefix-projected growth*," in ICDE, D. Georgakopoulos and A. Buchmann, Eds. IEEE Computer Society, 2001, pp. 215-224. [Online] Available: http://doi.ieeecomputersociety.org/10.1109/ICDE.2001.914830

[6] C.-W. Lin, T.-P. Hong, W.-H. Lu, and W.-Y. Lin, "*An incremental FUSP-tree maintenance algorithm*," in ISDA, J.-S. Pan, A. Abraham, and C.-C. Chang, Eds. IEEE Computer Society, 2008, pp. 445-449. [Online] Available: http://doi.ieeecomputersociety.org/10.1109/ISDA.2008.126

[7] Bithi A. A., Akhter M., & Ferdaus A. A. "*Tree Based Sequential Pattern Mining*", IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: *2249-9555*, Vol. *2*, No.6, December 2012. [Online] Available: http://www.ijcsits.org/papers/vol2no62012/25vol2no6.pdf

[8] Bithi A. A & Ferdaus A. A. "*Sequential Pattern Tree Mining*", IOSR Journal of Computer Engineering (IOSR-JCE), e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 15, Issue 5, (Nov.-Dec. 2013), pp 79-89. [Online] Available: http://iosrjournals.org/iosr-jce/papers/Vol15-issue5/N01557978.pdf?id=7557

[9] J. Han, J. Pei, and Y. Yin, "*Mining frequent patterns without candidate generation*," in SIGMOD Conference, W. Chen, J. F. Naughton, and P. A. Bernstein, Eds. ACM, 2000, pp. 1-12. [Online] Available: http://doi.acm.org/10.1145/342009.335372

[10] Z. Zheng, R. Kohavi, and L. Mason, "*Real world performance of association rule algorithms*," in KDD, 2001, pp. 401-406. [Online] Available: http://portal.acm.org/citation.cfm?id=502512.502572