An Efficient and Novel Approach for Sequential Access Pattern Mining

Krishnakant P. Adhiya
Department of Computer Engineering, SSBT's College of Engineering & Technology, Bambhori, Jalgaon, Maharashtra, India
Email: kpadhiya@yahoo.com

Satish R. Kolhe
School of Computer Sciences, North Maharashtra University, Jalgaon, Maharashtra, India
Email: srkolhe2000@gmail.com

***Abstract*—Sequential access pattern mining aims to discover interesting and frequent patterns from web data. Most of the sequential pattern mining algorithms are mainly Apriori based and Pattern-growth based. Various algorithms based on Apriori based technique bear the cost of multiple scans of database. Some of the algorithms based on Pattern-growth technique such as PrefixSpan, requires construction of projected databases. WAP-tree based mining techniques require reconstruction of large numbers of intermediate WAP-trees during mining process, which is very costly. In this paper, we propose an efficient sequential access pattern mining algorithm, based on CSB-mine [1]. The proposed algorithm focuses on constructing Web Access Sequence (WAS) list, Unique Symbol (US) list, and generation of SAP table without using WAP trees at any stage. The algorithm eliminates the use of any separate single sequence testing algorithm and it does not need any extra data structure to find first appearance of each symbol, thus saving the space. Also use of compact data structure avoids the reconstruction of projection database, which also saves space and time. The experiments are carried on synthetic data set and we present the performance of proposed algorithm considering memory utilization and run time. Experimental results show that the proposed algorithm outperforms the PrefixSpan and CSB-mine. The results show significant improvement in average memory usage and 10% to 15% improvement in the run time.**

***Index Term*—web usage mining, sequential pattern mining, frequent patterns, prefixspan, CSB-Mine**

I. Introduction

Web mining is considered as use of data mining techniques to discover and extract information from web resources. Mainly there are three categories to carry out web mining task: web usage mining, web structure mining and web content mining. The web log mining, also called as web usage mining focuses on techniques that could predict the behavior of users while they are interacting with the web. The user's navigational behavior can be stored in web logs. These raw web logs are stored in various repositories such as client browsers, proxy servers or web servers and work as input source to carry out web usage mining process [2, 3].

Various algorithms of sequential pattern mining can be used to carry out mining task in web usage mining process [4]. Sequential pattern mining can be considered as knowledge discovery from the database, which can be done by discovering frequent sequences where ordering of elements (element means item or itemset) is important [4, 5]. Some researchers [1] proposed their design of personalized web recommendation system using those frequent patterns [6]. Generally sequential pattern mining algorithms are differentiated by: number of scans required for the database, process to generate and store candidate set of k-itemsets, number of candidate sets generated and a process to count the support value [4, 5, 7]. Run time and memory utilization are two important measures for performance evaluation of those mining algorithms [4]. There exists number of sequential pattern mining algorithms with different techniques. Two techniques that are primarily used by most of them are: Apriori based and Pattern-growth based (also called as FP-growth) [4, 8]. AprioriTid, AprioriAll, AprioriSome, GSP (Generalized Sequence Pattern) are Apriori based algorithms while WAP-mine, FreeSpan, PrefixSpan belong to pattern-growth technique. Most of Apriori based algorithms encounter the problems such as: multiple scans of databases, generation of explosive number of candidate sequences and difficulties at mining long sequential patterns [4, 8, 9, and 10]. FP-growth based algorithms such as PrefixSpan involves the construction of projected databases in various steps [4, 9]. All these processes may be costly in terms of memory and run time. The basic mining algorithm based on WAP-tree is the WAP-mine algorithm which needs only two scans of sequence database. The algorithm builds a tree at start and then number of intermediate trees for frequent subsequences. This results in utilization of more memory [4, 15].

Manuscript received July 1, 2015; revised September 21, 2015.

doi: 10.12720/jetwi.7.1.5-12

In this paper, we have presented an efficient sequential access pattern mining algorithm. It is based on CSB-mine (Conditional Sequence Base mine) [1]. Our proposed algorithm does not require the construction of initial WAP-tree and reconstruction of number of intermediate conditional WAP-trees. There is no need of generating sub conditional sequence bases separately. Also it eliminates the need for recursive mining, thus saving space and time.

The rest of the paper is organized as follows. In section II, we introduce a study of related work, section III explains motivation for the proposed work, section IV details our proposed algorithm with data structures used, section V presents performance evaluation of proposed mining algorithm and in section VI conclusion of the paper is mentioned

II. Related Work

The problem of sequential pattern mining in transaction database was first presented by Agrawal and shrikant in [7]. Sequential pattern mining algorithms mainly belong to two types of categories: Apriori based and Pattern-growth based [4, 8]. Some algorithms are based on early-pruning techniques [4].

Rakesh Agrawal *et al*. [5], presented two algorithms to find out all association rules: Apriori and AprioriTid. The Apriori algorithm is used to find frequent itemsets using candidate generation. It employs a level-wise search, where all the transactions are scanned to calculate the support count for each item. The candidates are generated at different levels by considering only those itemsets that satisfy the minimum support count. The algorithm runs with basic principle of: k-itemsets are used to generate candidate set of (k+1) itemsets using join and prune actions at different levels. In AprioiTid algorithm, the original database D is used only at first level to count the support value. In next levels, $\bar{C}_k$ is used. $\bar{C}_k$ is a set of items of the form < TID, all k-itemsets corresponding to TID value>. In first level i.e. k=1, $\bar{C}_1$ corresponds to original database D. But for next level, i.e. k=2, 3, 4,…n, $\bar{C}_k$ contains only those transaction (TID) entries which contain any candidate k-itemsets. So number of entries in C_k will be reduced and it will be less than number of transactions in original database D. Apriori based algorithms scans the database many times and generates huge number of candidates for larger itemsets.

Rakesh Agrawal *et al.* [7], presented AprioriAll and AprioriSome algorithms for mining sequential patterns in a large database. AprioriAll belongs to count-all algorithm family while AprioriSome belongs to count-some family. AprioriAll employs the counting of all large sequences (itemsets), whereas AprioriSome focuses only on counting the maximal sequences. In AprioriSome algorithm all the maximal sequences are not found in a single phase, rather it uses forward phase to count certain length sequences and all remaining sequences are counted in backward phase. In AprioriAll, new candidate sequences are generated from large sequences. The large sequences are determined using the support value of candidate sequences. It is shown that both algorithms perform better in different applications.

Ramakrishnan Srikant *et al.* [11], presented GSP (Generalized Sequential Pattern) algorithm, which deals the limitation of problem definition in previous Apriori based algorithms. GSP includes sliding time window and time constraints to generalize the problem definition. It performs multiple pass for generating candidate sequences. In each pass a new candidate sequence is generated using candidate sequence of the previous pass. For larger itemsets explosive number of candidates may be generated, so some frequent sequences will be stored on the disk. It means that the algorithm is not only main memory algorithm. It is shown that GSP performs better than AprioriAll [11, 4].

Jiawei Han *et al.* [12] presented a new sequential pattern mining algorithm called FreeSpan (Frequent pattern-projected Sequential Pattern mining). It focuses to reduce or to avoid the costlier candidate sequences generation. The basic concept of algorithm is to generate the projected database. The mining task requires three scans of sequence database. In the first step, the sequence database is scanned to create a set of frequent 1-sequences. In next step, the database is again scanned to construct a frequent item matrix. This matrix is then used to create a set of frequent 2-sequences and projected database. From this, a set of frequent 3-sequences will be generated, and so on. In this way the algorithm performs recursive mining on projected database and it terminates when there is no longer patterns remaining to be mined. It is shown that it outperforms GSP [12, 4].

Jian Pei *et al.* [9] proposed a new mining algorithm called PrefixSpan (Prefix-projected Sequential Pattern mining). The main advantage of this algorithm is that, no candidate sequences are generated at any stage of execution. First, the original sequence database is scanned to find a set of frequent 1-sequences. Next, projected database is constructed by considering only postfix subsequences. The projected database is constructed recursively to find set of frequent 2-length sequences, then 3-length sequences, and so on. Construction of projected database is the only major effort associated with this algorithm [9, 4, 13].

Jay Ayers *et al.* [14], proposed a algorithm called SPAM (Sequential Pattern Mining), particularly for longer patterns. The algorithm employs construction of vertical bitmap to represent the given dataset. Then this bitmap is used to generate the candidate sequences. All sequences in a sequence database are stored in a tree called as sequence tree. This sequence tree is traversed using depth-first strategy and authors claim to be the first one to use this strategy in sequential pattern mining. The algorithm uses two pruning methods: S-step pruning and I-step pruning for performance improvement. It is shown that SPAM performance is far better than PrefixSpan for longer patterns. [14, 4].

Jian Pei *et al.* [15], introduced an efficient sequential pattern mining algorithm for web logs, called WAP-mine using a WAP-tree (i.e. Web Access Pattern tree). The process of mining starts with preprocessing of web logs,

where main task is to obtain web access sequences user wise. Then the initial WAP-tree is constructed by inserting web access sequence and support count. WAP-mine algorithm is applied to the WAP-tree generated at start and also applied to the trees generated in between, to obtain web access patterns. Unlike Apriori based algorithms, it requires only two scans of sequence database. Candidate sequences are not generated at any stage during the mining process. The main problem of this algorithm is the utilization of more memory, as large numbers of intermediate trees are generated during complete process.

Baoyao Zhou *et al*. [1], presented the algorithm called as Conditional Sequence Base mining algorithm (CSB-mine). It does not generate any candidate sequences like Apriori-based algorithms. Also there is no need to build any WAP-tree to store web access sequence. The algorithm starts with preprocessing step to build conditional sequence base. Next, events queues are constructed using this conditional sequence base. In further step, sub-conditional sequence base is constructed recursively to obtain frequent patterns. Like WAP-mine algorithm, it does not generate any costlier intermediate trees. Authors also presented web recommendation system using CSB-mine algorithm. It is shown that CSB-mine outperforms WAP-mine algorithm.

III. Motivation

The existing algorithms such as Apriori based suffer from the major cost such as: database is scanned many times to obtain frequent patterns and specifically for large number of itemsets explosive candidate sequences are generated [4, 8, 9, 10]. Other algorithms based on Pattern-growth either involve the construction of projected databases or construction of number of trees during the complete mining process [4, 9, 15]. The conditional Sequence Base (CSB) algorithm does not construct any costlier tree at any stage, but it generates sub conditional sequence base recursively to obtain frequent patterns [1]. It also includes a process to test if all generated sequences can be merged to form a single sequence. If yes, then the test process will be terminated.

For the evaluation of sequential pattern mining algorithms, run time and memory utilization are two important measures [4]. So we have tried to improve the performance of our algorithm by considering these two factors. Our proposed algorithm avoids recursive mining for sub-CSB and does not use any extra data structure to store first appearance of symbol. It stores only pointer to first appearance of symbol of first WAS list in Unique Symbol list with support count and then consecutive occurrence pointers are stored in WAS set as shown in Fig. 4, to form linked list. This technique saves space and time. The used data structure is compact and also the construction of projected database is avoided. This again saves space and time. At the end, a data structure SAP is generated which holds frequent patterns of all lengths with their support value. SAP can be further useful for developing system like web recommendations or personalization.

IV. Proposed Mining Algorithm

Let I is a set of unique items (item may also be referred as event or symbol). While surfing the web, users access various web resources e.g. it may be any URL, topic or web page. Assume I = {p, q, r, s, t}, then p, q, r, s and t can be any URL, topic or web page [1]. Let WS = $i_1 i_2 i_3 \ldots i_n$, where $i_x \in I$ for $1 \leq x \leq n$. WS is called as web access sequence. The length of WS is denoted by n and it is $|WS|$. Note, i_x and i_y are not necessarily different. It means repetition of items or symbols is allowed [1]. e.g. WS = qpsprs is a web access sequence. Table I shows various web access sequences as a sample database. The table contains five web access sequences i.e. it contains five WAS lists.

TABLE I. Various Web Access Sequences as A Sample Database

Web Access Sequence (WAS)
pqpsprs
tptqrpr
qpuptr
puqprurs
psqs

If web access sequence WS = psqs, then if WSprefix = p then WSsuffix will be sqs. WS can also be denoted as WS=p+sqs = ps+qs = psq+s [1].

Let, any web access sequence database = WSDB = {WS1, WS2, …., WSm}, where m is the size of database and it is $|WSDB|$. Any web access sequence WS is called as sequential access pattern if support of WS is ≥ minimum threshold value called as MinSupport. e.g. if MinSupport = 60% then it is required to find all sequential access patterns supported by at least 60% web access sequences from the sample database [1].

Following are some terms and definitions used in our proposed algorithm:

- InitWAS is initial WAS. It is the set of all web access sequences in the given database, as shown in Table I.
- The WAS generated in one session is considered as one WAS.
- Sessions file: It is a collection of sessions.
- WASLEN: It is a number of WAS in a session file.
- PerSupport : Percentage support.
- WAS Set: A set/collection of all WAS lists. WAS list is also called as Event Queue (EQ)
- US list: Unique Symbol list, also called as Header Table (HT).
- Ui: Unique Symbol i.
- SAPn: Sequential Access Patterns of length n.

The data structures used in the proposed algorithm are as follow:

A. *WAS List Node Structure*

Symbol	Next	Link

- Symbol: Each symbol of a WAS list.
- Next: It points to next appearance of a same symbol in next WAS list of WAS set.
- Link: It points to next symbol in WAS list.

B. Unique Symbol List Node Structure

Symbol	Symbol Count	Next	Link

- Symbol: Unique symbol found in a WAS list.
- Symbol Count: Number of times symbol found in WAS set (count once even if found more than once in a WAS list)
- Next: It points to first appearance of a same symbol in next WAS list.
- Link: It points to next symbol in US list.

The WAS list node structure used in our algorithm, is compact data structure, thus requires less memory. The proposed algorithm consists the steps such as: construction of WAS list, construction of Unique Symbol (US) list, preparing Conditional Suffix table and generation of SAP table.

Initially Web Access Sequence (WAS) set and Unique Symbol (US) list are constructed. The WAS set is constructed from session file, where session file contains number of web access sequences. The US list is constructed by considering each symbol (event) in WAS set whose support count is greater than or equal to the minimum support. So, US list contains only those unique symbols having support count ≥ minimum support. Symbols with count < minimum support are removed from WAS set. Then the conditional suffix table is constructed for each symbol in US list, to link that symbol with WAS set. The use of compact data structure and the technique of linking used, save space and time. All lists like WAS list, US list etc. are maintained as linked lists. Finally, Sequential Access Pattern (SAP) table is generated, which stores the frequent patterns.

The proposed mining algorithm is as follow:

1. Construct WAS set and US list
1.1. Construct WAS set from sessions file
1.1.1. Construct WAS list for each session
1.2. Define MinSupport = (PerSupport / 100) * WASLEN
1.3. Construct US list
1.3.1. For each Ui in WAS set, add Ui to US list if its count is ≥ MinSupport (multiple appearances of a symbol in a WAS is counted as one)
1.4. Remove symbols from WAS set which are not members of US list
1.5. Prepare Conditional Suffix table for each symbol Ui in US list
1.5.1. Link the symbol Ui from US list with first appearance of symbol Ui in WAS set.
1.5.2. Link the first appearance of symbol Ui in WAS list with the first appearance of same symbol Ui in the next WAS list, and continue linking till the end of WAS set.
2. Generate SAP
2.1. Initialize SAP table to US list (SAP_1 = US list)
2.2. Generate n+1 length Sequential Access Patterns
2.2.1. Repeat until SAP_{n+1} are generated with support count less than MinSupport
2.2.1.1. For each sequence SEQ of length 'n' from SAP table generated in previous iterations, do
2.2.1.1.1. For each symbol Ui from US list
2.2.1.1.1.1. Use WAS set to find SEQ + Ui, with support count more than MinSupport
2.2.1.1.1.2. Add $SEQ_{n+1} = SEQ_n + U_i$ to SAP table.

As shown in Table I, InitWAS = {pqpsprs, tptqrpr, qpuptr, puqprurs, psqs } with WASLEN as 5. To be qualified as a conditional frequent symbol (event) with MinSuport = 60% and WASLEN = 5, a symbol must have a count of at least 3 (multiple appearance of a symbol in a WAS is counted as one). Therefore, the conditional frequent symbols are (p: 5), (q: 5), (s: 3) and (r: 4). The frequent access symbol is represented as (symbol: count), and count means number of sequences which contains the given symbol.

After completion of step 1.1, WAS set will be constructed as shown in Fig. 1.

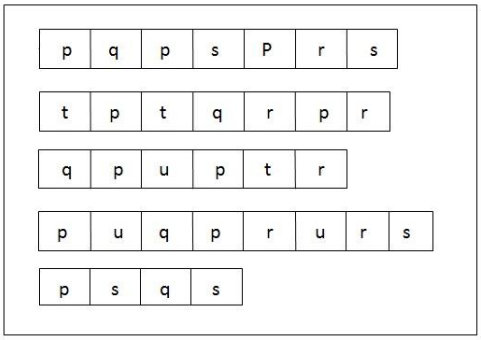

Figure 1. WAS set

After completion of step 1.3, symbols with less than support value 3 (i.e. non frequent symbols t and u) are removed from WAS set and unique symbol list. Therefore, the US list will be as shown in Fig. 2.

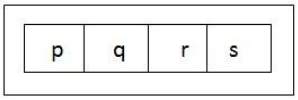

Figure 2. US list

After completion of step 1.4, contents of WAS set will be as shown in Fig. 3.

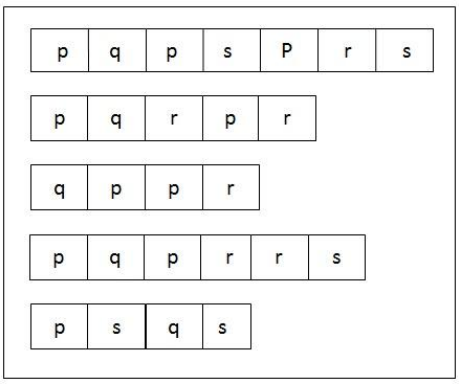

Figure 3. WAS set after removing non frequent symbol t and u

After completion of step 1.5, WAS set and unique symbols list entries will be linked as shown in Fig. 4.

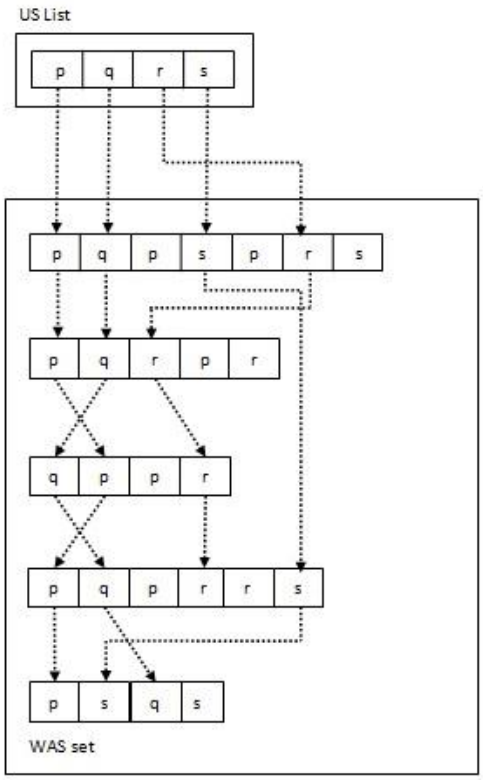

Figure 4. Linking of WAS set and US list.

All suffix sequences of p, q, r and s will be as follow:
For (p): {qpsprs, qrpr, pr, qprrs, sqs}.
For (q): {psprs, rpr, ppr, prrs, s}.
For (r): {s, pr, rs}.
For (s): {prs, qs}.

The proposed method is more efficient as every data structure is maintained as linked list, so there is no need of generating sub conditional sequence bases separately. They are mined directly from linked list. After completion of step 2, the complete sequential access patterns (SAP) with MinSupport of 60% will be as shown in Table II.

TABLE II.

Length of Patterns	Sequential Access Patterns (SAP)
1	p : 5, q : 5, s : 3, r : 4
2	pp :5, pq : 4, ps : 3, pr : 4, qp : 5,qr : 4
3	pqp : 4, pqr : 3, qpr : 4

V. PEFORMANCE EVALUATION OF PROPOSED MINING ALGORITHM

Run time and memory utilization are two important measures for performance evaluation of mining algorithms. In our proposed algorithm, as every data structure is maintained as linked list there is no need of generating sub conditional sequence bases separately. They are mined directly from linked list. Also there is no need of separate single sequence testing algorithm, as generation stops when no n+1 length sequence is generated having count ≥ support value. The mining is done by processing inter connected WAS set and US list. Also the algorithm uses a compact data structure for WAS list node structure and hence there is no need to reconstruct the projected database. All these techniques will save memory space and run time. The algorithm uses different technique to link US list with WAS list. WAS set is a collection of WAS lists. Every US node holds a unique symbol and its count occurred in WAS set. It also points to the first appearance of the same symbol in the next WAS list. First appearance of any symbol in WAS list points to first appearance of same symbol in next WAS list in sequence of WAS set. In short, we store only pointer to first appearance of symbol of first WAS list in Unique Symbol list with support count and then consecutive occurrence pointers are stored in WAS set as shown in Fig. 4, to form linked list. This saves memory space and time.

TABLE III. RESULTS OF PREFIX SPAN MINE ALGORITHM

Support in %	Support	Number of Symbols with MinSupport	Total Number of Sequential Access Patterns generated	Average Memory Usage (in KB)	Run Time (in Min:Sec)
2.0	200	462	533	40319	1:58
1.9	190	475	569	39290	2:09
1.8	180	492	620	40077	2:39
1.7	170	510	692	39445	3:22
1.6	160	526	771	40717	3:40
1.5	150	540	878	39557	4:36
1.4	140	573	1013	39646	6:07
1.3	130	598	1240	39920	8:23
1.2	120	621	1570	40162	11:26
1.1	110	641	1974	40194	16:40

TABLE IV. RESULTS OF CSB-MINE ALGORITHM

Support in %	Support	Number of Symbols with MinSupport	Total Number of Sequential Access Patterns generated	Average Memory Usage (in KB)	Run Time (in Min:Sec)
2.0	200	462	533	20086	1:56
1.9	190	475	569	20091	2:07
1.8	180	492	620	20098	2:33
1.7	170	510	692	20109	3:23
1.6	160	526	771	20117	3:43
1.5	150	540	878	20120	4:36
1.4	140	573	1013	20139	6:01
1.3	130	598	1240	20168	8:21
1.2	120	621	1570	20203	11:27
1.1	110	641	1974	20225	16:46

TABLE V. RESULTS OF PROPOSED MINING ALGORITHM

Support in %	Support	Number of Symbols with MinSupport	Total Number of Sequential Access Patterns generated	Average Memory Usage (in KB)	Run Time (in Min:Sec)
2.0	200	462	533	9386	1:27
1.9	190	475	569	9457	1:41
1.8	180	492	620	9548	2:01
1.7	170	510	692	9630	2:45
1.6	160	526	771	9693	3:25
1.5	150	540	878	9793	4:18
1.4	140	573	1013	9901	5:51
1.3	130	598	1240	10018	8:05
1.2	120	621	1570	10167	10:36
1.1	110	641	1974	10278	14:29

TABLE VI. NUMBER OF SEQUENTIAL ACCESS PATTERNS OF DIFFERENT LENGTHS

Support in %	Support	Length of Sequential Access Patterns							Total Number of Sequential Access Patterns generated
		1	2	3	4	5	6	7	
2.0	200	462	71	-	-	-	-	-	533
1.9	190	475	94	-	-	-	-	-	569
1.8	180	492	128	-	-	-	-	-	620
1.7	170	510	182	-	-	-	-	-	692
1.6	160	526	239	5	1	-	-	-	771
1.5	150	540	330	7	1	-	-	-	878
1.4	140	573	428	11	1	-	-	-	1013
1.3	130	598	598	32	12	-	-	-	1240
1.2	120	621	844	41	36	21	7	-	1570
1.1	110	641	1202	65	37	21	7	1	1974

For evaluation, we have implemented CSB-mine and our proposed mining algorithms in Java. To compare our experimental results, we have used the available standard code of PrefixSpan algorithm. The PrefixSpan algorithm is proposed by Jian Pei *et al.* [9]. These experiments are performed on Pentium Dual core T4200 @ 2.00 Ghz machine with Windows 7.0 Enterprise N operating system. We have used publicly available synthetic data set T25i10D10K to carry out our experiments. It is made available by IBM Quest data mining project. The

correctness of the implementations is confirmed by checking that the frequent patterns generated for the same dataset by all algorithms are the same. Table III shows the results of PrefixSpan mine algorithm, while Table IV gives the results for CSB-mine algorithm and finally Table V shows the results of proposed mining algorithm.

Run time means total execution time. It includes time required to read session file, build data structure and generate final patterns. Memory usage implies the memory required to store all data structures and generated final patterns. Table VI shows the number of sequential access patterns generated, with different lengths and different support.

Fig. 5 shows the memory usage comparison between PrefixSpan, CSB-mine and our proposed mine algorithms. It shows that the proposed mine algorithm requires less memory as compared to other two algorithms for all support values. Fig. 6 shows the run time comparison between all three algorithms and shows 10% to 15% improvement for proposed mine algorithm.

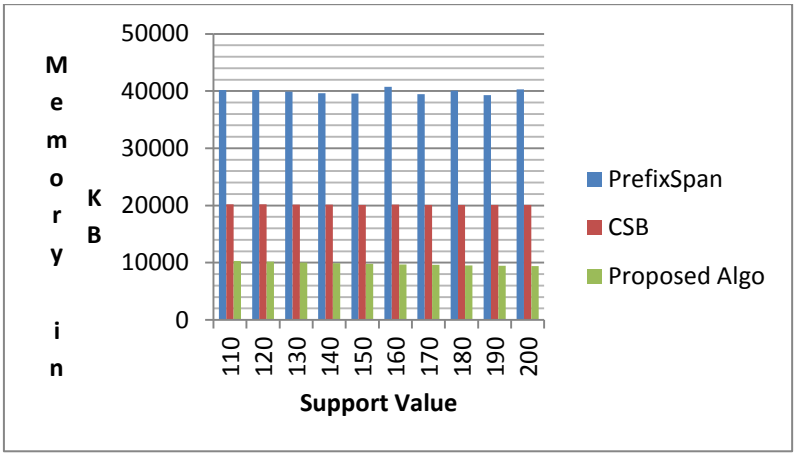

Figure 5. Memory usage comparison between PrefixSpan, CSB-mine and proposed mine algorithms.

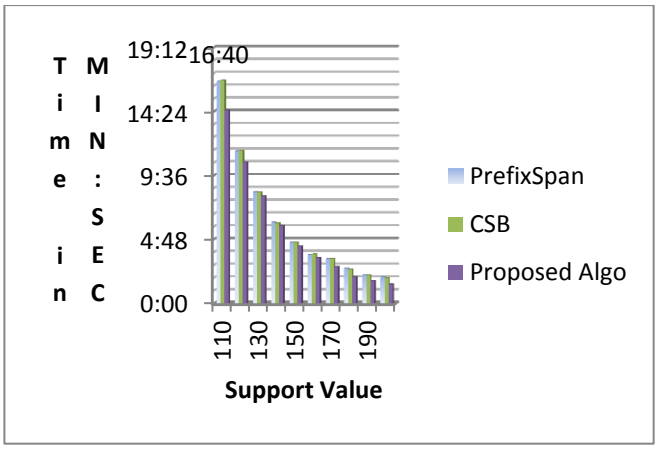

Figure 6. Run time comparison between PrefixSpan, CSB-mine and proposed mine algorithms.

VI. Conclusion

In this paper, we have proposed an efficient algorithm for mining sequential access patterns from web access sequence (WAS) database. The performance of the proposed mining algorithm has been evaluated in comparison with PrefixSpan and CSB-mine algorithms. Our proposed algorithm does not require construction of initial tree or intermediate conditional trees. It eliminates the need for generating sub conditional sequence bases separately, does not require any separate single sequence testing algorithm, avoids to use any separate data structure to store first appearance of symbol and uses a compact data structure and thus avoids the reconstruction of projected database. All these techniques save considerable amount of space and time.

Experimental results have shown that the proposed algorithm outperforms the PrefixSpan and CSB-mine algorithms. The results show significant improvement in average memory usage and 10% to 15% improvement in the run time.

References

[1] Y. Z. Bao, C. H. Siu, and A. C. Ming Fong, "Efficient sequential access pattern mining for web recommendations," *International*

Journal of Knowledge Based and Intelligent Engineering Systems, ACM, vol. 10, no. 2, pp. 155-168, April 2006.

[2] J. Srivastava, R. Cooley, M. Deshpande, and P. N. Tan, "Web usage mining: Discovery and applications of usage patterns from web data," *SIGKDD Explorations, ACM SIGKDD*, vol. 1, no. 2, pp. 12-23, Jan. 2000.

[3] K. R. Suneetha and R. Krishnamoorthi, "Identifying user behavior by analyzing web server access log file," *IJCSNS International Journal of Computer Science and Network Security*, vol. 9, no. 4, April 2009.

[4] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Computing Surveys*, vol. 43, no. 1, pp. 3:1-3:41, November 2010.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th VLDB Conference*, 1994, pp. 487-499.

[6] U. Niranjan**,** R. B. V. Subramanyam, and V. Khanaa, "An efficient system based on closed sequential patterns for web recommendations," *IJCSI International Journal of Computer Science Issues*, vol. 7, no. 3, no. 4, pp. 26-34, May 2010.

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

[8] K. C. Srikantaiah, N. K. Krishna, K. R. Venugopal, and L. M. Patnaik, "Bidirectional growth based mining and cyclic behaviour analysis of web sequential patterns," *International Journal of Data Mining & Knowledge Management Process*, vol. 3, no. 2, March 2013.

[9] J. Pei, J. W. Han, B. Mortazavi-Asi, H. Pinto, Q. M. Chen, U. Dayal, and M. C. Hsu, "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. 2001, International Conference Data Engineering*, April 2001, pp. 215-224.

[10] H. Wang, C. Yang, and H. Zeng, "Design and implementation of a web usage mining model based on fpgrowth and prefixspan," *Communications of the IIMA*, vol. 6, no. 2, pp. 71-86, 2006.

[11] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th International Conference on Extending Database Technology: Advances in Database Technology, Lecture Notes in Computer Scienc*e, vol. 1057, Springer, 1996, pp. 3-17.

[12] J. W. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "Free span: Frequent pattern-projected sequential pattern mining," in *Proc. 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM*, 2000, pp. 355-359.

[13] J. Pei, J. W. Han, B. Mortazavi-Asl, J. Y. Wang, H. Pinto, Q. M. Chen, U. Dayal, and M. C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424-1440, November 2004.

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

[15] J. Pei, J. W. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," *Knowledge Discovery and Data Mining, Current Issues and New Applications, Lecture Notes in Computer Science*, vol. 1805, Springer, 2000, pp. 396-407.

Krishnakant P. Adhiya received the B.E. degree in Computer Engineering from Amravati University, India in 1990, the M.E. degree in Computer Science and Engineering from Allahabad University, India in 1996. He is currently working as Associate Professor in department of Computer Engineering at SSBT's College of Engineering and Technology, Bambhori, Jalgaon, India. He has teaching experience of 23 years. His research interest lies in the field of Data Mining.

Satish R. Kolhe received the B.E. degree in Computer Engineering from Amravati University, India, in 1991, the M.Tech. degree in Engineering Systems from Dayalbagh Educational Institute, Agra, India, in1994, and the Ph.D. degree in Computer Engineering from North Maharashtra University, Jalgaon, India, in 2007. He is currently working as a Professor with the School of Computer Sciences, North Maharashtra University. His research interests include artificial intelligence and pattern recognition. Prof. Kolhe is a Fellow Life Member of Institute of Electronics and Telecommunication Engineers, India, Life Member of Computer Society of India, India, Linguistic Society of India, and Indian Science Congress Association. He is also a member of Special Interest Group in Artificial Intelligence, India