Modified Web Access Pattern (mWAP) Approach for Sequential Pattern Mining

Jatin D Parmar[1]
Sanjay Garg[2]

[1]Shri S'ad Vidya Mandal Institute of Technology
Bharuch, Gujarat - India
jatin_d_parmar@yahoo.co.in

[2]A.D Patel Institute of Technology
Vallabh Vidyanagar, Gujarat - India
gargsv_adit@yahoo.com

Abstract. With the explosive growth of data available on the World Wide Web, discovery and analysis of useful information from the World Wide Web becomes a practical necessity. Web access pattern, which is the sequence of accesses pursued by users frequently, is a kind of interesting and useful knowledge in practice. Sequential Pattern mining is the process of applying data mining techniques to a sequential database for the purposes of discovering the correlation relationships that exist among an ordered list of events. Web access pattern tree (WAP-tree) mining is a sequential pattern mining technique for web log access sequences, which first stores the original web access sequence database on a prefix tree, similar to the frequent pattern tree (FP-tree) for storing non-sequential data. WAP-tree algorithm then, mines the frequent sequences from the WAP-tree by recursively re-constructing intermediate trees, starting with suffix sequences and ending with prefix sequences. An attempt has been made to modify WAP tree approach for improving efficiency. mWAP totally eliminates the need to engage in numerous re-construction of intermediate WAP-trees during mining and considerably reduces execution time.

Keywords: WAP tree, data mining, sequential data mining, frequent pattern tree

(Received July 01, 2006 / Accepted January 03, 2007)

1 Introduction

Data Mining is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. With the wide spread use of databases and the explosive growth in their sizes, organization are faced with the problem of information overload. The problem of effectively utilizing these massive volumes of data is becoming a major problem for all enterprises. Traditionally, we have been using data for querying a reliable databases repository via some well-circumscribed application for canned report-generating utility. While this mode of interaction is satisfactory for a large class of applications, there exist many other applications which demand exploratory data analyses. These applications support query-triggered usage of data, in the sense that the analysis is based on a query posed by a human analyst. On the other hand, data mining techniques support automatic exploration of data. Data mining attempts to source out patterns and trends in the data and infers rules from these patterns. With these rules the user will be able to support, review and examine decisions in some related business or scientific area. This opens up the possibility of a new way of interacting with databases and data warehouses.

Sequential mining is the process of applying data mining techniques to a sequential database for the purposes of discovering the correlation relationships that exist among an ordered list of events.

The objective of this work is to apply data mining techniques to a sequential database for the purposes of discovering the correlation relationships that exist among an ordered list of events. Given a WASD (Web Access Sequence Database) , the problem to find frequently occurring Sequential patterns on the basis of minimum support provided. The application of sequential pattern mining are in areas like Medical treatment, science & engineering processes, telephone calling patterns. Sequential pattern mining Web usage mining for automatic discovery of user access patterns from web servers. It is used by an e-commerce company, this means detecting future customers likely to make a large number of purchases, or predicting which online visitors will click on what commercials or banners based on observation of prior visitors who have behaved either positively or negatively to the

advertisement banners.

2. Background

Sequential Pattern Mining comes in Association rule mining. For a given transaction database T, an association rule is an expression of the form $X \rightarrow Y$, where X and Y are subsets of A and $X \rightarrow Y$ holds with confidence τ, if τ% of transactions in D that support X also Y. The rule $X \rightarrow Y$ has support σ in the transaction set T if σ% of transactions in T support X U Y. Association rule mining can be divided into two steps. Firstly, frequent patterns with respect to support threshold min sup are mined. Secondly association rules are generated with respect to confidence threshold minimum confidence. Pattern Mining is of two types:

[1]**Non Sequential Pattern Mining:** The items occurring in one transaction have no order.

[2] **Sequential Pattern Mining:** The items occurring in one transaction have an order between the items (events) and an item may re-occur in the same sequence.

WAP-tree, which stands for web access pattern tree. The main steps involved in this technique are summarized next. The WAP-tree stores the web log data in a prefix tree format similar to the frequent pattern tree (FP-tree) for non-sequential data. The algorithm first scans the web log once to find all frequent individual events. Secondly, it scans the web log again to construct a WAP-tree over the set of frequent individual events of each transaction. Thirdly, it finds the conditional suffix patterns. In the fourth step, it constructs the intermediate conditional WAP-tree using the pattern found in previous step. Finally, it goes back to repeat Steps 3 and 4 until the constructed conditional WAP-tree has only one branch or is empty.

TID	Web access sequence	Frequent Subsequence
100	pqspr	pqpr
200	tptqrp	pqrp
300	opqupt	qpqp
400	puqprur	pqprr

Table 1. Sequence database for WAP-tree

Thus, with the WAP-tree algorithm, finding all frequent events in the web log entails constructing the WAP-tree and mining the access patterns from the WAP tree. The web log access sequence database in Table 1 is used to show how to construct the WAP-tree and do WAP-tree mining. Suppose the minimum support threshold is set at 75%, which means an access sequence, *s* should have a count of 3 out of 4 records in our example, to be considered frequent. Constructing WAP-tree, entails first scanning database once, to obtain events that are frequent. When constructing the WAP-tree, the non-frequent part of every sequence is discarded. Only the frequent sub-sequences are used as input. For example, in Table 1, the list of all events is p, q, r, s, t, u and the support of p is 4, q is 4, r is 3, s is 1, t is 2, and u is 2.With the minimum support of 3, only p, q, r are frequent events. Thus, all non-frequent events (like *s*, *t*, *u*) are deleted from each transaction sequence to obtain the frequent subsequence shown in column 3 of Table 1.

With the frequent sequence in each transaction, the WAP-tree algorithm first stores the frequent items as header nodes so that these header nodes will be used to link all nodes of their type in the WAP-tree in the order the nodes are inserted. When constructing the WAP tree, a virtual root (Root) is first inserted. Then, each frequent sequence in the transaction is used to construct a branch from the Root to a leaf node of the tree. Each event in a sequence is inserted as a node with count 1 from Root if that node type does not yet exist, but the count of the node is increased by 1 if the node type already exists. Also, the head link for the inserted event is connected (in broken lines) to the newly inserted node from the last node of its type that was inserted or from the header node of its type if it is the very first node of that event type inserted. For example, as shown in figure 1(a), to insert the first frequent sequence pqpr of transaction ID 100 of the example database, since there is no node labeled p yet, which is a direct child of the Root, a left child of Root is created, with label p and count 1. Then, the header link node for frequent event p is connected (in broken lines) to this inserted *a* node from the p header node. The next event q is inserted as the left child of node p with a count of 1 and linked to header node q, the third event p is inserted as the left child of the node q having a count of 1, and the p link is connected to this node from the inserted p. The fourth and last event of this sequence is r and it is inserted as the left child of the second p on this branch with a count of 1 and a connection to r header node. Secondly, insert the sequence pqrp of the next transaction with ID 200, starting from the virtual Root (figure 1(b)). Since the root has a child labeled p, the node p's count is increased by 1 to obtain (p: 2). similarly, (q: 2) is also in the tree. The next event, r, does not match the next existing node p, and new node r:1 is created and

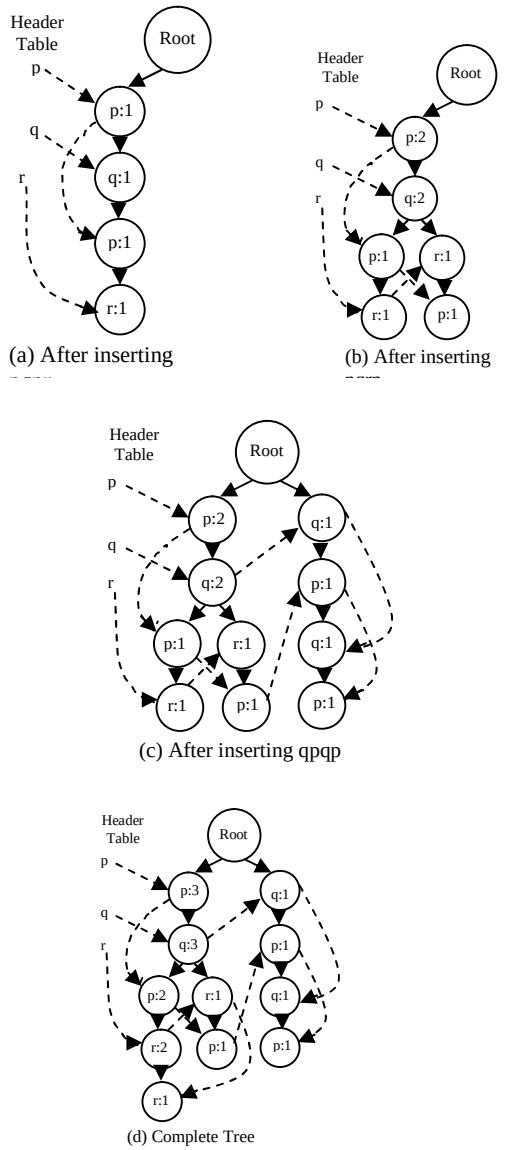

Figure 1. Construction of WAP Tree

Inserted as another child of q node. The third sequence qpqp of ID 300 and the fourth sequence pqprr are inserted next to obtain figure 1(c) and (d) respectively.

Once the sequential data is stored on the complete WAP-tree (figure 1(d)), the tree is mined for frequent patterns starting with the lowest frequent event in the header list, in our example, starting from frequent event r as the following discussion shows. From the WAP-tree of figure 1(d), it first computes prefix sequence of the base r or the conditional sequence base of *c as*: pqp:2; pq:1; pqpr:1; pqp:-1. The conditional sequence list of a suffix event is obtained by following the header link of the event and reading the path from the root to each node (excluding the node). The count for each conditional base path is the same as the count on the suffix node itself. The first sequence in the list above, pqp represents the path to the first r node in the WAP tree. When a conditional sequence in a branch of a WAP-tree, has a prefix subsequence that is also a conditional sequence of a node of the same base, the count of this new subsequence is subtracted because it has contributed before. Thus, the conditional sequence list above pqp with counts of -1. This is because, when the subsequence, pqpr is added to the list, its subsequence pqp was already in the list. Thus, the count of pqp with -1 has to be added to prevent it from contributing twice. To qualify as a conditional frequent event, one event must have a count of 3. Therefore, after counting the events in sequences above, the conditional frequent events are p(4) and q(4) and r with a count of 1, which is less than the minimum support, is discarded. After discarding the non-frequent part r in the above sequences, the conditional sequences based on r are listed as: pqp:2; pq:1; pqp:1; pqp:-1.

Using these conditional sequences, a conditional WAP tree, WAP-tree|r, is built using the same method as shown in figure 1. The new conditional WAP-tree is shown in figure 2(a). Recursively, based on the WAP-tree in figure 2(a), the next conditional sequence base for the next suffix subsequence, qr is found as p(3). With p as the only frequent pattern in this base, the frequent sequence base of qr used to construct the next WAP tree shown in figure 2(b) is p(3). This ends the re-construction of WAP trees that progressed as suffix sequences |r, |qr and the frequent patterns found along this line are r, qr and pqr. The recursion continues with the suffix path |r, |pr. Thus, the conditional sequence base for suffix pr is computed from figure 2(a) as Ø, pq:3. This list is used to construct the WAP tree of figure 2(c). The algorithm keeps running, finding the conditional sequence bases of qpr as p: 3. from the list, the conditional frequent events of pqr is only p: 3. Then, the conditional WAP-tree|qpr is built as shown in figure 2(d). Now back to completing the mining of frequent patterns with suffix pr, figure 2(c) is mined for conditional sequence bases for suffix ppr and we get NULL.

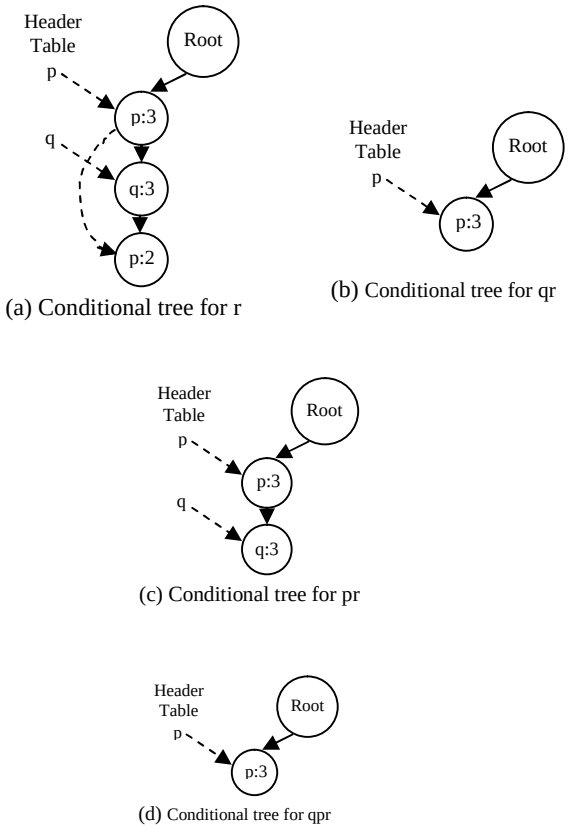

Figure 2. Reconstruction of WAP trees for mining conditional pattern base r.

The conditional search of r is now finished. The search for frequent patterns that have the suffix of other header frequent events (starting with suffix base |q and then |p) are also mined the same way the mining for patterns with suffix r is done above. After mining the whole tree, discovered frequent pattern set is: {r, qpr, pqpr, pr, pqr, qr, qb, pq, p, pp, qp, pqp}.

WAP-tree algorithm scans the original database only twice and avoids the problem of generating explosive candidate sets as in Apriori-like algorithms. Mining efficiency is improved sharply, but the main drawback of WAP-tree mining is that it recursively constructs large numbers of intermediate WAP-trees during mining and this entails storing intermediate patterns, which are still time consuming operations.

Pre-Order linked WAP [14] tree algorithm is a version of the WAP tree algorithm that assigns unique position code to each tree node and performs the header node linkages pre-order fashion. Both the pre-order linkage and binary position codes enable the PLWAP to directly mine the sequential patterns from the one initial WAP tree starting with prefix sequence, without re-constructing the intermediate WAP trees.

3 Related Work

Sequential mining was proposed, using the main idea of association rule mining presented in Apriori algorithm of Agrawal and Srikant [2]. Later work on mining sequential patterns in web log include the GSP[2], the PSP[12], the *G* sequence and the graph traversal[11] algorithms. Agrawal and Srikant proposed three algorithms (Apriori, AprioriAll, AprioriSome) to handle sequential mining problem. Following this, the GSP (Generalized Sequential Patterns) [2] algorithm, which is 20 times faster than the Apriori algorithm in Agrawal and Srikant[1] was proposed. The GSP Algorithm makes multiple passes over data. The first pass determines the frequent 1-item patterns (*L*1). Each subsequent pass starts with a seed set: the frequent sequences found in the previous pass (*Lk*-1). The seed set is used to generate new potentially frequent sequences, called candidate sequences (*Ck*). Each candidate sequence has one more item than a seed sequence. In order to obtain *k*-sequence candidate *Ck*, the frequent sequence *Lk*-1 joins with itself Apriori-gen way. The GSP algorithm uses a hash tree to reduce the number of candidates that are checked for support in the database.

The PSP [12] approach is much similar to the GSP algorithm[2]. At each step *k*, the database is browsed for counting the support of current candidates. Then, the frequent sequence set, *Lk* is built. The only difference between the PSP algorithm and the GSP is that it introduces the prefix-tree to handle the procedure. Any branch, from the root to a leaf stands for a candidate sequence, and a terminal node provides the support of the sequence from the root to the considered leaf inclusive.

The main idea of Graph Traversal mining which is proposed by Nanopoulos and Manolopoulos[11], is using a simple unweighted graph to reflect the relationship between the pages of web sites. Then, a graph traversal algorithm similar to Apriori algorithm, is used to traverse the graph in order to compute the *k*-candidate set from the (*k* - 1)-candidate sequences without performing the apriori-gen join. From the graph, if a candidate node is large, the adjacency list of the node is retrieved. The database still has to be scanned several times to compute the support of each candidate sequence although the number of computed candidate sequences is drastically reduced from that of the GSP algorithm.

The FP-tree structure [6] first reorders and stores the frequent non sequential database transaction items on a prefix tree, in descending order of their supports such that database transactions share common frequent prefix paths on the tree. Then, mining the tree is accomplished by recursive construction of conditional pattern bases for each frequent 1-item (in ordered list called *f* -list), starting with the lowest in the tree. Conditional FP-tree is constructed for each frequent conditional pattern having more than one path, while maximal mined frequent patterns consist of a concatenation of items on each single path with their suffix *f* -list item. FreeSpan like the FP-tree method, lists the *f* -list in descending order of support, but it is developed for sequential pattern mining. FreeSpan mines frequent sequential patterns starting with each of its *f* –list items *α*, through recursive construction of projected databases of this *f* -list item *α*. A projected database of an ordered *f* -list item *α* from the database *D*, consists of all sequences in *D* containing this *f* -list item *α* but removing all items after *α* in the ordered *f* -list. PrefixSpan [8] is a pattern-growth method like FreeSpan, which reduces the search space for extending already discovered prefix pattern *p* by projecting a portion of the original database that contains all necessary data for mining sequential patterns grown from *p*. While FreeSpan supports frequent pattern guided projection, PrefixSpan supports prefix guided projection. Thus, projected database for each *f* -list prefix pattern *α* consists of all sequences in the original database *D*, containing the pattern *α* and only the subsequences prefixed with the first occurrence of *α* are included. Although PrefixSpan projects smaller sized databases than FreeSpan, they both still incur non-trivial costs for constructing and storing these projected databases for every sequential pattern in the worst case. Optimization techniques include (1) bi-level projecting for reducing the number and sizes of projected databases, and (2) Pseudo-projection for projecting memory-only databases, where each projection consists of the pointer to the sequence and offset of the postfix to the sequence.

3 The Modified Web Access Pattern tree Approach

The modified Web Access Pattern approach is based on WAP-tree, but avoids recursively re-constructing intermediate WAP-trees during mining of the original WAP tree for frequent patterns. The modified WAP algorithm is able to quickly determine the suffix of any frequent pattern prefix under consideration by comparing the assigned binary position codes of nodes of the tree.

A tree is a data structure accessed starting at its root node and each node of a tree is either a leaf or an interior node. A leaf is an item with no child. An interior node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Like WAP-tree mining, every frequent sequence in the database can be represented on a branch of a tree. Thus, from the root to any node in the tree defines a frequent sequence. For any node labeled e in the WAP-tree, all nodes in the path from root of the tree to this node (itself excluded) form a prefix sequence of e. The count of this node e is called the count of the prefix sequence. Any node in the prefix sequence of e is an ancestor of e. On the other hand, the nodes from e (itself excluded) to leaves form the suffix sequences of e.

Given a WAP-tree with some nodes, the binary code of each node can simply be assigned following the rule that the root has null position code, and the leftmost child of the root has a code of 1, but the code of any other node is derived by appending 1 to the position code of its parent, if this node is the leftmost child, or appending 10 to the position code of the parent if this node is the second leftmost child, the third leftmost child has 100 appended, etc. In general, for the nth leftmost child, the position code is obtained by appending the binary number for 2n-1 to the parent's code. A node α is an ancestor of another node β if and only if the position code of α with "1" appended to its end, equals the first x number of bits in the position code of β, where x is the ((number of bits in the position code of α) + 1).

The tree data structure, similar to WAP-tree, is used to store access sequences in the database, and the corresponding counts of frequent events compactly, so that the tedious support counting is avoided during mining. A Binary code is assigned to each node in modified WAP-tree. These codes are used during mining for identifying the position of the nodes in the tree. The header table is constructed by linking the nodes in sequential events fashion. Here the linking is used to keep track of nodes with the same label for traversing prefix sequences. This mining algorithm is prefix sequence search rather than suffix search.

The algorithm scans the access sequence database first time to obtain the support of all events in the event

3.1 The Algorithm

Input : Access sequence database D(i), min support MS ($0 < MS \leq 1$)
Output : frequent sequential patterns in D(i).
Variables : Cn stores total number of events in suffix trees, A stores whether a node is ancestor in queue.

```
Begin
1.    Scan D(i) to  discover frequent individual events L;
2.    Scan D(i) again .Create a root node of Tree T.
3.     code(root)= NULL;
4.     count = 0;
5.       {
6.        For ( each access sequence, fs in D(i))
7.        {
8.        Extract frequent subsequence F=(fs1fs2 . . . fsn) by removing all events
9.        that are not in L;
10.      current node -> leftmost_Child(root);
11.        for ( k=1 to n  )
12.          {
13.          if (current node = NULL)
14.           {Create a new child node with position code equal to "1" appended
                 To position code of parent of current node ;}
15.          elseif (current node = fsk ) { NdFd =  true ;}
16.          else { make current node point to current node sibling}
17.          }
18.       if (NdFd = true)
19.          {count (fsk) ++;
20.          Make current node point to fsk ;}
21.          Else {create new child node with position code of current node with
22.          "0" appended at the end;
23.          Make current node point to new created node ;}
24.           }
25.        }
26.      From root node, do a sequential Traversal of Tree T to make appropriate linkage queue;
27.      PATTERN_DIS (Suffix tree roots STR, Frequent sequence FS);
28.      end;

1.       PATTERN_DIS(R, F)
2.       {
3.       If (STR=empty) return;
4.       for (each suffix tree of event in L)
5.            {
6.            Save first event in ei  queue to A;
7.            if (event ei is descendent of any event in STR, and is not descendent of A)
9.            {Insert ei suffix tree header set STR';
10              Add count of ei to Cn;
11.           Replace the A with ei. ; }
12.           If(Cn > MS )
13.           {Append ei after FS to FS';
14.            print (FS');
15.             PATTERN_DIS (STR',FS');
16.       }
```

set, *E*. All events that have a support greater than or equal to the minimum support are frequent. Each node in a modified tree registers three pieces of information: node label, node count and node code, denoted as label: count: position. The root of the tree is a special virtual node with an empty label and count 0. Every other node is labeled by an event in the event set *E*. Then it scans the database a second time to obtain the frequent sequences in each transaction. The non-frequent events in each sequence are deleted from the sequence. This algorithm also builds a prefix tree data structure by inserting the frequent sequence of each transaction in the tree the same way the WAP-tree algorithm would insert them. Once the frequent sequence of the last database transaction is inserted in the tree, the tree is traversed to build the frequent header node linkages. All the nodes in the tree with the same label are linked by shared-label linkages into a queue. Then, the algorithm recursively mines the tree using prefix conditional sequence search to find all web frequent access patterns. Starting with an event, *ei* on the header list, it finds the next prefix frequent event to be appended to an already computed *m*-sequence frequent subsequence, which confirms an *en* node in the root set of *ei* , frequent only if the count of all current suffix trees of *en* is frequent. It continues the search for each next prefix event along the path, using subsequent suffix trees of some *en* (a frequent 1-event in the header table), until there are no more suffix trees to search. To mine the tree, the algorithm starts with an empty list of already discovered frequent patterns and the list of frequent events in the head linkage table. Then, for each event, *ei*, in the head table, it follows its linkage to first mine 1-sequences, which are recursively extended until the *m*-sequences are discovered. The algorithm finds the next tree node, *en*; to be appended to the last discovered sequence, by counting the support of *en* in the current suffix tree of *ei* (header linkage event). Note that *ei* and *en* could be the same events. The mining process would start with an *ei* event and given the tree, it first mines the first event in the frequent pattern by obtaining the sum of the counts of the first *en* nodes in the suffix subtrees of the Root. This event is confirmed frequent if this count is greater than or equal to minimum support. To find frequent 2-sequences that start with this event , the next suffix trees of *ei* are mined in turn to possibly obtain frequent 2-sequences respectively if support thresholds are met. Frequent 3-sequences are computed using frequent 2-sequences and the appropriate suffix subtrees. All frequent events in the header list are searched for, in each round of mining in each suffix tree set. Once the mining of the suffix subtrees near the leaves of the tree are completed, it recursively backtracks to the suffix trees towards the root of the tree until the mining of all suffix trees of all patterns starting with all elements in the header link table are completed.

3.2 Complexity

Suppose the number of frequent sequential patterns that can be extracted from the WAP-tree is *p* and the number of 1-sequences is *f*, the number of constructed intermediate WAP-trees is *p- f*. The time complexity for constructing tree is the same as that of WAP tree and is $O(nl)$, where *n* is the number of sequences in the WASD and *l* is the length of the longest frequent sequence in the WASD. The time complexity for mining mWAP tree is O (f *p*), where *f* is the number of frequent 1-events and *f* is the total number of frequent patterns to be discovered. This means that for WAP algorithm, this time complexity is multiplied by (*p-f)* times needed for constructing intermediate trees. From the memory view, it can be seen that WAP-tree needs more memory space for intermediate trees and conditional databases. For every node in the WAP-tree, we need 1 byte to store the label, 1 byte to store the count, 2 bytes to store the links which indicate its child and its sibling. The deeper the recursive round the WAP-tree algorithm calls, the more space it needs. In the worst case, it may need the space of the original WAP-tree multiplied several times. When the original WAP-tree is large and the sequence we try to find is long, WAP-tree may face the problem of not being able to store all information in the main memory.

4 Experimental Results

This experiment uses fixed size database and different minimum support .The datasets and algorithms are tested with minimum supports between 0.8% and 10% against the 60 thousand (60 K) database.

From Table 2 and figure 7, it can be seen that

	time in secs at different supports				
Algorithms	2	3	4	5	10
WAP	750	510	330	280	150
Modified WAP	230	160	110	95	48

Table 2. Execution times for dataset at different minimum supports.

the execution time of every algorithm decreases as the minimum support increases. This is because when the minimum support increases, the number of candidate sequence decreases. Thus, the algorithms need less time to find the frequent sequences. The modified WAP algorithm always uses less runtime than the WAP algorithm. WAP tree mining incurs higher storage cost (memory or I/O). Even in memory only systems, the cost of storing intermediated trees adds appreciably to the overall execution time of the program. It is however, more realistic to assume that such techniques are run in regular systems available in many environments, which are not memory only, but could be multiple processor systems sharing memories and CPU's with virtual memory support. As the minimum support threshold decreases, the number of events that meet minimum support increases. This means that WAP-tree becomes larger and longer, and the algorithm needs much more I/O work during mining of WAP tree. As minimum support decreases, the execution time difference between WAP-tree and modified WAP increases.

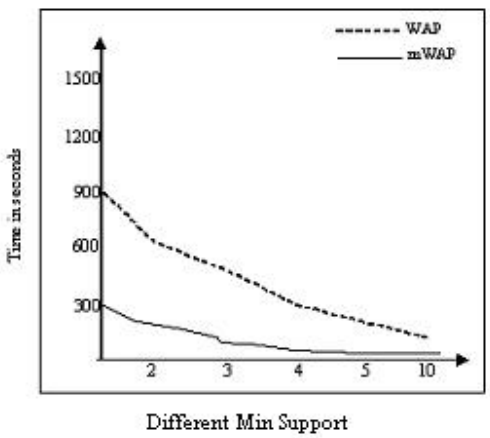

Figure 3. Execution times trend with different minimum supports.

Now, databases with different sizes from 20 K to 100 K with the fixed minimum support of 7% are used.

	Different changed transaction size				
Algorithms time in sec	20k	40k	60k	80k	100k
WAP	148	265	320	445	540
Modified WAP	50	75	97	145	179

Table 3. Execution times trend with different data sizes.

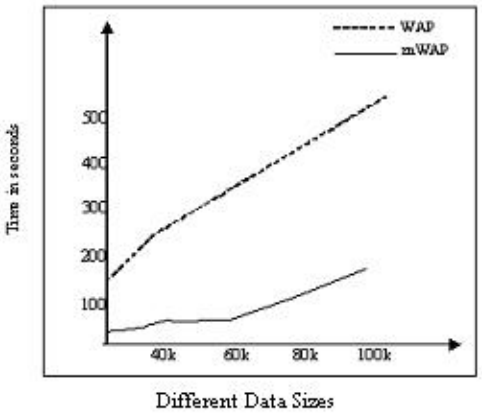

Figure 4. Execution times trend with different data sizes.

5 Conclusion

In this paper, we analyze the problem of sequential pattern mining. Here after discussing the two approached it is clear that the modified version is more efficient than the web access pattern tree approach. This presents a discussion of the advantages and disadvantages of both approaches conduced by comparing the performance with help of graph.

The modified algorithm eliminates the need to store numerous intermediate WAP trees during mining. Since only the original tree is stored, it drastically cuts off huge memory access costs, which may include disk I/O cost in a virtual memory environment, especially when mining very long sequences with millions of records. This algorithm also eliminates the need to store and scan intermediate conditional pattern bases for re-constructing intermediate WAP trees. This algorithm uses the pre-order linking of header nodes to store all events *ei* in the same suffix tree

closely together in the linkage, making the search process more efficient. A simple technique for assigning position codes to nodes of any tree has also emerged, which can be used to decide the relationship between tree nodes without repetitive traversals.

References

[1] Agrawal, R. and Srikant, R. Mining sequential patterns. In Proc. 1995 Int. Conf. Data(ICDE'95), p.3–14, March 1995.

[2] Agrawal, R. and Srikant, R.,. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on very Large Databases Santiago, Chile, p.487–499, 1994.

[3] A. Nanopoulos and Y. Manolopoulos. Mining patterns from graph traversals. Data and Knowledge Engineering, 37(3):243–266, 2001.

[4] Etzioni, O. The world wide web: Quagmire or gold mine. Communications of the ACM, p.65 – 68, 1996.

[5] Han, J., Pei, J. et al. FreeSpan: Frequent pattern projected sequential pattern mining. In SIGKDD, p.355–359, Aug. 2000.

[6] Han, J., Pei, J., Yin, Y. and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. International Journal of Data Mining and Knowledge Discovery, p.53–87, Jan 2004.

[7] Srivastava, J., Cooley, R., Deshpande, M. and Tan, P. Web usage mining: Discovery and applications of usage patterns from web data. SIGKDD Explorations, 2000.

[8] Han, J., Pei, J., Mortazavi-Asl, B. and Pinto, H. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings of the 001 International Conference on Data Engineering (ICDE 01), p.214–224, 2001.

[9] Han, J., Pei, J., Mortazavi-Asl, B. and Zhu, H. Mining access patterns efficiently from web logs. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00) Kyoto Japan, 2000.
Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu

[10] Han, J., Pei, J., Mortazavi-Asl, B., and Pinto, H. 2001. PrefixSpan: Mining sequential patterns efficiently by prefixprojected pattern growth. In Proceedings of the 2001 International Conference on Data Engineering (ICDE'01). Germany, Heidelberg, p. 215–224.

[11] Pujari, A. : Data Mining Techniques , Universities Press, India, February 2001.

[12] Masseglia, F., Poncelet, P. and Cicchetti, R. An efficient algorithm for web usage mining. Networking and Information Systems Journal (NIS), p.571–603, 1999.

[13] Zaki, M. SPADE: An efficient algorithm for mining frequent sequences. Machine Learning, p.31–60, 2001.

[14] Ezeife, C. and Lu, Y. Mining web log sequential patterns with position coded pre-order linked wap-tree. International Journal of Data Mining and Knowledge Discovery (DMKD) Kluwer Publishers, p.5–38, 2005.