BC-WASPT : Web Acess Sequential Pattern Tree Mining

D.Vasumathi [1] **A .Govardhan** [2]
1 Assoc. Professor, Dept.Of CSE, JNTUCEH, JNTUniversity, Hyderabad-85.
2 Professor & HOD, Dept.of CSE, JNTUCEH, JNTUniversity, Hyderabad-85.

Abstract
In this paper, we explore a new sequence pattern technique called BC-WAPT (Binary coded Web Access pattern Tree).it eliminates recursive reconstruction of intermediate WAP tree during the mining by assigning the binary codes to each node in the WAPTree. 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. BCWAPT totally eliminates the need to engage in numerous reconstruction of intermediate WAP-trees during mining and considerably reduces execution time.

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

1. Introduction

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.

Manuscript received June 5, 2009
Manuscript revised June 20, 2009

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 → Y, where X and Y are subsets of A and X → Y holds with confidence t, if t % of transactions in D that support X also Y. The rule X → Y has support δ in the transaction set T if t% 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.

Table 1. Sequence database for WAP-tree

TID	Web Access Sequence	Frequent Subsequence
100	Pqspr	Pqpr
200	Tptqrp	Pqrp
300	Opqupt	Qpqp
400	Puqrur	Pqprr

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 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.

3 Related Works

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*).

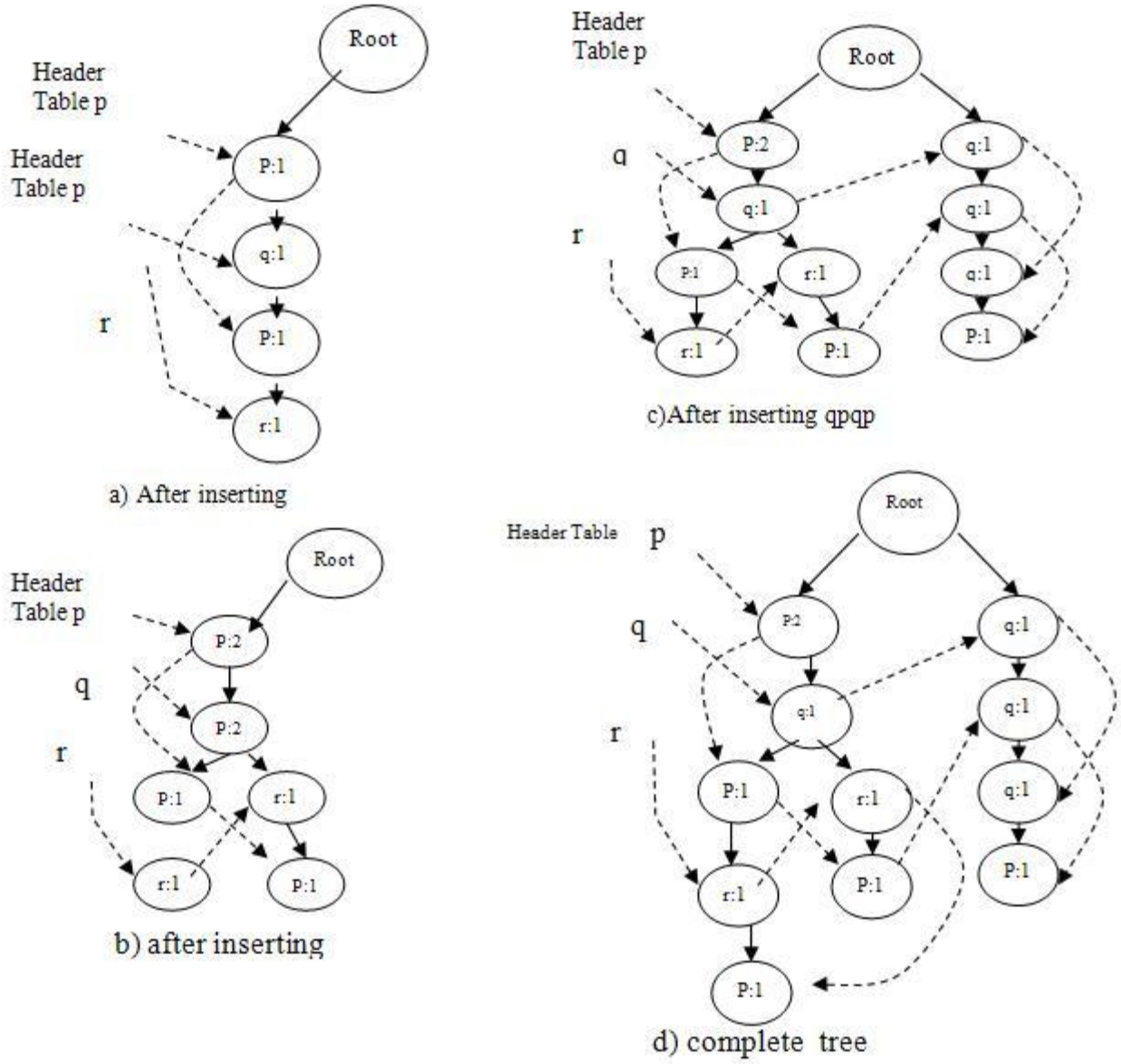


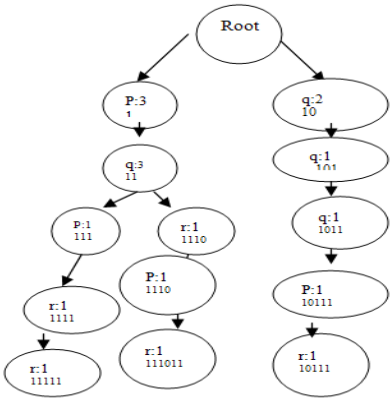
Figure1. Construction of WAP tree

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

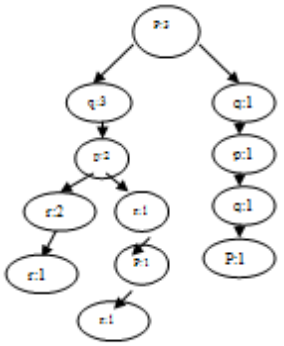
number of computed candidate sequences is drastically reduced from that of the GSP algorithm.

3.1 Tree binary position code assignment algorithm

Position code is a binary code used to indicate the position of the nodes in the WAP-tree.In data structure , when implementing a general tree data structure, a tree is usually transformed into its equivalent binary tree, which has a fixed number of child nodes. To convert a given general tree, T , with nodes at n levels, and root at level 0, the leaf nodes at level $(n - 1)$, to a binary tree, the following rule is applied. The root of the binary tree is the leftmost child of the root of the general tree, T . Then, starting from level 1 of the general tree and working down to level $n - 1$ of the tree, for every node: (1) the leftmost child of this node in the general tree is the left child of the node in the binary tree, and (2) the immediate right sibling of this node in the general tree is the right child of this node in the binary tree. For example, given a tree shown as figure 2, it can be transformed into its binary tree equivalent shown in figure 2(b), where every node has at most two links, one is its left child, and the other is its sibling.



a)Binary coded WAPTree



b) Complete WAPTree

Fig 2: Position code assignment with node position in its binary tree

The position code is assigned to the nodes on the binary tree equivalent of the tree using the Huffman coding idea. Here, the code assignment rule, starts from the leftmost child of the root node of the general tree, which has a binary position code of 1 because this node is the root of the binary tree equivalent of the tree. Thus, given the binary tree equivalent of a tree, with root node having a code of 1, the single temporary position code assignment rule assigns 1 to the left child of each node, and 0 is assigned to the right child of each node. These temporary position codes are used to define the actual binary position code for each node in the original general tree. The position code of a node on the WAP tree is defined as the concatenation of all temporary position codes of its ancestors from the root to the node itself (inclusive) in the transformed binary tree equivalent of the tree. For example, in figure 2(a), the position code of the rightmost leaf node (*r*: 1) is obtained by concatenating all temporary position codes from path (p: 3) to (r: 1) of the rightmost branch of figure 2(b) to obtain 101111. The transformation to binary tree equivalent is mainly used to come up with a technique for defining and assigning position codes (presented below as Rule 2.1) to nodes of the tree that will accomplish the identification task needed during PLWAP tree mining. It should be stressed that the PLWAP algorithm does not involve physical transformation of a WAP-tree to a binary tree before mining. The tree transformation technique presented in this section is mainly used to define a mechanism for assigning position codes straight to PLWAP tree nodes during their construction. After observing the position code assignment with nodes in figure 2(a), the following properties are defined.

Rule 2.1. *Given a WAP-tree with some nodes, the position 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.*

Property 2.1. *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 o f β, where x is the* ((*number of bits in the position code of α*) + 1). For example, in figure 2(a), (p: 1:1110) is an ancestor of (r: 1:111011) because the position code of (*r*: 1:1110) is 1110, and after appending 1 at the end of 1110, we get 11101, which is equal to the first 5 (i.e., length of r + 1) bits of (r: 1:111011). On the other hand, (r: 1:1110) is

not the ancestor of (r: 1:101111), since after appending 1, the code will be 11101 and is not equal to the first 5 bits position code of (r: 1:101111). Not only can we use the position code to find the ancestor and descendant relationships between nodes, but we can also find whether one node belongs to the right-tree or left-tree of another node. From figure 2(a), it can be seen that node (r: 1:1111) and node (r: 1:111011) are two nodes that belong to two sub trees, which are rooted at (p: 2:111) and (p: 1:1110) respectively. The node (p: 1:1111) belongs to a left-tree of (p: 1:111011) since the fourth bit of (p: 1:111011) is 0, which means the node is extended from the node with position code 1110. The node with position code 1110 is a right sibling of node with 111, which is an ancestor of node (p: 1:1111). Thus, (*p*: 1:111011) is a right-tree of (p: 1:1111).

3.2 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. Create a root node for T;
2. For each access sequence S in the access sequence database BC-WAT do

a) Extract frequent subsequence $S^1 = S_1\ S_2\ \ldots\ldots S_n$, WHERE

$S_1 (1<=I<=n)$ are events in S^1.Let current node point to the root of T.

b) for i=1 to n do ,

if cuurent_node has a child labled S_i by 1 and make cuurent_node point S_i ,

else create anew childnode(S_1:1),make current_node point to the new node,and insert it into the S_i queue

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 3, it can be seen that.
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.

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

	time in sec's at different supports				
Algorithm	2	3	4	5	10
WAP	750	510	330	280	150
mWAP	230	160	110	95	48

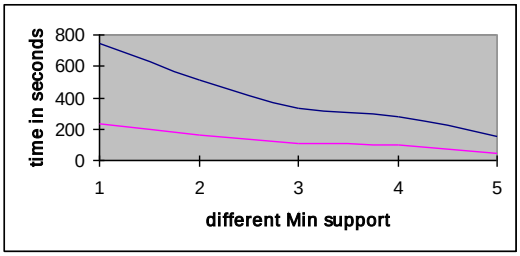


Figure 3. Execution times trend with different minimum supports.

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

Table 3. Execution times trend with different data sizes

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

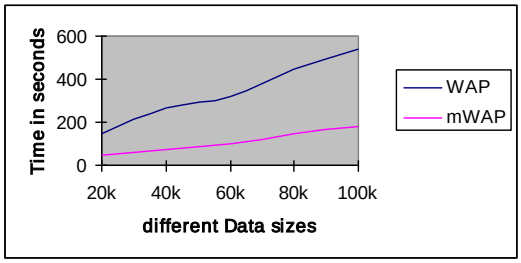


Figure 4. Execution times trend with different data sizes.

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 of binary coded WAPtree 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 reconstructing 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 preorder linked wap-tree. International Journal of Data Mining and Knowledge Discovery (DMKD) Kluwer Publishers, p.5–38, 2005.