Position Coded Pre-Order Linked WAP-Tree for Web Log Sequential Pattern Mining

Yi Lu
School of Computer Science, University of Windsor.
C.I. Ezeife*
School of Computer Science, University of Windsor,
Windsor, Ontario, Canada N9B 3P4
cezeife@uwindsor.ca

Abstract. Web access pattern tree algorithm mines web log access sequences by first storing the original web access sequence database on a prefix tree (WAP-tree). WAP-tree algorithm then mines frequent sequences from the WAP-tree by recursively re-constructing intermediate WAP-trees, starting with their suffix subsequences.
This paper proposes an efficient approach for using the preorder linked WAP-trees with binary position codes assigned to each node, to mine frequent sequences, which eliminates the need to engage in numerous re-construction of intermediate WAP-trees during mining. Experiments show huge performance advantages for sequential mining using prefix linked WAP-tree technique.

keywords: Sequential Mining, Web Usage Mining, WAP-tree Mining, Pre-Order Linkage, Position Codes

* This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

1 Introduction

In sequential pattern mining, an order exists between the items (events) making up an itemset, called a sequence, and an item may re-occur in the same sequence. An example sequence of events that could represent a sequential pattern is the order that customers in a video rental store may follow in renting movie videos, such as: rent "star wars", then "Empire strikes back", and then "Return of Jedi". The measures of support and confidence, used in basic association rule mining for deciding frequent itemsets, are also used in sequential pattern mining to determine frequent sequences and strong rules generated from them.

Sequential pattern mining technique is useful for finding frequent web access patterns [8, 2, 5]. For example, the access information in a simplified web log format could stand for access web site represented as events {a, b, c, d, e, f}. The web log events are first pre-processed, to group them into sets of access sequences for each user identifier and to create web access sequences in the form of a transaction database for mining. The web log sequences in a transaction database, obtained after pre-processing the web log has each tuple consisting of a transaction ID and the sequence of this transaction's web accesses as shown in Table 1. Thus, for example, user ID 100, from the table has accessed sites, *a* then *b*, *d*, *a*, and *c*. The problem of mining sequential patterns from web logs are now

Table 1. Sample Web Access Sequence Database

TID	Web access Sequences
100	abdac
200	eaebcac
300	babfaec
400	afbacfc

based on the database of Table 1. Given a set of events E, the access sequence S can be represented as $e_1 e_2 \ldots e_n$, where $e_i \in E$ $(1 \leq i \leq n)$. Considering the sequences S' = ab, S = babcd, we can say that S' is a subsequence of S. We can also say that ac is a subsequence of S. A frequent pattern is an access sequence to be discovered during the mining process and it should have a support that is higher than minimum support. In sequence eaebcac, eae is a prefix of bcac, while the sequence, bcac is a suffix of eae. The support of pattern S in web access sequence database, WASD is defined as the number of sequences S_i, which contain the subsequence S, divided by number of transactions in the database WASD. Although events can be repeated in an access sequence, a pattern gets at most one support count contribution from one access sequence. The problem of web usage mining is that of finding all patterns which have supports greater than λ, given the web access sequence database WASD and a minimum support threshold λ.

1.1 Related Work

Sequential mining was proposed in [1], using the main idea of association rule mining presented in Apriori algorithm. Later work on mining sequential patterns in web log include the GSP [9], the PSP [4], and the graph traversal [5] algorithms. Pei et al. [6] proposed an algorithm using WAP-tree, which stands for web access pattern tree. This approach is quite different from the Apriori-like algorithms. The main steps involved in this technique are: the WAP-tree 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. Although WAP-tree technique improves on mining efficiency, it recursively reconstructs large numbers of intermediate WAP-trees during mining and this entails expensive operations of storing intermediate patterns.

1.2 Contributions

This paper proposes the Pre-order Linked WAP tree algorithm, which stores the sequential data in a Pre-order Linked WAP tree. Each of this tree's nodes has a binary position code assigned for directly mining the sequential patterns without re-constructing the intermediate WAP trees improving a lot on the efficiency of the WAP tree technique. This paper also contributes the technique for assigning a binary position code to nodes of any general tree, which, can be used to quickly define the ancestors and descendants of any node.

1.3 Outline of the Paper

Section 2 presents the proposed Pre-Order Linked WAP-Tree Mining (PLWAP) algorithm with the Tree Binary Code Assignment rule. Section 3 presents an example sequential mining of a web log database with the PLWAP algorithm. Section 4 discusses experimental performance analysis, while section 5 presents conclusions and future work.

2 Pre-Order Linked WAP-tree Mining Algorithm

PLWAP algorithm is able to quickly determine the suffix trees or forests of any frequent pattern prefix event under consideration by comparing the assigned binary position codes of nodes of the tree. Binary Code Assignment (TreBCA) technique is defined for assigning unique binary position codes to nodes of any general tree, by first transforming the tree to its binary tree equivalent and using a rule similar to that used in Huffman coding [7], to define a unique code for each node.

2.1 Important Concepts for PLWAP-Tree Based Mining

From the root to any node in the tree defines a sequence. Ancestor nodes of a node, e_i in PLWAP-tree form its prefix sequence, while its suffix sequence are the descendant nodes. The count of this node e_i is called the count of the prefix sequence. Each branch from a child of node e_i, to a leaf node represents a suffix sequence of e_i, and these suffix branches of e_i are called the suffix trees (forest) of e_i. The suffix trees of a node, e_i are rooted at several nodes that are children of e_i, called the suffix root set of e_i. The suffix root sets are used to virtually represent the suffix forests without the need to physically store each forest. For example, in Figure 1(a), the suffix trees of node, Root are rooted at nodes (a:3:1) and (b:1:10), while the suffix trees of node (a:3:1) are rooted at (b:3:11). To avoid reconstructing trees, the idea of PLWAP is to use the suffix trees of the last frequent event in an *m-prefix sequence* to recursively extend the subsequence to m+1 sequence by adding a frequent event that occurred in the suffix sequence. Thus, binary *position codes* are introduced for identifying the position of every node in the PLWAP tree.

2.2 Tree Binary Position Code Assignment Rule

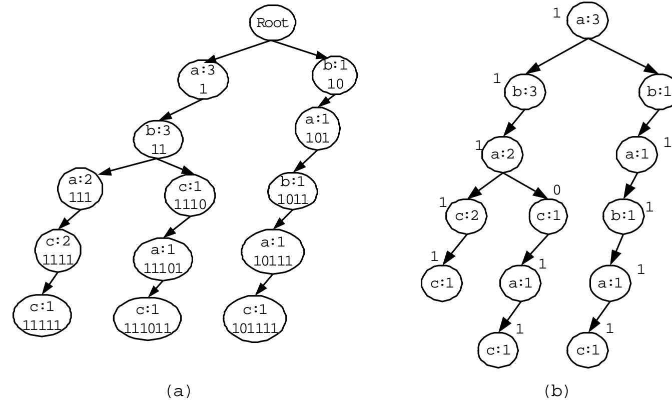

Fig. 1. Position Code Assignment Using Single Codes in its Binary Tree

A temporary position code of 1 is assigned to each left child of a node in the binary tree transformation of a given PLWAP-tree, while a temporary code of 0 is assigned to each right child. The actual position code of a node on the PLWAP tree is, then, 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, given a general tree shown as Figure 1(a), it can be transformed into its binary tree equivalent shown in Figure 1(b). In Figure 1(a), the actual position code of the rightmost leaf node (c:1) is obtained

by concatenating all temporary position codes from path (a:3) to (c:1) of the rightmost branch of Figure 1(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 21).

Rule 21 *Given a PLWAP-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* 2^{n-1} *to the parent's code.* ■

Property 21 *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).* ■

For example, in Figure 1(a), (c:1:1110) is an ancestor of (c:1:111011) because after appending 1 to the position code of (c:1:1110), which is 1110, we get 11101. This code 11101 is equal to the first 5 (i.e, length of c +1) bits of code from (c:1:111011). Position codes are also used to decide if one node belongs to the right-tree or left-tree of another node.

2.3 The PLWAP Algorithm

The PLWAP algorithm is based on the following properties.

Property 22 *If there is a node in an* e_i *current suffix subtree, which is also labeled* e_i*, the support count of the first* e_i *is the one that contributes to the total support count of* e_i*, while the count of any other* e_i *node in this same subtree is ignored.* ■

From Figure 1(a), property 22 allows us to obtain the support count of event a on the left suffix subtree of Root as 3, and that on the right subtree as 1.

Property 23 *The support count of a node* e_i*, in the current* e_i *suffix trees (also called current conditional PLWAP-trees), ready to be mined is the sum of all first* e_i *nodes in all suffix trees of* e_i*, or in the suffix trees of the Root if the first event of a frequent pattern is being mined.* ■

Applying property 23 to Figure 1(a), the support count of node a in the Root suffix trees is 4 (the sum of a:3:1 and a:1:101). The support count of a in subsequent a suffix trees (that is subtrees rooted at b:3:11 and b:1:1011) is 4, from the sum of a:2:1111, a:1:11101 and a:1:10111. The support count of event b in this same a suffix trees rooted at (b:3:11 and b:1:1011) is 4.

Property 24 *e_i is the next frequent event in the mined prefix subsequence if the node e_i in the current suffix tree of e_i has a support count greater than or equal to the minimum support threshold.* ■

Property 25 *For any frequent event e_i, all frequent subsequences containing e_i can be visited by following the e_i linkage starting from the last visited e_i record in the PLWAP-tree being mined.* ■

The sequence of steps involved in the PLWAP-tree algorithm is:
Step 1: The PLWAP algorithm scans the access sequence database first time to obtain the support of all events in the event set, E. All events that have a support of greater than or equal to the minimum support are frequent. Each node in a PLWAP-tree registers three pieces of information: node label, node count and node position code, denoted as label: count: position. The root of the tree is a special virtual node with an empty label and count 0.
Step 2: The PLWAP scans the database a second time to obtain the frequent sequences in each transaction. The non-frequent events in each transaction are deleted from the sequence. PLWAP algorithm also builds a prefix tree data structure, called PLWAP tree, by inserting the frequent sequence of each transaction in the tree the same way the WAP-tree algorithm would insert them. The insertion of frequent subsequence is started from the root of the PLWAP-tree. Considering the first event, denoted as e, increment the count of a child node with label e by 1 if there exists this node, otherwise create a child labeled by e and set the count to 1, and set its position code by applying Rule 21 above. Once the frequent sequence of the last database transaction is inserted in the PLWAP-tree, the tree is traversed pre-order fashion (by visiting the root first, the left subtree next and the right subtree finally), 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, called event-node queue.
Step 3: Then, PLWAP algorithm recursively mines the PLWAP-tree using prefix conditional sequence search to find all web frequent access patterns. Starting with an event, e_i on the header list, it finds the next prefix frequent event to be appended to an already computed m-sequence frequent subsequence, by applying property 24 , which confirms an e_n node in the suffix root set of e_i, frequent only if the count of e_n in the current suffix trees of e_i is frequent. It continues the search for each next prefix event along the path, using subsequent suffix trees of some e_n (a frequent 1-event in the header table), until there are no more suffix trees to search. For example, the mining process would start with an e_i event a, and given the PLWAP-tree, it first mines the first a event in the frequent pattern by obtaining the sum of the counts of the first e_n (i.e, a) nodes in the suffix subtrees of the Root. This event is confirmed frequent if this count is greater than or equal to minimum support. Now, the discovered frequent pattern list is {a}. To find frequent 2-sequences that start with this event a, the next suffix trees of last known frequent event in the prefix subsequence, e_i (i.e., a) are mined for events a, b, c (1-frequent events in the header node), in turn to possibly obtain frequent 2-sequences *aa, ab, ac* respectively if support thresholds

Algorithm 21 *(PLWAP-Mine - Mining the Pre-Order Linked WAP Tree)*

Algorithm PLWAP-Mine()
Input: PLWAP tree T, header linkage table L,
minimum support λ $(0 < \lambda \leq 1)$, Frequent m-sequence F).
suffix tree roots set R (R is root and F is empty first time)
Extendible set L (is frequent 1-sequence set the first time)
Output: Frequent (m+1)-sequence, F'.
Other Variables: S stores whether node is ancestor of the following nodes
in the queue, C stores the total number of events e_i in the suffix trees.
begin
(1) If R is empty, or the summation of R's children is less than λ, return
(2) For each event, e_i in L, find the suffix tree of e_i in T (i.e,$e_e|suffixtree$), do
(2a) Save first event in e_i-queue to S.
(2b) Following the e_i-queue
If event e_i is the descendant of any event in R, and is not descendantof S,
Insert it into suffix-tree-header set R'
Add count of e_i to C.
Replace the S with e_i.
(2c) If C is greater than λ
Append e_i after F to F' and output F'
Call Algorithm PLWAP-Mine of Figure 2 passing R' and F'.
Else Remove e_i from extendible set L
end // of PLWAP-Mine //

Fig. 2. The PLWAP-Tree Mining for Frequent Patterns Algorithm

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. The formal algorithm for PLWAP mining is presented as Figure 2. An example, showing the construction and mining of the PLWAP-tree is given in section 3.

3 An Example - Constructing and Mining PLWAP-Tree for Frequent Patterns

The same web access sequence database used in section 1 for introducing WAP tree mining is used here for showing how the PLWAP algorithm constructs a PLWAP tree and how it mines the tree to obtain frequent sequences with a minimum support of 75%.

3.1 Example Construction of PLWAP Tree

The PLWAP algorithm scans the WASD once to obtain the supports of the events in the event set E = {a, b, c, d, e, f} and stores the frequent 1-events in

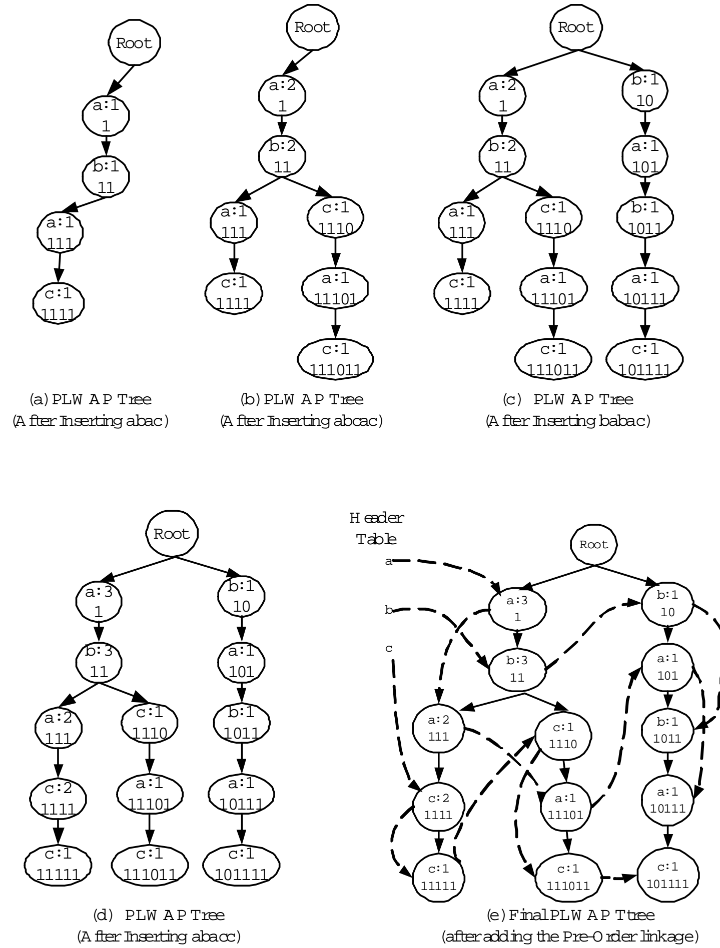

Fig. 3. Construction of PLWAP-Tree Using Pre-Order Traversal

the header List, L as {a:4, b:4, c:4}. The non-frequent events are d:1, e:2, f:2. The PLWAP algorithm next uses only the frequent events in each transaction to construct the PLWAP tree and pre-order linkages of the header nodes for frequent events a, b, and c. Thus, the PLWAP algorithm inserts the first frequent sequence abac in the PLWAP tree with only the root by checking if an a child exists for the root, inserting an a as the leftmost child of the root as shown in Figure 3(a). The count of this node is 1 and its position code is 1 since it is the leftmost child of the root. Next, the b node is inserted as the leftmost child of the a node with a count of 1 and postion code of 11 from Rule 21. Then, the third event in this sequence, a, is inserted as a child of node b with count 1 and position code 111. Finally, the last event c is inserted with count 1 and position code 1111. All inserted nodes on the tree have information recorded as

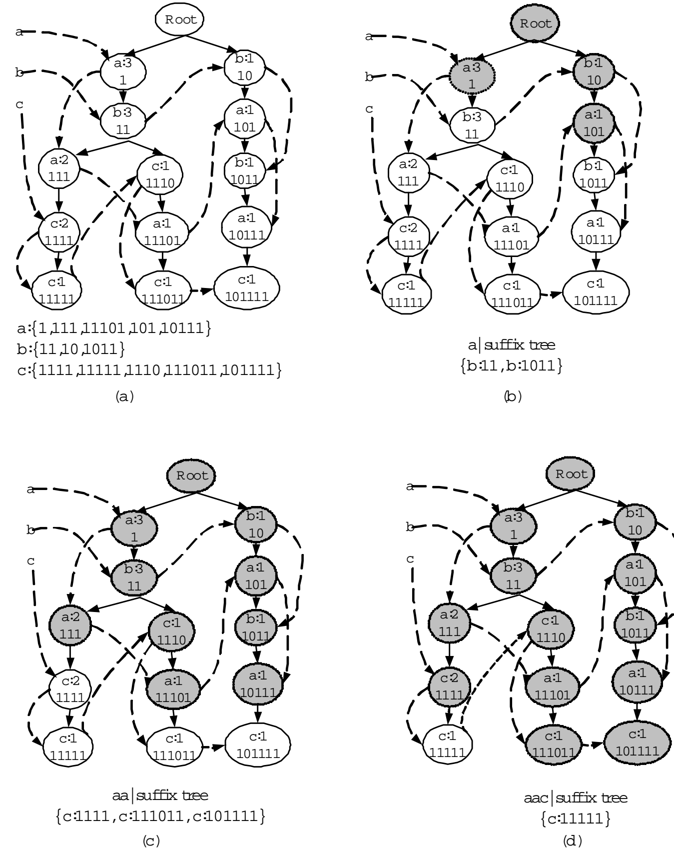

Fig. 4. Mining PLWAP-Tree to Find Frequent Sequence Starting with aa

(node label: count : position code). Next, the second frequent sequence, abcac is inserted starting from root as shown in Figure 3(b). Figures 3(c) and (d) show the PLWAP-tree after sequences babac and abacc have been inserted into the tree. Now, that the PLWAP tree is constructed, the algorithm traverses the tree to construct a pre-order linkage of frequent header nodes, a, b and c. Starting from the root using pre-order traversal mechanism to create the queue, we first add (a:3:1) into the a-queue. Then, visiting its left child, (b:3:11) will be added to the b-queue. After checking (b:3:11)'s left child, (a:2:111) is found and labeled as a, and it is added to the a-queue. Then, (c:2:1111) and (c:1:11111) are inserted into the c-queue. Since there is no more left child after (c:1:11111), algorithm goes backward, until it finds the sibling of (a:2:111). Thus, (c:1:1110) is inserted into c-queue. Figure 3(e) shows the completely constructed PLWAP tree with the pre-order linkages.

3.2 Example Mining of Constructed PLWAP Tree

The algorithm starts to find the frequent sequence with the frequent 1-sequence in the set of frequent events(FE) {a, b, c}. For example, the algorithm starts by mining the tree in Figure 4(a) for the first element in the header linkage list, *a* following the *a* link to find the first occurrencies of *a* nodes in a:3:1 and a:1:101 of the suffix trees of the root since this is the first time the whole tree is passed for mining a frequent 1-sequence. Now the list of mined frequent patterns F is {a} since the count of event *a* in this current suffix trees is 4 (sum of a:3:1 and a:1:101 counts). The mining of frequent 2-sequences that start with event *a* would continue with the next suffix trees of *a* rooted at {b:3:11, b:1:1011} shown in Figure 4(b) as unshadowed nodes. The objective here is to find if 2-sequences aa, ab and ac are frequent using these suffix trees. In order to confirm aa frequent, we need to confirm event *a* frequent in the current suffix tree set, and so on. Using the position codes, we continue to mine all frequent events in the suffix trees of a:3:1 and a:1:101, which are rooted at b:3:11 and b:1:1011 respectively. From Figure 4(b), we find the first occurrence of 'a' on each suffix tree, as a:2:111, a:1:11101 and a:1:10111 giving a total count of 4 to make *a* the next frequent event in sequence. Thus, a is appended to the last list of frequent sequence 'a', to form the new frequent sequence 'aa'. We continue to mine the conditional suffix PLWAP-tree in Figure 4(c). The suffix trees of these 'a' nodes which are rooted at c:2:1111, c:1:111011 and c:1:101111, give another *c* frequent event in sequence, to obtain the sequence 'aac'. The last suffix tree (Figure 4(d)) rooted at c:1:11111 is no longer frequent, terminating this leg of the recursive search. Backtracking in the order of previous conditional suffix PLWAP-tree mined, we search for other frequent events. Since no more frequent events are found in the conditional suffix PLWAP-tree in Figure 4(c), we backtrack to Figure 4(b), to find that b:3:11, b:1:1011 yield frequent event for b to give the next frequent sequence as ab. This leg of mining of patterns that begin with prefix subsequence ab is not shown here due to space limitation. So far, discovered frequent patterns are {a, aa, aac, ab}. The discovered frequent patterns in this *ab* leg are: *aba*, *abac*. Next the algorithm backtracks to Figure 4(b) to mine for the pattern *ac* using the *c* link. This completes the mining of frequent patterns starting with event *a* and the patterns obtained so far are {a, aa, aac, ab, aba, abac, abc, ac}. This process will be repeated in turn for patterns that start with frequent events *b* and *c* respectively. Finally, we have the frequent sequence set {a, aa, aac, ab, aba, abac, abc, ac, b, ba, bac, bc, c}.

4 Performance Analysis and Experimental Evaluation

This section compares the experimental performance of PLWAP, WAP-tree, and the Apriori-like GSP algorithms. The three algorithms are implemented with C++ language running under Inprise C++ Builder environment. All experiments are performed on 400MHz Celeron PC machine with 64 megabytes memory. The operating system is Windows 98. Synthetic datasets are generated using the publicly available synthetic data generation program of the IBM

Quest data mining project at http://www.almaden.ibm.com/cs/quest/, which has been used in most sequential pattern mining studies [9, 6]. The parameters shown below are used to generate the data sets.
$|D|$ Number of sequences in the database
$|C|$: Average length of the sequences
$|S|$: Average length of maximal potentially frequent sequence
$|N|$: number of events
For example, C10.S5.N2000.D60k means that $|C| = 10$, $|S| = 5$, $|N|= 2000$, and $|D| = 60$k. It represents a group of data with average length of the sequences as 10, the average length of maximal potentially frequent sequence is 5, the number of individual events in the database are 2000, and the total number of sequences in database is 60 thousand.

Experiment 1: Execution Time for Different Support

This experiment uses fixed size database and different minimum support to compare the performance of PLWAP, WAP and GSP algorithms. The datasets are described as C10.S5.N2000.D60k, and algorithms are tested with minimum supports between 0.8% and 10% on the 60 thousand sequences database. Table 2

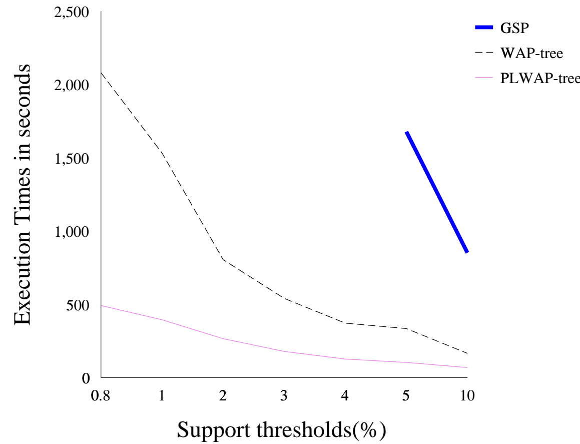

Fig. 5. Execution Times Trend with Different Minimum Supports

and Figure 5 show that the execution time of GSP increases sharply, when the minimum support decreases and that the PLWAP algorithm always uses less runtime than the WAP algorithm.

Experiment 2: Execution Times for Databases with Different Sizes

In this experiment, databases with different sizes from 20k to 100k with fixed minimum support of 5% were used. The five datasets are C10.S5.N2000.D20k, C10.S5.N2000.D40k, C10.S5.N2000.D60k, C10.S5.N2000.D80k and C10.S5.N2000.D100k. The execution times of the WAP and PLWAP algorithms are presented in Table 3 and Figure 6.

Table 2. Execution Times for Dataset at Different Minimum Supports

Algorithms	Runtime (in secs) at Different Supports						
	0.8	1	2	3	4	5	10
GSP	-			-	-	1678	852
WAP-tree	2080	1532	805	541	372	335	166
PLWAP-tree	492	395	266	179	127	104	69

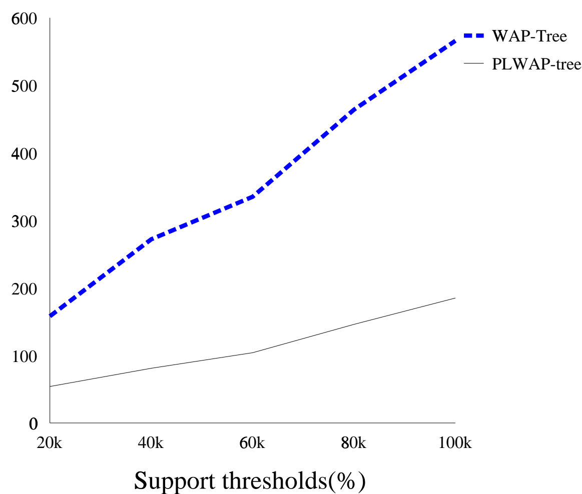

Fig. 6. Execution Times Trend with Different Data Sizes

Experiment 3: Different Length Sequences
In experiment 3 (Table 4), the performance of WAP and PLWAP algorithms for the sequences with varying average lengths of 10, 20 and 30 was observed. Three datasets used for the comparison are: C10.S5.N1500.D10k, C20.S8.N1500.D10k and C30.S10.N1500.D10k. The minimum support is set at 1%.

5 Conclusions and Future Work

The algorithm PLWAP proposed in this paper, improves on mining efficiency, by finding common prefix patterns instead of suffix patterns as done by WAP-tree mining. To avoid recursively re-constructing intermediate WAP-trees, pre-order frequent header node linkages and position codes are proposed. While the pre-order linkage provides a way to traverse the event queue without going backwards, position codes are used to identify the position of nodes in the PLWAP tree. With these two methods, the next frequent event in each suffix tree is found without traversing the whole WAP-tree. Experiments show that mining web log using PLWAP algorithm is much more efficient than with WAP-tree and

Table 3. Execution Times at Different Data Sizes on Support 0.5%

Algorithms	Different Changed Transaction Size				
(times in secs)	20K	40K	60K	80K	100K
WAP-tree	158	272	335	464	566
PLWAP-tree	54	81	104	146	185

Table 4. Execution Times with Different Sequence Lengths at Support 1%

Algorithms	Different Changed Transaction Size		
(times in secs)	10(5)	20(8)	30(10)
WAP-tree	402	1978	5516
PLWAP-tree	217	743	1751

GSP algorithms, especially when the average frequent sequence becomes longer and the original database becomes larger. Future work should consider applying PLWAP-tree mining techniques to distributed mining as well as to incremental mining of web logs and sequential patterns.

References

1. Agrawal, R., Srikant, R.: Mining Sequential Patterns. Proceedings of the 11th International Conference on Data Engineering, Taiwan, 1995.
2. Berendt, B., Spiliopoulou, M.: Analyzing Navigation Behaviour in Web Sites Integrating Multiple Information Systems. VLDB Journal, Special Issue on Databases and the Web, volume 9, number 1, 2000, pages 56-75.
3. Han, J., Kamber, M.: Data Mining-Concepts and Techniques, Morgan Kaufmann Publisher, 2001.
4. Masseglia, F., Poncelet, P., Cicchetti, R.: An Efficient Algorithm for Web Usage Mining. Networking and Information Systems Journal (NIS), volume 2, number 5-6, 1999, pages 571-603.
5. Nanopoulos, A., Manolopoulos, Y.: Mining Patterns from Graph Traversals. Data and Knowledge Engineering, volume 37, number 3, 2001, pages 243-266.
6. Pei, J., Han, J., Mortazavi-Asl, B., Zhu, H.: Mining Access Patterns Efficiently from Web Logs. Proceedings of the 2000 Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00), Kyoto, Japan, 2000.
7. Shaffer, C.A.: A Practical Introduction to Data Structures and Algorithm Analysis. Prentice Hall Inc., September 2000.
8. Spiliopoulou, M.: The Laborious Way from Data Mining to Web Mining. Journal of Computer Systems Science and Engineering, Special Is sue on Semantics of the Web, volume 14, 1999, pages 113-126.
9. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proceedings of the Proceedings of the Fifth International Conference On Extending Database Technology (EDBT), Avignon, France, 1996.