Knowledge Discovery from Web Usage Data: Research and Development of Web Access Pattern Tree Based Sequential Pattern Mining Techniques: A Survey

G. Shivaprasad, N. V. Subbareddy and U. Dinesh Acharya
Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal University, Manipal, India

Abstract—Sequential pattern mining is the process of applying data mining techniques to a sequential database, to extract frequent subsequences to discover correlation that exists among the ordered list of events. Web Usage mining (WUM) discovers and extracts interesting knowledge/patterns from Web logs is one of the applications of Sequential Pattern Mining. In this paper, we present a survey of the sequential pattern mining techniques based Web Access Pattern (WAP) tree on Web logs.

I. INTRODUCTION
Sequential pattern mining, which finds high-frequency patterns with respect to time or other patterns, was first introduced by [3] as follows: given a sequence database where each sequence is a list of transactions ordered by transaction time and each transaction consists of a set of items, find all sequential patterns with a user specified minimum support, where the support is the number of data sequences that contain the pattern. Since the access patterns from web log take on obvious time sequence characteristic, it is natural to apply the technology of sequential pattern mining to web mining. Sequential pattern mining [4] discovers the existing frequent sequences in a given database, when the data to be mined has some sequential nature. These patterns may represent very valuable information. A sequential pattern (as applied to Web usage mining) is defined as an ordered set of pages that satisfies a given support and is maximal (i.e., it has no subsequence that is also frequent). Support is the percentage of the customers who have the pattern. Since a user may have many sessions, it is possible that a sequential pattern should span a lot of sessions. It also need not be contiguously accessed pages.

There exists a great diversity of algorithms for sequential pattern mining. Sequential access pattern mining techniques are mainly based on two approaches: Apriori-based mining algorithms and WAP tree based mining algorithms. Most of the earlier algorithms for sequential pattern mining are based on the Apriori property proposed in association rule mining [3]. The apriori-like algorithms are not efficient when applied to long sequential patterns, which is an important drawback when working with Web logs. In [5], Pei et al proposed another category of algorithm based on Web access pattern (WAP) tree. Several improvements of WAP-tree algorithm, the Pre-order Linked WAP tree and Layer Coded Breadth-First WAP-tree were then proposed. In the next section, we present a brief discussion of these algorithms.

In order to make it easier for us to compare those algorithms, we use the same sequential database, a web log (shown in Table I). 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. The frequent sub-sequences are shown in the 3rd column of Table I are used as input.

II. WAP TREE BASED MINING ALGORITHMS
A. WAP-Mine
The WAP-mine algorithm[5] based on WAP-tree is specifically tailored for mining Web access patterns. The algorithm scans the access sequence database twice. The set of frequent events are determined in the first scan. Using these frequent events, a WAP-tree with count information is built in the next scan. The WAP-tree stores the web log data in a prefix tree format. Then WAP-mine recursively mine the WAP-tree using conditional search to find all Web access patterns. Conditional search narrows the search space by looking for patterns with the same suffix and count frequent events in the set of prefixes with respect to condition as suffix. The process of recursive mining of a conditional suffix WAP tree ends when it has only one branch or is empty.

The WAP-tree algorithm first stores the frequent items as header nodes for linking all the nodes of their type in the WAP-tree in the order the nodes are inserted.

<table>
<thead>
<tr>
<th>TID</th>
<th>Web access sequence</th>
<th>Frequent subsequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>abdac</td>
<td>abdac</td>
</tr>
<tr>
<td>200</td>
<td>cebdac</td>
<td>abcac</td>
</tr>
<tr>
<td>300</td>
<td>dbfac</td>
<td>dbfac</td>
</tr>
<tr>
<td>400</td>
<td>abc afc</td>
<td>abc afc</td>
</tr>
</tbody>
</table>

Initially, a virtual root (Root) is first inserted in the WAP-tree. Then, for each frequent sequence in the transaction, a branch from the Root to a leaf node of the
tree is constructed. Each event in a sequence is inserted as a node with count 1 from Root if that node type does not yet exist. If the node type already exists, then the count of the node is increased by 1. 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. Fig. 1 shows the complete WAP-tree for the sequential data of Table I.

The basic structure of mining all Web access patterns in WAP-tree is as follows: If the WAP-tree has only one branch, return all (ordered) combinations events in the branch; Otherwise, for each frequent event $e_i$ in the WAP-tree by following the $e_i$ queue started from header table, an $e_i$-conditional access sequence base is constructed, denoted as $PS[e_i]$, which contains all and only all prefix sequences of $e_i$. Each prefix sequence in $PS[e_i]$ carries its count from the WAP-tree. For each prefix sequence of $e_i$ with count $c$, when it is inserted in to $PS[e_i]$, all of its sub-prefix sequences of $e_i$ are inserted in to $PS[e_i]$ with count $−c$, to prevent it contributing twice. It is easy to show that by accumulating counts of prefix sequences, a prefix sequence in $PS[e_i]$ holds its unsubsumed count. Then, the complete set of Web access patterns which are prefix sequence of $e_i$ can be mined by concatenating $e_i$ to all Web access patterns returned from mining the conditional WAP-tree, and $e_i$ itself.

For the WAP-tree of Fig. 1, first the prefix sequence of the base c (the lowest frequent event) or the conditional sequence base of c is computed as follows:

aba:2; ab:1; abca:1; ab:−1; baba:1; ab:1; ab−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

Since a is the only frequent pattern in this base, a(4) is used to construct the next WAP tree for the frequent sequence base of bc as shown in Fig. 2(b). Now the reconstruction of WAP trees that with suffix sequences $|c$ and $|bc$ is completed and the frequent patterns found along this line are c, bc and abc. Next, the recursion continues with the suffix path $|c$, $|ac$, $|bac$, $|aac$. Now, the conditional search of c is completed. The search for frequent patterns that have the suffix of other header frequent events are also mined in the same way. After mining the whole tree, the discovered frequent pattern set is: {c, aac, bac, abac, ac, abc, bc, b, ab, a, aa, ba, aba}.

![Figure 1. WAP-tree for the Frequent Sequence in Table I](image1)

![Figure 2. Conditional WAP-tree](image2)
The WAP-tree algorithm scans the original database only twice and avoids the problem of generating explosive candidate sets as in Apriori-like algorithms. So, the mining efficiency is improved sharply. The main drawback of WAP-tree algorithm is that it requires recursive construction of large numbers of intermediate WAP-trees during mining. This necessitates storing intermediate patterns and hence consumes much time and space.

B. Pre-order Linked WAP (PLWAP)

Pre-Order linked WAP tree algorithm [6] assigns unique binary position code to each node of the WAP tree. The header nodes are linked in pre-order fashion. Both the pre-order linkage and binary position codes avoids recursive re-construction of the intermediate WAP trees.

The position code for each node are assigned as follows: the root has null code, and the leftmost child of any parent node has a code that appends ‘1’ to the position code of its parent, while the position code of any other node has ‘0’ appended to the position code of its nearest left sibling. In general, for the \( n \)th leftmost child, the position code is obtained by appending the binary number \( 2^{n-1} \) to the parent’s code.

In PLWAP, first the common prefix sequences are found. The main idea is to find a frequent pattern by progressively finding its common frequent subsequences starting with the first frequent event in a frequent pattern. For example, if pqrs is a frequent pattern to be discovered, the PLWAP technique, will find the prefix event \( p \) first, then, using the suffix trees of node \( p \), it will find the next prefix subsequence \( pq \) and continuing with the suffix tree of \( q \), it will find the next prefix subsequence \( pqr \) and finally, \( pqr \). The binary position codes are introduced for identifying the position of every node in the WAP tree.

The PLWAP algorithm builds a prefix tree, PLWAP tree, by inserting the frequent sequence of each transaction in the tree in the same way as in WAP-Mine. The position codes are set by applying the above mentioned position coding rule. Once the frequent sequence of the last database transaction is inserted in the PLWAP-tree, the tree is traversed in pre-order fashion to build the frequent header node linkages. Event-node queue links all the nodes in the tree with the same label. The event-node queue with label \( e_i \) is also called \( e_i \)-queue. There is one header table \( L \) for a PLWAP-tree, and the head of each event-node queue is registered. Fig. 3 shows the completely constructed PLWAP tree with the pre-order linkages for the data sequence database of Table I.

The algorithm starts by mining the tree in Fig. 3 for the first element in the header linkage list, \( a \) following the \( a \)-link to find the first occurrences of a node 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), and more than the minimum support. 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:1:101\} shown in Fig. 4 as un-shadowed nodes. The objective here is to find if 2-sequences \( aa, ab, ac, b, ba, bac, bc, c \) 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 similarly, to confirm \( ab \) frequent, we should again follow the \( b \) link to confirm event \( b \) frequent using this suffix tree set, same for \( ac \). The process continues to obtain the frequent sequence sets:

\{a, aa, aac, ab, aba, abac, abc, ac, b, ba, bac, bc, c\}

The PLWAP algorithm eliminates the need to store numerous intermediate WAP trees during mining, which drastically cuts off huge memory access costs. The pre-order linking of header nodes makes the search process more efficient. Thus, it saves the processing time and more so when the number of frequent patterns increases and the minimum support threshold is low.

---

**Figure 3. PLWAP Tree adapted from [6]**

**Figure 4. a) Suffix tree**
PLWAP’s performance seems to degrade with very long sequences having sequence length more than 20 because of the increase in the size of position code that it needs to build and process for very deep PLWAP tree

C. Layer Coded Breadth-First WAP-tree

The Layer Coded Breadth First WAP-tree algorithm [7] builds the frequent header node links of the original WAP-tree in a Breadth-First fashion and uses the layer code of each node to identify the ancestor-descendant relationships between nodes of the tree. It then, finds each frequent sequential pattern, through progressive Breadth-First sequence search, starting with its first Breadth-First subsequence event.

Suppose that any ancestor node in WAP tree has not more than n (n<100) children node. Given a WAP-tree with some nodes, the layer code for each node is assigned as follows: the root has a layer code of 0 and the leftmost child of the root has a code of 001, the second leftmost node has a code of 002, and so on. So the layer code of a node in WAP tree is its parent node layer code plus its position (from left to right) 01-99. Also, a node α is an ancestor of another node β if and only if the layer code of α equals the leftmost n bits layer code of β, n is the length of layer code a.

The Pre-Order event linkage [6] can’t reflect ancestor-descendant relationship directly and clearly. In Fig. 4, by following the linkage started in head table, the chain of event a is {a:3:1, a:2:111, a:1:11101, a:1:101, a:1:10111}. In this chain, a:2:111 appears before a:1:101, but a:1:101 is the second level of root and a:2:111 is the third level of root. So in order to find all first descendant of root of event a, we must scan entire chain. Hence, this algorithm employs the Breadth-First traversal mechanism. Breadth-First traversal mechanism makes use of a Queue into which the nodes of the tree are inserted. First all child nodes of root enter the queue from left to right. Then, as per the FIFO (first in first out) rule, the first element of the queue is accessed. After adding it to the corresponding event queue, all of its children nodes are inserted into the queue and then it is deleted from the queue. The remaining elements of the queue are dealt in the same manner. Fig.5 shows the completely constructed BFWAP tree for the sequence database of Table I.

The algorithm starts mining the tree in Fig. 5 for finding frequent 1-sequence by following the a link to find the first occurrences of a node in a:3:001 and a:1:00201 of the suffix trees of the Root. Since the count of event a, in this current suffix trees is 4 (sum of a:3:001 and a:1:00201 counts), and more than the minimum support {a} is listed as frequent pattern. Next, we need to confirm whether the 2-sequences aa, ab and ac are frequent using the suffix trees rooted at a i.e at { b:3:00101, b:1:0020101}. For example, to confirm aa is frequent, we need to confirm that event a is frequent in the current suffix tree set. The layer codes of the nodes are used to identify which nodes of each suffix tree are descendants of the first event node e, in the same subtree. To mine all the frequent events in the suffix trees of a:3:001 and a:1:201 , which are rooted at b:3: 00101 and b:1: 0020101 respectively, we find the first occurrence of 'a' on each suffix tree, as a:2:0010101, a: 1:001010201 and a:1:002010101. The total count of a is 4 making a, the next frequent event in sequence. Thus, a is appended to the last list of frequent sequence 'a', forming the new frequent sequence 'aa'. This way, we continue to mine other frequent events in the suffix trees. Backtracking in the order of previous conditional suffix BFWAP-tree mined, we search for other frequent events the same as above. Finally, we have the following frequent sequence set:

{ a, aa, aac, ab, aba, abac, abc, ac, b, ba, bac, bc, c}.

The BFWAP algorithm makes use of the WAP-tree structure for storing frequent sequential patterns to be mined. However, to improve the mining efficiency, the algorithm finds layered patterns instead of the prefix patterns as done by PLWAP. The Breadth-First linkage provides a way to traverse the event queue without going backwards. The layer codes are used to identify the level of nodes in the BFWAP tree. With these two methods, the next frequent event in each suffix tree is found without traversing the whole WAP-tree. Thus, it avoids re-constructing WAP-tree recursively. It has been experimented in [7], that mining web log using BFWAP algorithm is much more efficient than with WAP-tree and OSP algorithms, especially when the average frequent sequence becomes longer and the original database becomes larger.
III. CONCLUSIONS

Sequential Pattern Mining techniques based on Apriori require multiple scans of the sequence database and generate huge number of candidate sets for long web access sequences. This problem makes these algorithms ineffective and inefficient in dealing with large sequence sets with long sequences, especially with web logs. Experimental results on some limited data sets carried out by researchers indicated that the WAP-tree algorithm is "an order of magnitude faster than conventional Apriori methods". Several modifications were then proposed in order to further improve the speed of the WAP-tree algorithm.

This paper presents a study of different sequential pattern mining algorithms based on web access pattern tree. It is found that the efficiency and effectiveness of these algorithms will deteriorate rapidly when applied to larger log files with long clicking streams. This weakness will become even more serious as the numbers of web pages and users still keep increasing quickly. Therefore, there is a real need for improving the current web usage mining algorithms or developing new efficient ones.

REFERENCES


